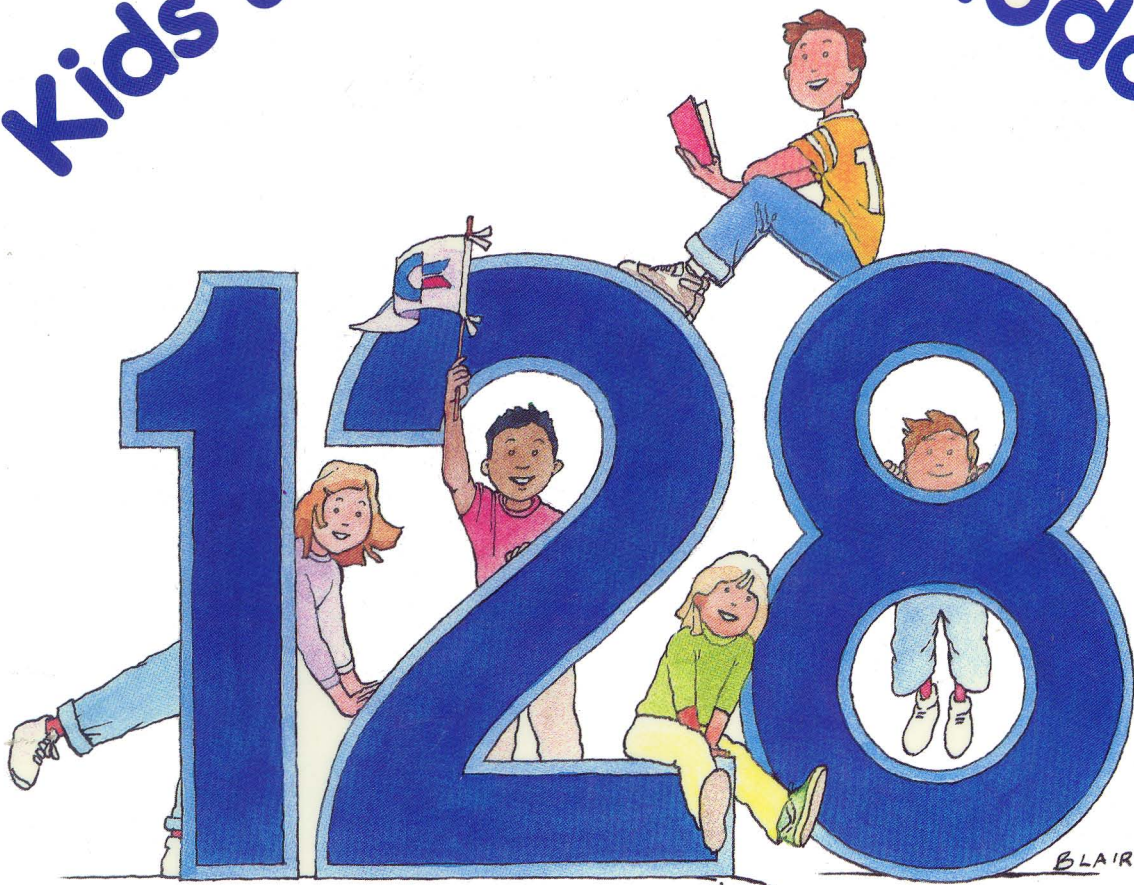


COMPUTE!'s Kids and the Commodore



Edward H. Carlson
Illustrated by Paul D. Trap

For kids ages 8 to 80. This fun, easy-to-use guide to BASIC has everything you need to get started programming on the Commodore 128.

100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

COMPUTE!'s

Kids

and the

Commodore 128



Edward H. Carlson, Ph.D.
Illustrated by Paul D. Trap

COMPUTE! Publications, Inc. 

One of the ABC Publishing Companies

Greensboro, North Carolina

Copyright 1986, 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

10 9 8 7 6 5 4 3 2 1

0-87455-032-7

The author and publisher have made every effort in the preparation of this book to insure the accuracy of the programs and information. However, the information and programs in this book are sold without warranty, either express or implied. Neither the author nor COMPUTE! Publications, Inc., will be liable for any damages caused or alleged to be caused directly, indirectly, incidentally, or consequentially by the programs or information in this book.

The opinions expressed in this book are solely those of the author and are not necessarily those of COMPUTE! Publications, Inc.

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 128 is a trademark 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	9
3. LIST, Boxes in Memory	16
4. The Cursor Keys and Drawing Pictures	23
5. Tricks with PRINT	30
6. The INPUT Statement	36
7. The LET Statement, Gluing Strings	41
8. The GOTO Statement and the RUN/STOP Key	47
9. The IF Statement	56
10. Introducing Numbers	63
11. TAB and Delay Loops	71
12. The IF Statement with Numbers	77
13. Random Numbers and the INT Function	83
14. Save to Disk	90

Graphics, Games, and All That

15. Some Shortcuts	97
16. Moving Pictures	107
17. FOR-NEXT Loops	113
18. DATA, READ, RESTORE	119
19. Sound Effects	125
20. Names, Clocks, and Modes	130
21. Color Graphics	137
22. POKEing Graphics	142
23. Secret Writing and the GET Statement	149
24. Pretty Programs, GOSUB, RETURN, END	155
25. Logic: AND, OR, NOT	161

Advanced Programming

26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN	169
27. Switching Numbers with Strings	176
28. Action Games and the Joystick	182
29. Music	189
30. Arrays and The DIM Statement	195
31. Sprites for Action Graphics	204
32. User-Friendly Programs	210
33. Debugging, STOP, CONT	217

Appendices

Appendix A. Disk Usage	227
Appendix B. Reserved Words	230
Appendix C. Error Messages	232
Appendix D. Glossary	238
Appendix E. Answers to Assignments	250

Commands, Statements, Functions, and Operators Index	275
--	-----

Acknowledgments

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

This book is ninth 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.

I greatly appreciate the skill and energy of Ann Davies and 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" 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 128 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 that you invent.

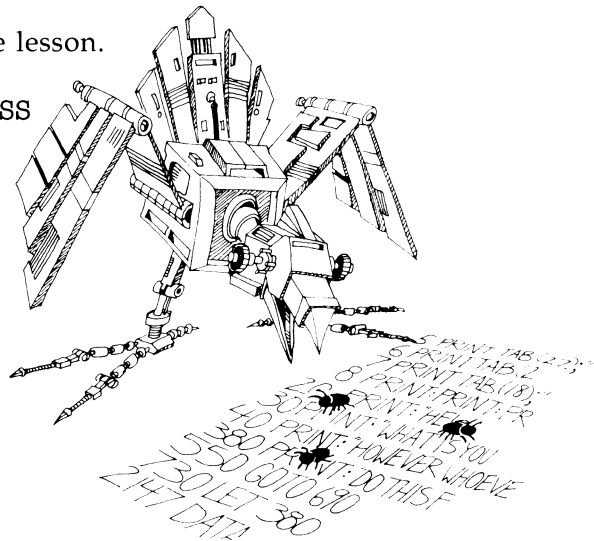
Perhaps your record collection or your paper route needs the organization that your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

You can 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 ask a parent or teacher for help.

Each lesson has review questions.
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 128 to children 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 which you can check later. Answers for program assignments are in the back of the book.

Your child will probably need 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 topics 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 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

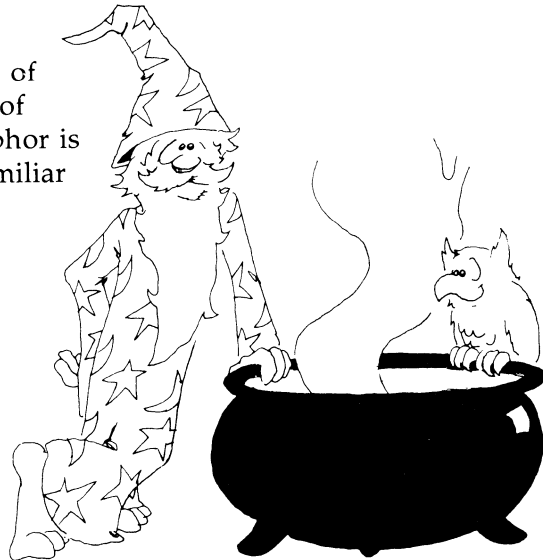
This book is designed for students from the fifth grade up. It teaches BASIC and the features of the Commodore 128.

The lessons contain explanations, 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 you may also use it 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 into 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, choice of examples, and cartoons. Thus, metaphor is used to make 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 and statements. 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 "How I Spent My Summer," writing a program involves skill and planning on several levels. To write an essay, a child must organize thoughts 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, a child may discover that the triangle block, first used to make roofs, makes splendid fir trees. What is needed at the smaller 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 are strengthened. But the strongest features of programming are balanced between analysis (why doesn't it work as I want) and synthesis (planning on several levels, 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

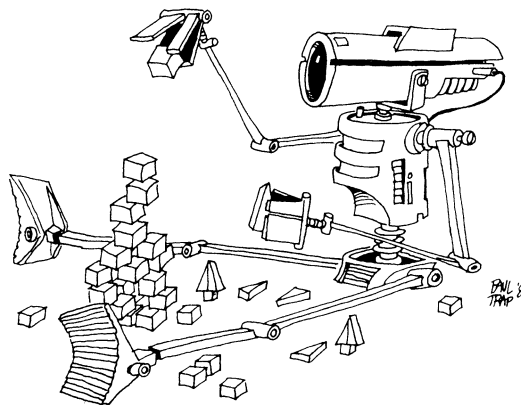
This book is arranged in 33 lessons, each beginning with notes to the instructor and each containing assignments and review questions.

For instructors who feel 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 and leads quickly to a point where interesting programs can be written. The central part of the book emphasizes more advanced and powerful techniques. The final part of the book continues building on these, but also deals with the 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 are solutions for assigned programs, some of them written by children who have used the book.

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



100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

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

This lesson introduces the computer. Your student may have many questions 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.

Be sure the CAPS LOCK, 40/80 DISPLAY, and SHIFT LOCK keys are all "up." If you use a 1902 monitor, the VIDEO MODE switch in its little cubby hole on the front should be in the SEP position.

Here are the contents of the first lesson:

- Turning on the computer; introduction to the keyboard.
- Typing versus entering commands or lines; the RETURN key.
- The computer understands only a limited number of words.
- REM puts remarks in the program.
- What is a program? Numbered lines.
- Clearing the screen.
- White letters on a black background.
- Clearing memory with NEW.
- What is seen on the screen and what is in memory are different (this concept may be hard for students to understand at first).
- RUN makes the computer go to memory, look at the statements in the lines (in order), and perform the statements.
- You can skip numbers when you choose 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 or monitor 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

Turn on the computer. You will see a message on the screen. The last word in the message 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 in something.

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

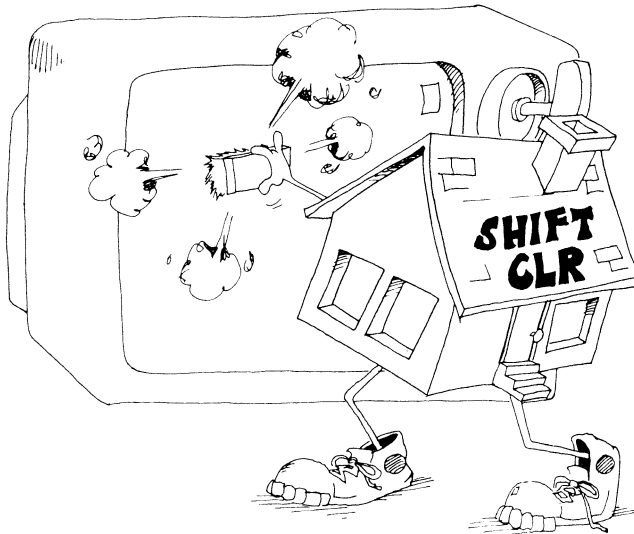
Typing

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

Cursor, Go Home

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

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



Erasing the Screen

Use two keys together to 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."

The Computer Is at Your Command

Type this line and then press the RETURN key:

```
GIVE ME CANDY
```

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 162 words. You need to learn which words the computer understands.

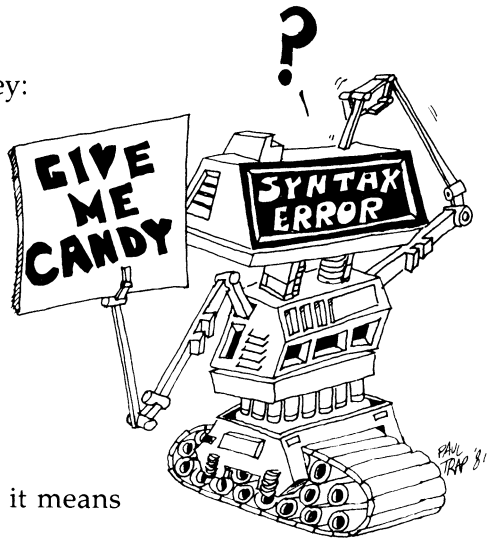
Here are the first four words to learn:

```
NEW, PRINT, REM, and RUN
```

The NEW Command

Type NEW and press RETURN.

NEW empties the computer's memory so that 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 " " around the word HI are quotation marks. To make quotation marks, hold down the SHIFT key and press the key that has the 2 with the " on it.

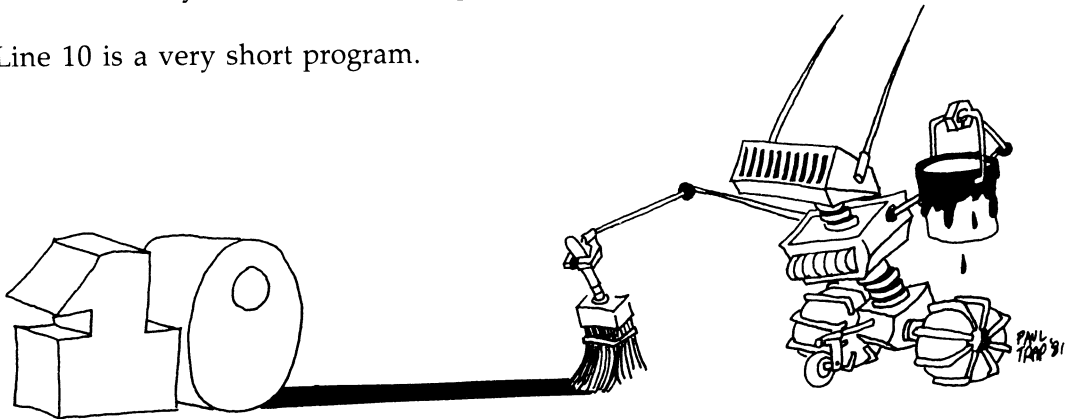
(Did you remember to press the RETURN key at the end of the line?)

Now line 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

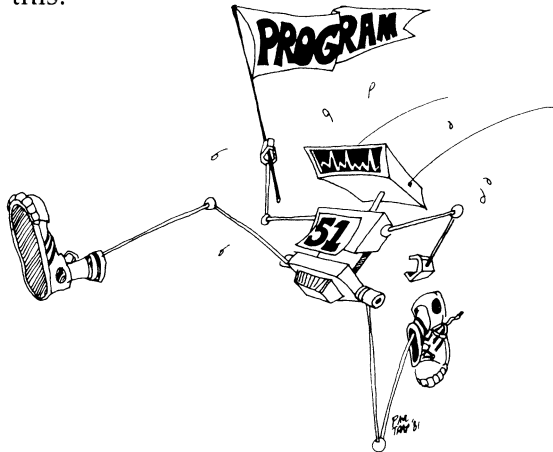
and the letter O like this:

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

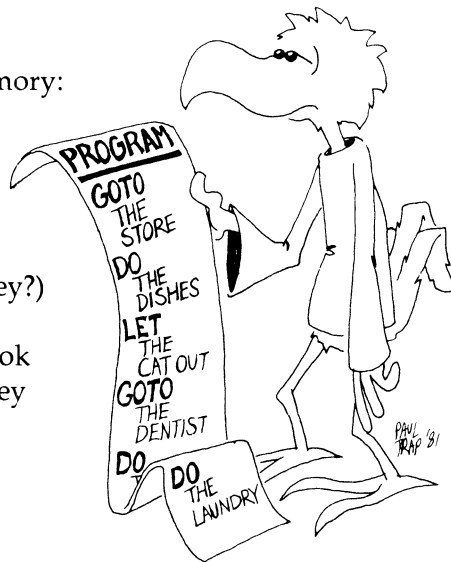
A moment ago you put this program in memory:

```
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. Then enter NEW and type in this program:

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

This program has three lines. Each line starts with a number followed by a one-word 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:

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

It's the same program, but it has different numbers. The numbers are in order, but some numbers were skipped. You skip numbers so that you can add new lines between the old lines if you need to fix the program later.

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

It prints: HI FRIEND

The REM Statement

The REM statement is for writing little notes to yourself. The computer ignores the notes. Use REM to put 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 NEW command. Explain what it does.
3. Write a program that uses REM once and PRINT twice. Then use the RUN command to make the program obey the statements. Explain what it does.

Instructor Notes 2. Color and the Keyboard

The Commodore 128 has powerful color and graphics characters available from the keyboard. They provide the student with plenty of "bells and whistles" to increase program richness.

Each key has up to three functions, chosen by just pressing the key, or by pressing it while holding down the SHIFT, CONTROL, or Commodore key. For colors, the CONTROL 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 128 some very powerful options, and several lessons in the book are devoted to exploiting them.

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

If you choose black for the screen color and also black for the letter color, you will see nothing when you type. Try COLOR 0,16 for a gray screen that will show letters of most of the colors obtained from the color keys.

Pressing the CONTROL key and the Rvs On key 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 CONTROL and Rvs On, then hold down the space bar to make a colored bar. Repeat for each of the colors you get from CONTROL 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 black?

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.

Make a Black Background with Red Letters

Enter: COLOR 0,1

The screen turns black.

Now look at the 3 key. There are four things written on it.

The character 3 in the middle.

The number sign character (#) at the top.

The words Red and L Red (light red) on the front of the key.

Here are the four things this key can do:

Press the 3 key. It prints 3.

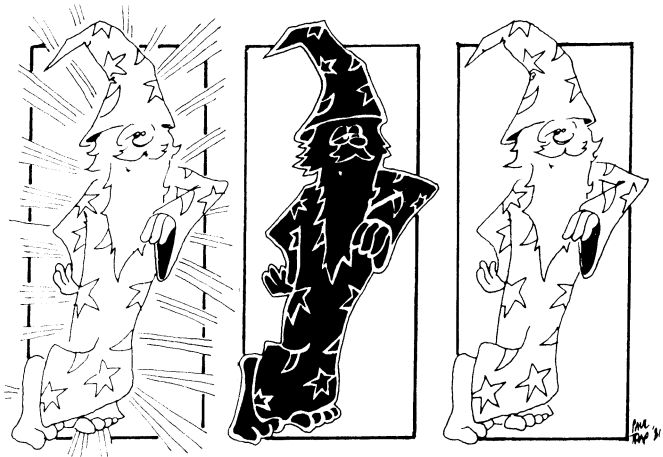
Hold down the SHIFT key and press 3. It prints #.

Hold down the CONTROL key and press 3. It changes the color.

Hold down the Commodore key and press 3. The color becomes light red.

The CONTROL key helps control the color that is printed.

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



More Colors

There are 16 colors on the number keys. They are:

Black	Blk	Orange	Orng
White	Wht	Brown	Brn
Red	Red	Light Red	L Red
Cyan (blue-green)	Cyn	Dark Gray	D Gry
Purple	Pur	Medium Gray	M Gry
Green	Grn	Light Green	L Grn
Blue	Blu	Light Blue	L Blu
Yellow	Yel	Light Gray	L Gry

Try typing letters in different colors. Hold down the CONTROL key or Commodore key and press one of the color keys. Try each color.

Tune the TV for Good Colors

You can tune the color controls on your television or color monitor 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 you want by using a PRINT statement.

Enter the next program, but in lines 20 and 40, do this:

In line 20, press CONTROL and the Red key instead of {RED}
In line 40, press CONTROL and the Yel key instead of {YEL}

```
10 REM RAINBOW
20 PRINT "{ RED } "
30 PRINT "POT "
40 PRINT "{ YEL } "
50 PRINT "OF GOLD "
60 PRINT "{ CYN } "
```

It will look like this on the screen:

```
20 PRINT "#"  
40 PRINT "π"
```

Run the program. It should print POT in red letters and OF GOLD in yellow letters. The cursor will be blue. (CYN in line 60 stands for *cyan*, which is a blue-green color.)

Brace Yourself, Kids

How can you tell when to press CONTROL and a color key in a PRINT statement?

Like this. I'll use capital letters inside braces to mean color keys. Braces look like this: { }

This line: 20 PRINT "{CYN}"
means: 20 PRINT "hold CONTROL key and press Cyn"
and gives 20 PRINT "█" on the screen

But this line: 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	On the screen
10 REM RAINBOW	10 REM RAINBOW
20 PRINT "{GRN}POT"	20 PRINT "█ POT"
30 PRINT "{YEL}OF GOLD"	30 PRINT "π OF GOLD"

Remember, "{GRN}" means hold down the CONTROL key and press the 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
   {SPACE}{YEL} S"
```

I put spaces in line 30 so that 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?

What does line 30 look like on the screen? _____



Other Instructions in PRINT Statements

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

"{CLR}" means SHIFTed CLR/HOME

and "{HOME}" means CLR/HOME.

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

Run this:

```
10 REM MEETING
12 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 this:

```
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 is a different color.

Instructor Notes 3. LIST, Boxes in Memory

In this lesson we will cover:

- LIST, LIST 30
- Memory boxes holding lines
- Erasing one line from memory
- Adding a line between old lines
- Replacing 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's better for a student 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 an incorrect 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. (We'll start each lesson by clearing the 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.

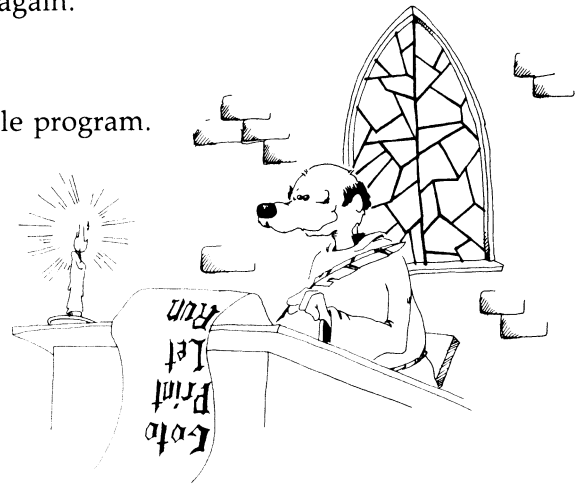
Listing the Program

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
```



Memory

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

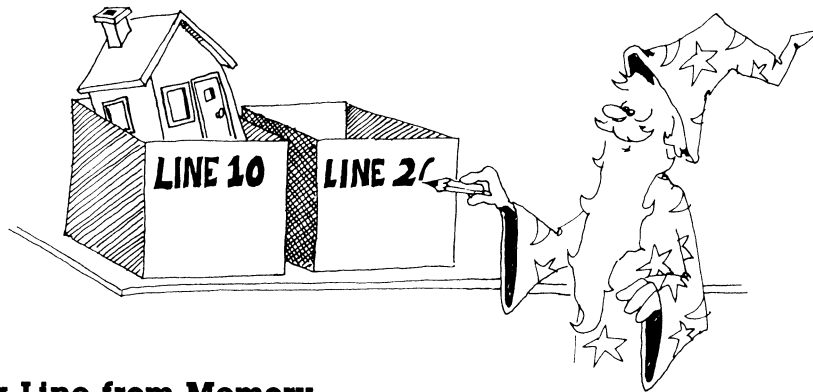
When you entered `10 REM HOUSE`

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 if you do a LIST, 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 the two old lines, and type your line in. The computer puts your new line in the correct place.

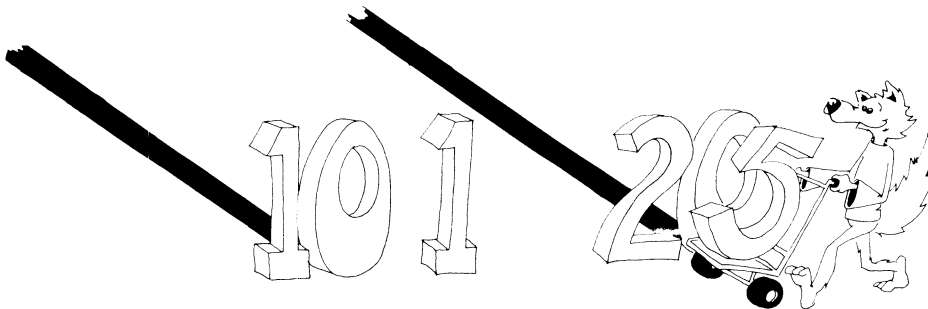
Enter NEW and this:

```
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, you can change line 40 in the program above by entering:

```
40 PRINT "DOGIE"
```

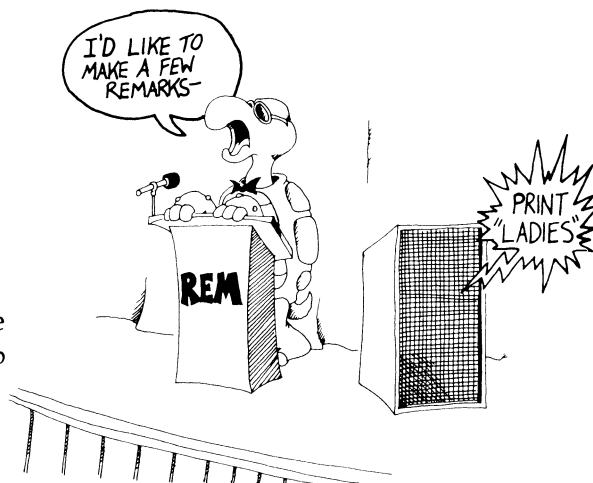
What did the computer do to the box named Line 40 when you entered the line?

The REM Statement

Enter NEW and this program:

```
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 little notes in the program that can help a user understand the program.



Using the CLR/HOME Key in a PRINT Statement

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

Run this:

```
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 CONTROL 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 "XXXXXXXXXXXX"
50 PRINT " O{7 SPACES}O"
```

Spaces are part of the drawing. Whenever you see braces, { }, it means that 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:

```
50 SPACE P R I N T SPACE " SPACE O SPACE SPACE SPACE SPACE SPACE  
SPACE SPACE O "
```

Don't forget the space in line 30 before the X's.



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 move 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 ones.

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, select the right graphics symbol 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 `COLOR 0,C` (C is a number from 1 to 16). The change screen color statement can be given in a program line.

Be careful not to 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. 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.

Lesson 4. The Cursor Keys and Drawing Pictures

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 just above the space bar on the right. 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.

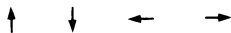
Careful: We do not mean any of these arrow keys:



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.

There are four more cursor keys at the top of the keyboard. They are dark in color and have these arrows on them:



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

Try 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's under the RUN/STOP key and looks like a large C with a flag flying from its side (Ⓒ).

Here's how 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.)

Now 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, L, and @ 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: 10 REM ZRAGON

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

Type a D instead.

Now the line is correct and looks like this:

```
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:

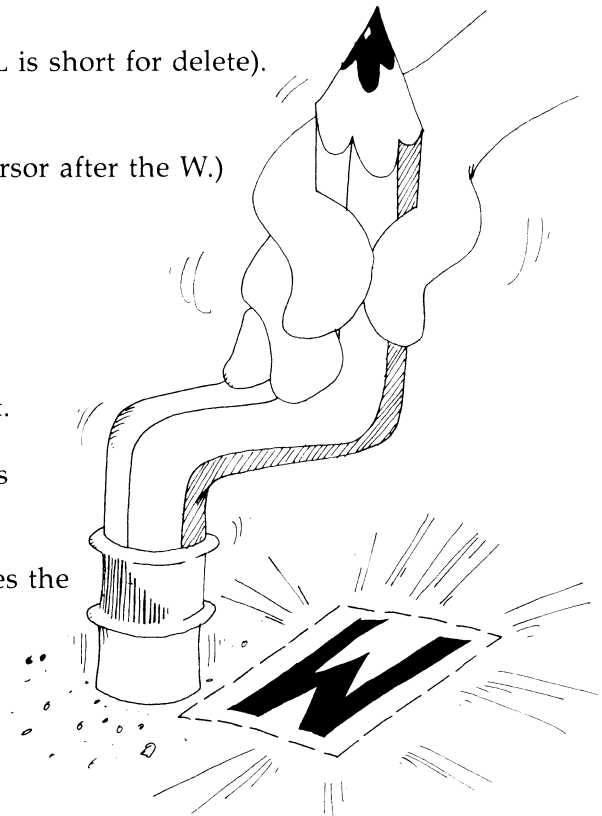
```
20 PRINT "HI THERW" (Leave the cursor after the W.)
```

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's 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. Be careful: You may erase more than you want.

Copycat Lines

Enter:

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

Run the program and list it. Then use the cursor arrow keys to move the cursor onto the 2 of line 20. Type 3, and then press the RETURN key.

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

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

Colored Screens

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

Black: 10 COLOR 0,1

White: 10 COLOR 0,2

Choose another color by picking another number instead of 1 or 2. You can use any number from 1 to 16.

Enter:

```
10 REM RED FLASH
20 PRINT "{WHT} HI THERE"
30 COLOR 0,3
```

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

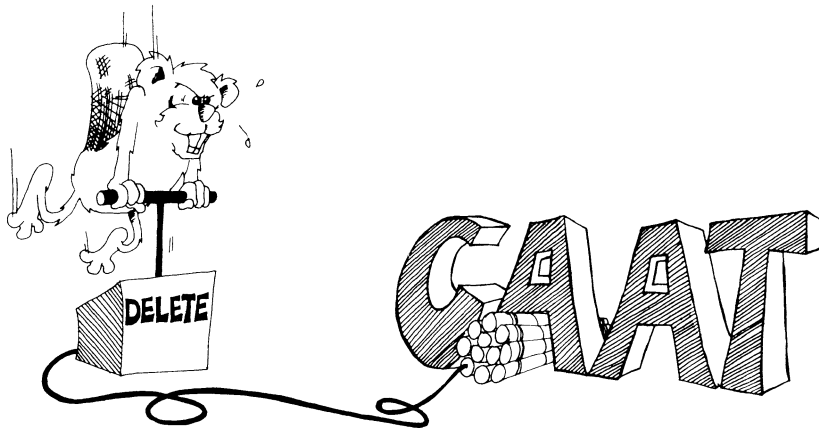
Assignment 4

1. Try making different-colored screens and different-colored letters on each screen. Which combinations look best?
2. Practice using the INST/DEL and the CRSR keys. Type these lines and fix them:

```
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

This 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 numeric constants, variables, and expressions. Numeric 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

Enter this program and run it:

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

Each PRINT statement prints a separate line.

Now run the cursor up onto lines 30 and 40 and change them to:

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

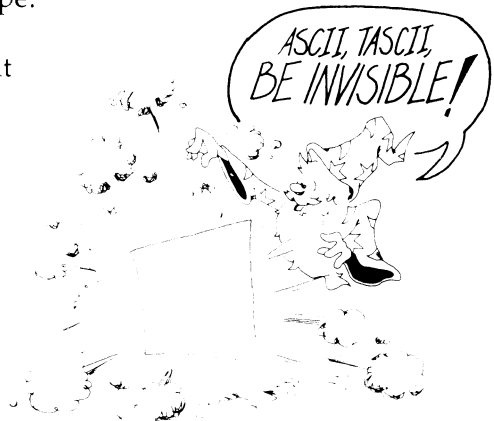
Don't change or erase the other lines. Be careful to put the space at the end of "TOAST " and at the end of "AND " and the semicolon at the end of each line. (Don't forget to press RETURN after you change each line.)

Run it again. What was different from the first time?

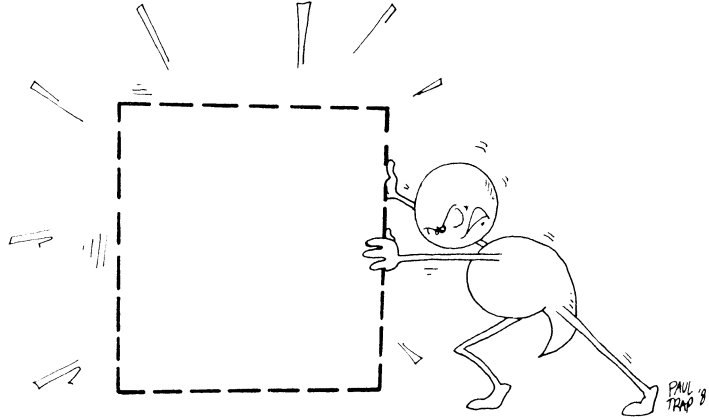
The Hidden Cursor

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 onto what has already been written on the same line.



Famous Pairs

The PRINT statement can print several strings, one after another. You need to put a semicolon (;) 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. We'll explain the INPUT statement 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 run it, try this, too:

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

Rule: A comma between items in a PRINT statement puts spaces between the items 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 fronts of the keys are characters, too. They are called *graphics characters*.

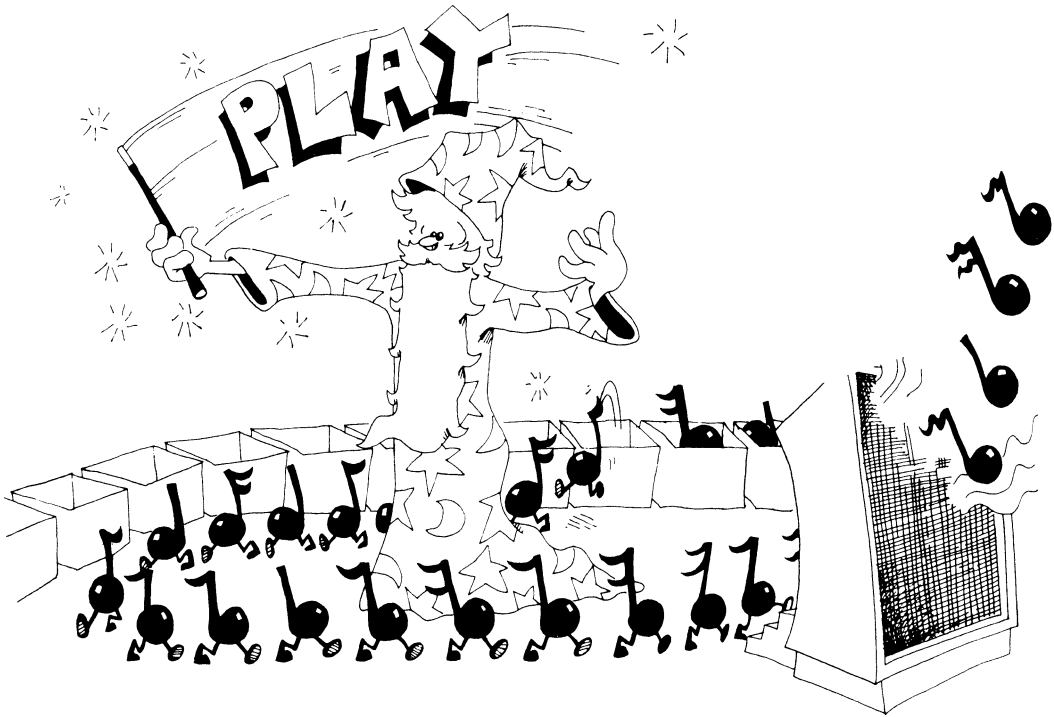
String Constants

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's a string because it is made of letters, numbers, punctuation marks, and graphics characters in a row. It's 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 thing, 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 the statement's simplest form, INPUT A\$, there is no message in quotes in front. We want students to concentrate on the central feature of an INPUT.

Similarly, we will give only the essential feature of each statement in the first section of the book (through lesson 14). We want students to see the forest before going into details. Creating interesting programs requires these five instructions:

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 we get 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.

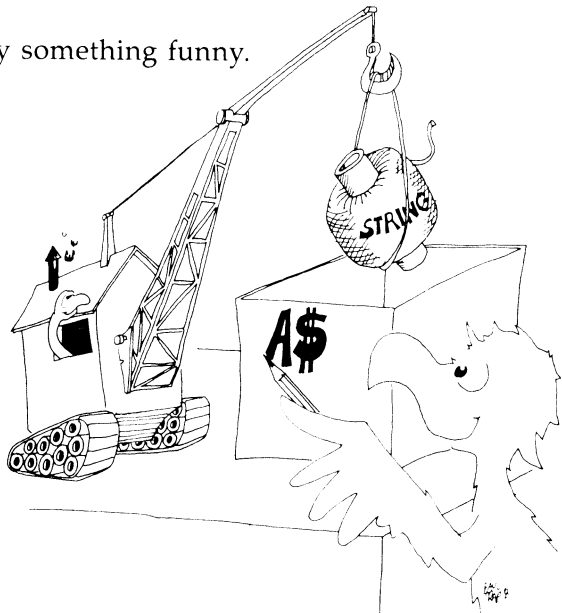
When you enter HI, the computer stores this word in a box named A\$.

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.

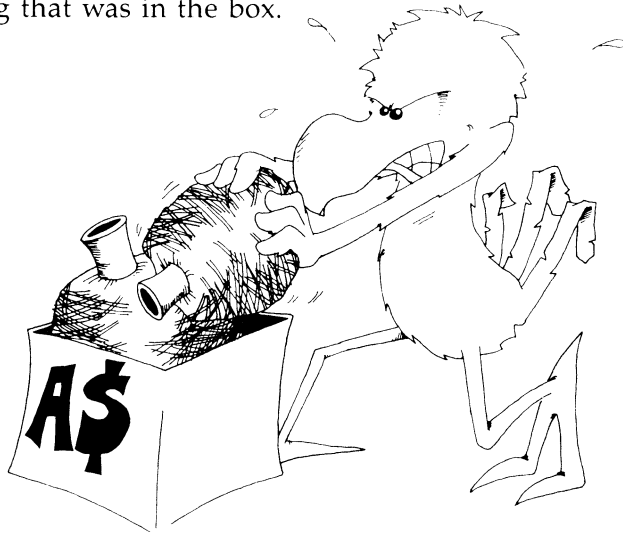
Error Messages from INPUT

Run this three times:

```
10 REM MISTEAKS???  
12 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 a comma or colon 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

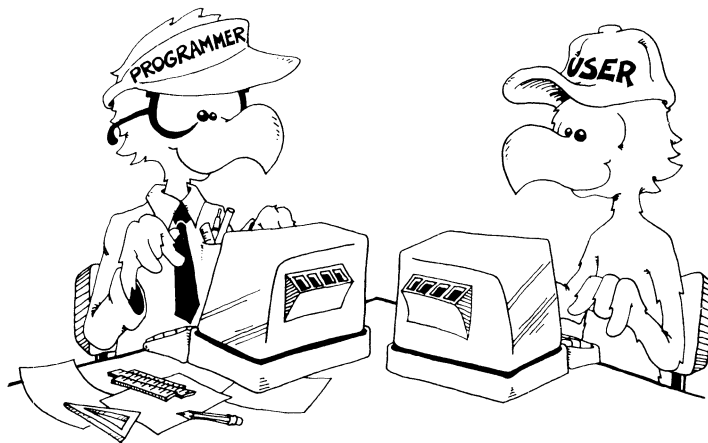
You are a *programmer* when you write a program. The person who runs the program is a *user*.

Of course, if you run your *own* program, then you are a user as well as a programmer.

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's 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 asks for the the user's favorite color—put it in a box called C\$. Then let it ask 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

This lesson introduces the LET statement and concatenation.

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. When LET puts new contents in a box, the old contents are automatically erased first.

These are the statements used so far:

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

Special keys discussed so far:

RETURN, CRSR arrows, SHIFT, CLR/HOME, CONTROL, 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, what will be in box A\$ at the end? What will be in box B\$?

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

-
-
3. In this next program, what is "MOM" called? What is the name of the string variable in the program? What is the value of the string variable after the program runs?

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

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

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

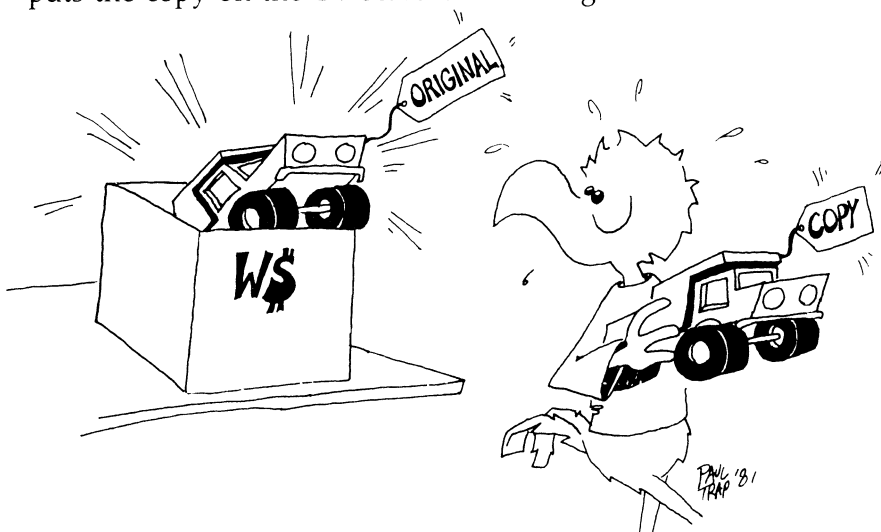
Lesson 7. The LET Statement, Gluing Strings

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

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

Here is what the computer does:

- Line 10 This is a REM statement that gives the name of the program. The computer ignores it.
- Line 12 The computer clears the screen.
- Line 20 It sees that a box named W\$ is needed. It looks in its memory for this box. It doesn't find it, because W\$ has not been used in this program before. So it takes an empty box and writes W\$ on the front. Then it 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

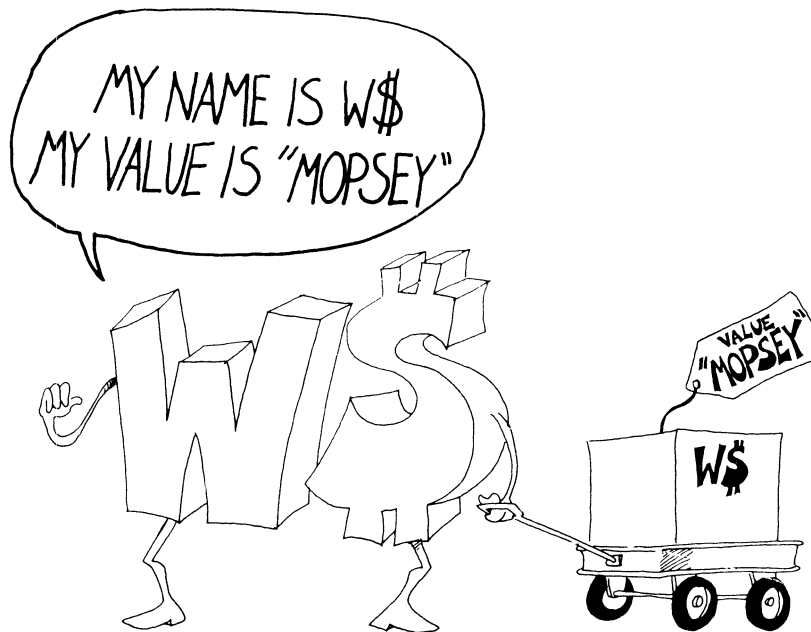
This line makes a string variable:

```
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 REM PICKLES  
12 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 _____

12 _____

20 _____

30 _____

35 _____

40 _____

50 _____

Gluing the Strings

You can stick two strings together to make a longer string. Enter this program:

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



Before you run it, 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 plus 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 the flow of statement execution once we introduce the IF statement. It provides a slow and easy entrance 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 reaches 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. The student looks at some examples of spaghetti programs. Although some fun is had with them, make sure the student is aware 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. In the next lesson, we'll discuss the IF statement, which will change the computer from a mere record player into a machine that can evaluate situations and make decisions accordingly.

Questions

1. What will appear on the screen when this program is run?

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

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

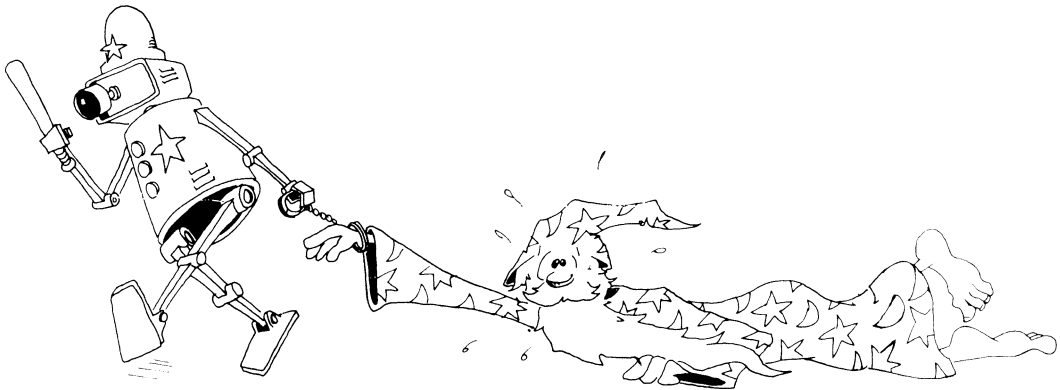
Enter this program.

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

Now run it. 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 just above the Commodore key.

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

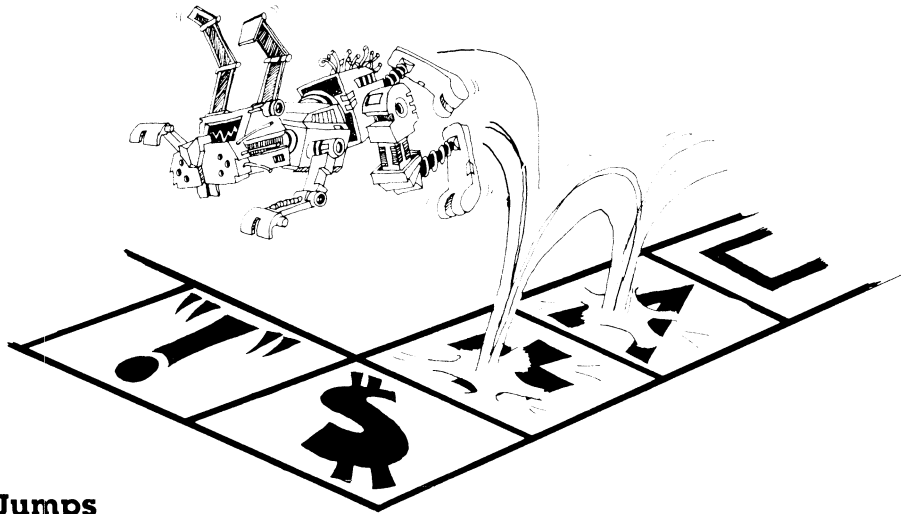
Enter this program:

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



When you run it, type an answer every time you see the question mark and the flashing cursor. Jiggle the RUN/STOP and RETURN keys to end the program.

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



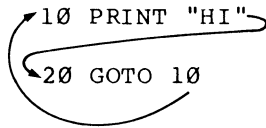
Kinds of Jumps

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

Jumping back gives a *loop*. Look at this:

```
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 when we discuss the IF statement in the next lesson.

The RUN/STOP Key

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

If you're 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.

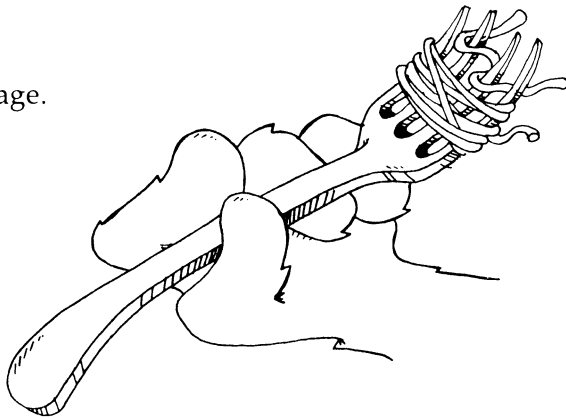
A Can of Spaghetti

Look at the program on the next page.

This is not a good, clear program.

It's a "spaghetti" program.

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



```
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!"
```

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, or INST/DEL key, you will see funny characters on the screen.

Assignment 8

1. Just for practice in understanding the GOTO statement, draw the road map for the “Forked Tongue” spaghetti program on the next page.
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 line, 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.

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

```
20 GOTO 40
```

```
30 PRINT "N"
```

```
31 GOTO 60
```

```
40 PRINT "S"
```

```
41 GOTO 30
```

```
50 PRINT "E"
```

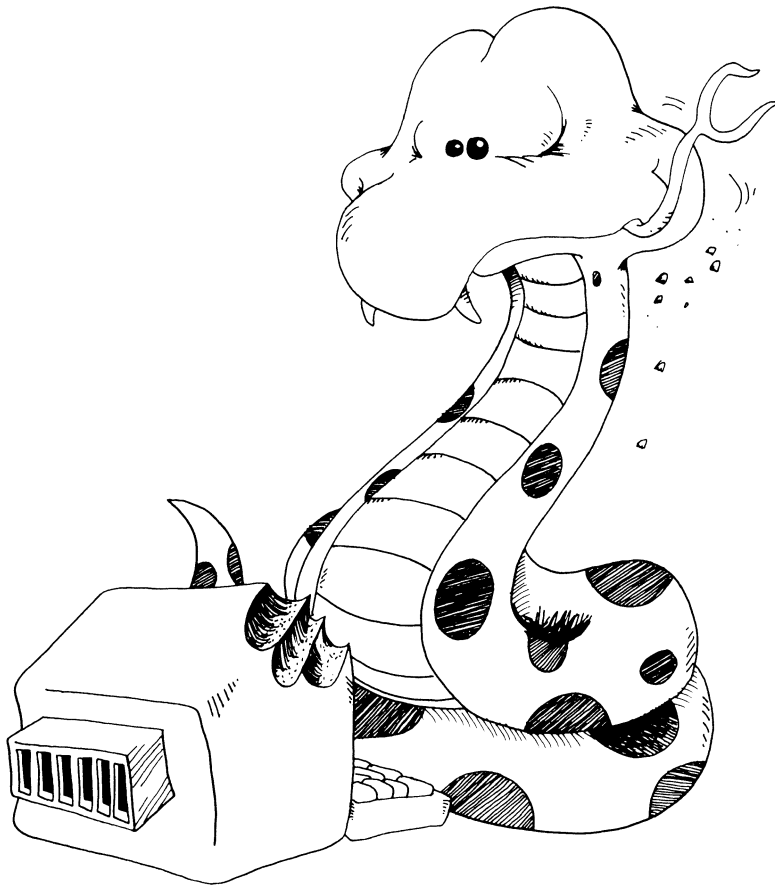
```
51 GOTO 99
```

```
60 PRINT "A"
```

```
90 PRINT "K"
```

```
91 GOTO 50
```

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

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 your 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*:

$$A\$ = B\$$$

$$C\$ \langle \rangle D\$$$

On the next level, 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 that asks if you like chocolate or vanilla ice cream. The answer is 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 like this:

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

Assignment 9A

Clear the computer's 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.

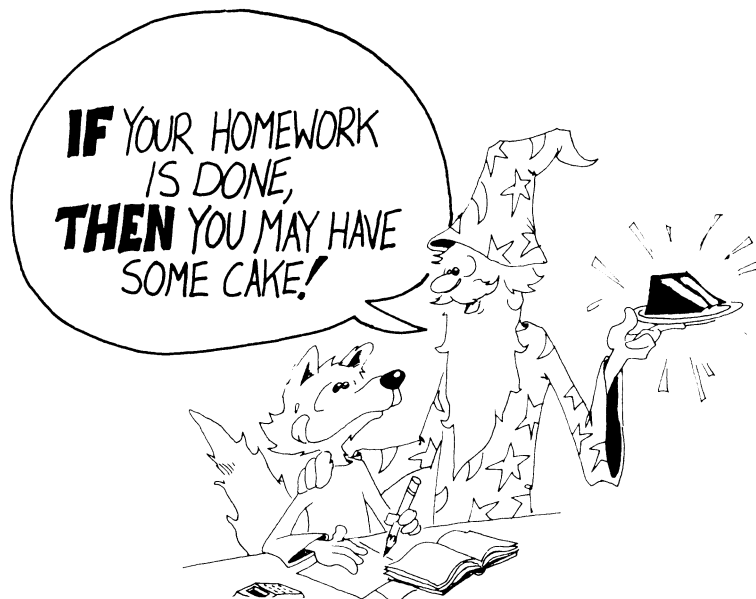
IF in English and in BASIC

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

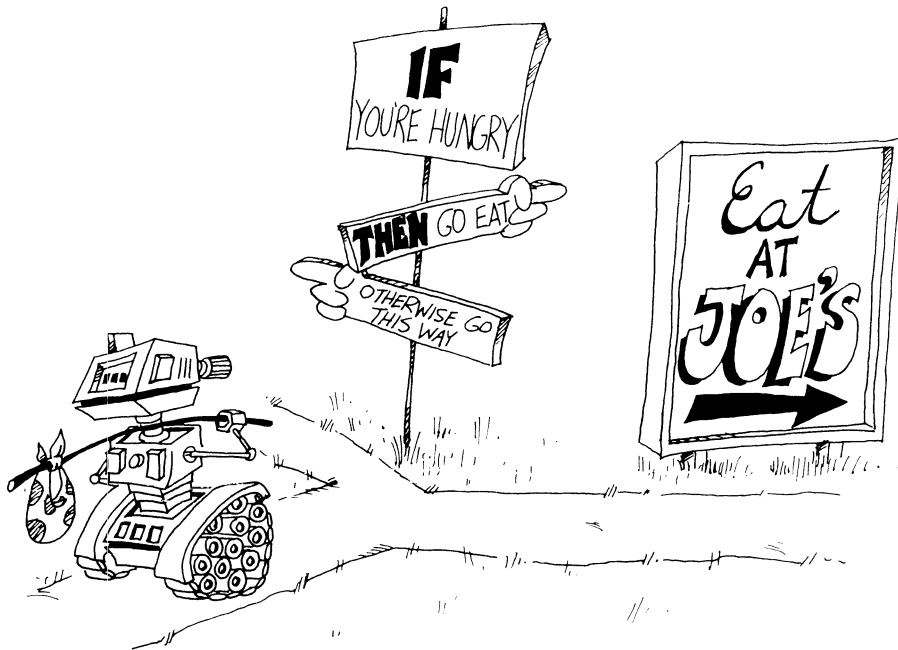
When the computer sees IF, it must choose which road to take.

IF *phrase A* is true, it must go past 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:

30 ↘
40 IF A\$="HUNGRY" → THEN PRINT "EAT"
50 ↘

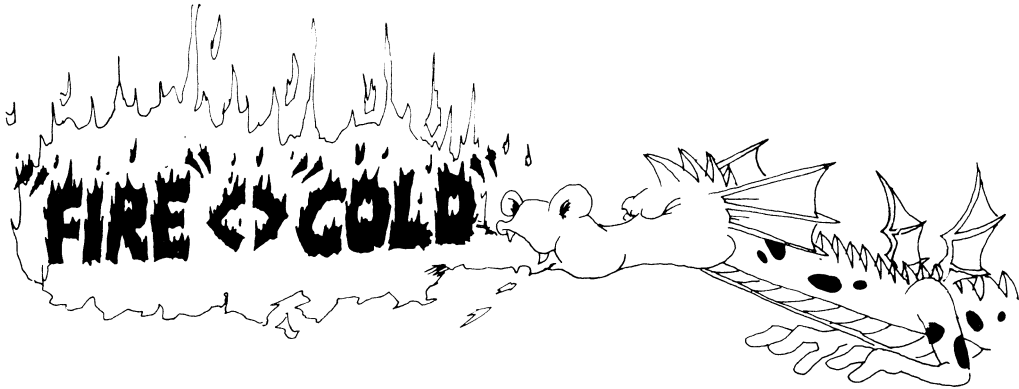


The Not Equal Sign

= means equal.

<> means not equal.

To make the <> sign, hold down the SHIFT key and press the < key, then the > key. To find these keys, look just above the space bar on the right side of the keyboard.



Using the <> Sign

Let's look at an example:

```
40 IF phrase A THEN statement C
```

Phrase A is a phrase that is true or false.

Choose this for *phrase A*: B\$<>"FIRE"

Put it in an IF statement:

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

```
IF        box B$ 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.

On the other hand,

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

Here's how it looks in a program:

```
10 REM PET
11 PRINT
12 PRINT "WITH DOGS IT'S A COLD NOSE"
20 PRINT "WITH DRAGONS, IT'S..."
21 PRINT
25 PRINT "HOW IS YOUR DRAGON'S BREATH?"
26 PRINT
28 PRINT "(ENTER 'FIRE' OR 'COLD')"
```

Assignment 9B

1. Write a "Pizza" program. Ask what topping is wanted. Make the computer answer something silly for each 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

Here, we introduce numeric variables and operations and revisit the LET, INPUT, and PRINT statements. 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 your student. Division will produce decimal numbers, but since most arithmetic will be addition and subtraction and a little multiplication, familiarity with decimal numbers is helpful but not essential.

Students may find it strange 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 numeric values can be printed by PRINT.

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

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

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

Here is another idea from arithmetic that doesn't work in a LET:

In arithmetic, $N = 3$ means the same as $3 = N$.

In 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 / sign.
 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 numeric variables. For string variables.

Lesson 10. Introducing Numbers

So far, we have used only strings. We can use numbers, 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
```

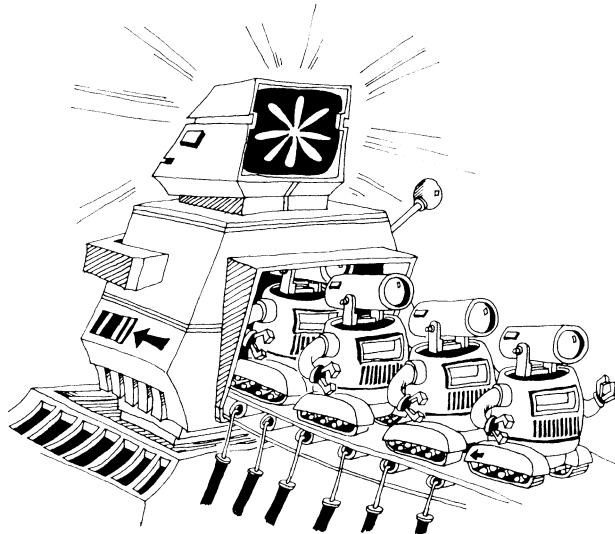
Arithmetic

The plus and minus signs are side by side on 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.

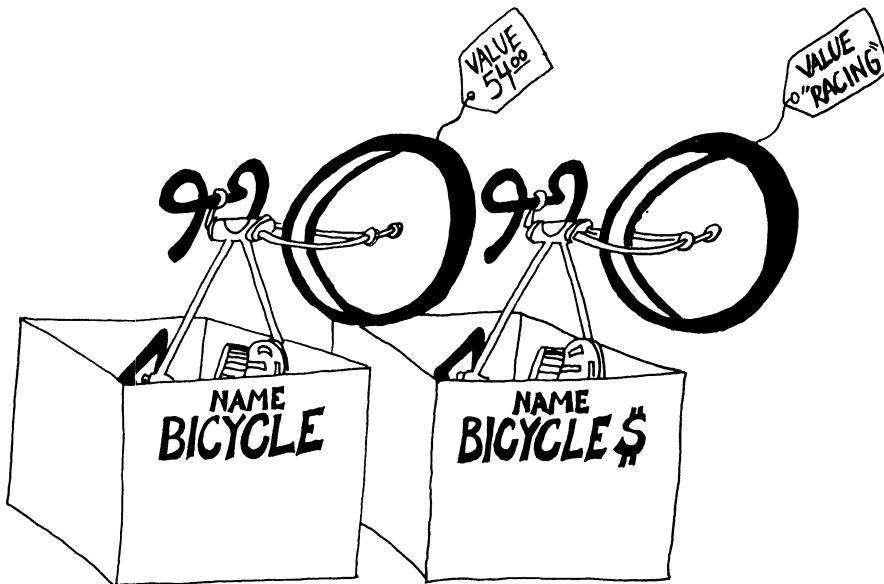


Variables

The name of a box that contains a string must end with a dollar sign. Some examples are N\$, A\$, and Z\$.

The name of a box that contains a number doesn't have a dollar sign. Here are some examples: N, A, and Z.

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



Arithmetic in the LET Statement

Take a look at this program:

```
10 LET A=2001
20 LET B=1985
30 LET C=A-B
40 PRINT "HOW MUCH LONGER, HAL?"
50 PRINT C;"YEARS"
```

Be careful.

Numbers and strings are different. Now look at this example: "1985" is not a number. It is a string constant because it is inside quotation marks.

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

Some numeric constants are 5, 22, 3.14, and -50.

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

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

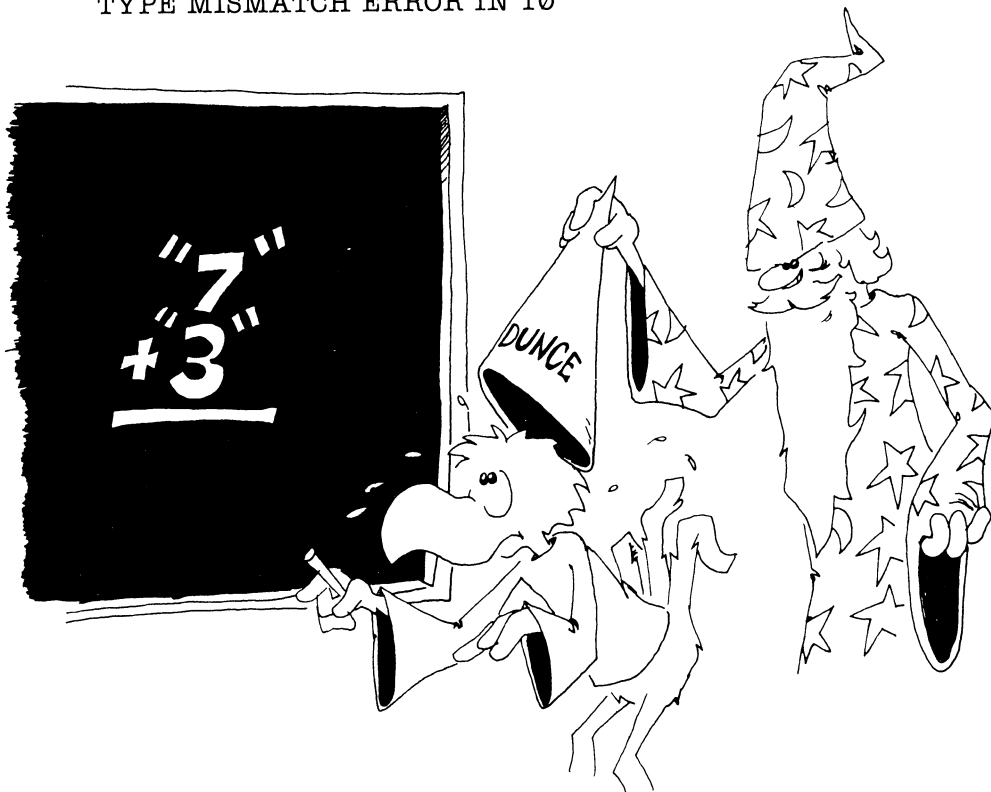
Correct: 10 LET A = 3 + 7

Incorrect: 10 LET A\$ = 3 + 7

Incorrect: 10 LET A = "3" + "7"

If you run either of these incorrect 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, but 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

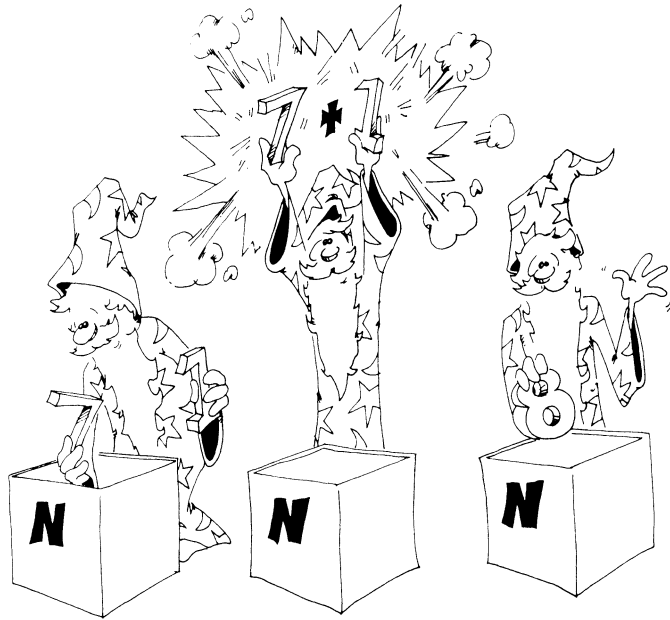
The = sign in computing does not exactly mean *equal*. Look at this line:

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

This doesn't make sense in arithmetic. Suppose N is 7. It would say that

$$7 = 7 + 1$$

which is not correct.



But it's 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.

Here's another way to say the same thing:

10 LET N = N + 1

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

Don't Be Backward

In arithmetic, you can put the two numbers on whichever side of the equal sign you want. But in a LET statement you cannot.

In arithmetic: $N = 3$ is the same as $3 = N$

In BASIC: $LET\ N = 3$ is correct
 $LET\ 3 = N$ is incorrect

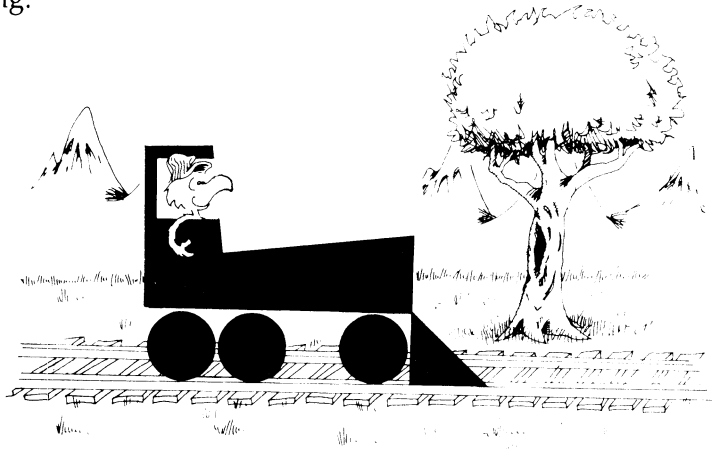
In BASIC: $LET\ N = B$ is not the same as
 $LET\ B = N$ Why not? (What is in each box
 after the line runs?)

$LET\ N = B$ means _____

$LET\ B = N$ means _____

Assignment 10

1. Write a program that asks for your age and the current year. Then subtract and print out the year of your birth. Be sure to use PRINT statements to tell what is wanted and what the final number means.
2. Write a program that asks for two numbers and then prints out their product (multiplies them). Be sure to use plenty of PRINTs to tell the user what is happening.



Instructor Notes 11. TAB and Delay 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 inside the parentheses, (), must increase each time. That is, TAB cannot be used to move the cursor back to the left.

It's not always necessary to use a semicolon between TAB and the thing to be printed, but it is recommended.

The Commodore 128 has a more general and powerful way of moving the cursor around. 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. Delay loops slow a program down so that you can observe its operation more easily. You can also use delay loops for parts of a program that must run at certain speeds. These loops are called *timing loops*.

The delay loop is all on one line, with a colon to separate the NEXT statement. The size of the loop variable determines the length of the delay. A value of 500 gives about a one-second delay.

When students see 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 them 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

TAB() in a PRINT statement is like the TAB on a typewriter. It moves the PRINT cursor 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() by itself.

You Cannot TAB Backward

Now try this:

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

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

Your Name Is Falling

In this program, put your own first name in line 20 between the quotation marks.

```
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 GOTO 40
```

Press RUN/STOP to stop the program.

This program prints your name diagonally down the screen, from top left to bottom right. Try other values of N. Try changing lines 15 and 50:

```
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 from 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.
```

(XX is the line number.)

You can use TAB() with strings, too.

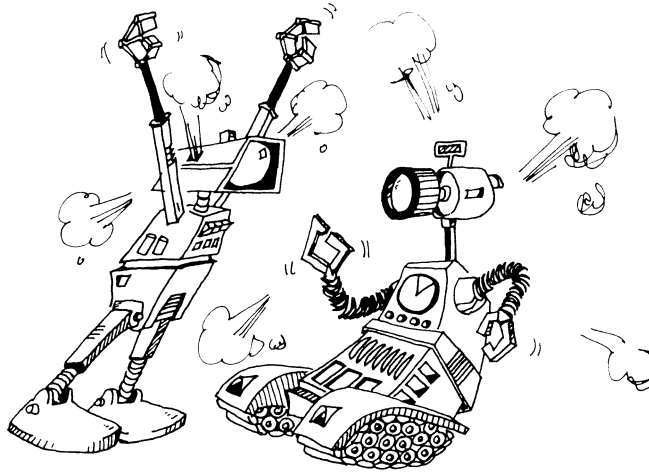
Look at this example:

```
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(), and LEFT\$() later in the book. The number inside the parentheses is called the *argument of the function*. TAB() says "move the cursor over," and the argument tells where to move it.



Assignment 11A

1. Write a program that asks for last names and nicknames. Make it 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.

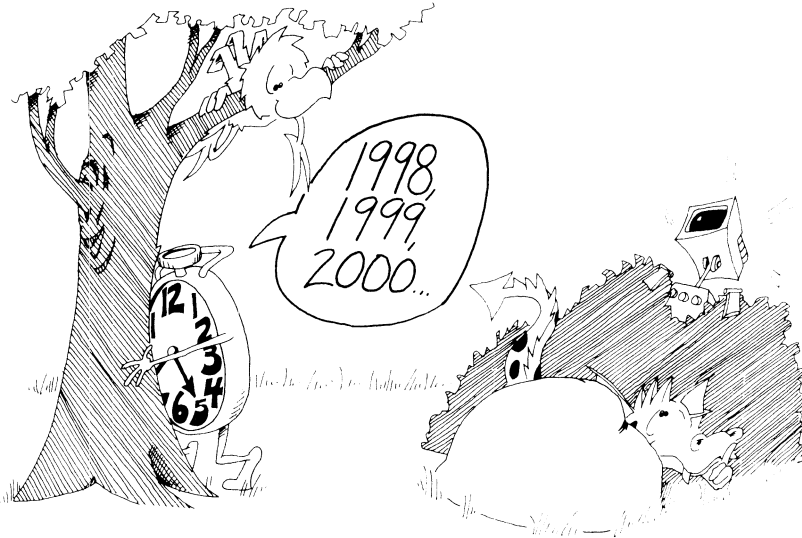
Delay Loops

Here is a way to slow down parts of a 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 1000:NEXT I
50 PRINT "COMING READY OR NOT!"
```

Line 40 is the delay loop. The computer counts from 1 to 1000 before going on to the next line. It's like counting when you are "it" in a game of hide and seek.



Try changing the number 1000 in line 40 to some other number.

Each 500 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*500
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 the seconds back to zero. Do the 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

In this lesson the IF statement is extended to numeric expressions. The logical relations used are

= > < <>

It's a good idea to get your 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 $>$ symbols points to the smallest of the two numbers.

This lesson also demonstrates the use of nested IFs. This is a powerful construction, but may be confusing. Go through the example to make sure your student understands the construction.

A homemade loop is demonstrated in the "Guessing Game," but is 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 THEN in an IF statement?
3. After this 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!"
```

These IF statements are 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."

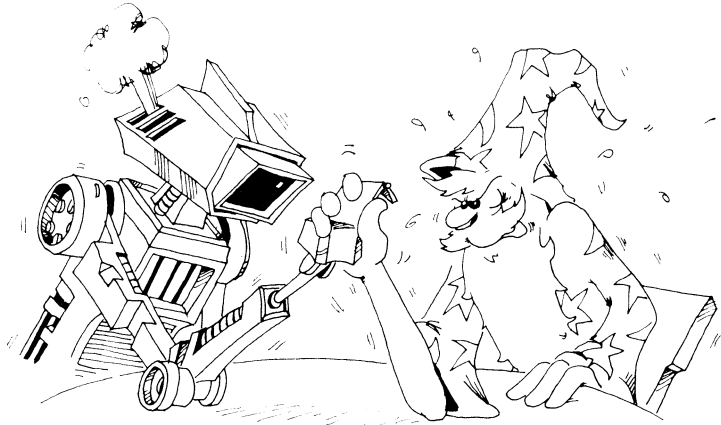
5 < 7 is pronounced "five is less than seven."

Practice

In the following 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 this line:

```
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

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 COLOR 0,1
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, 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 _____

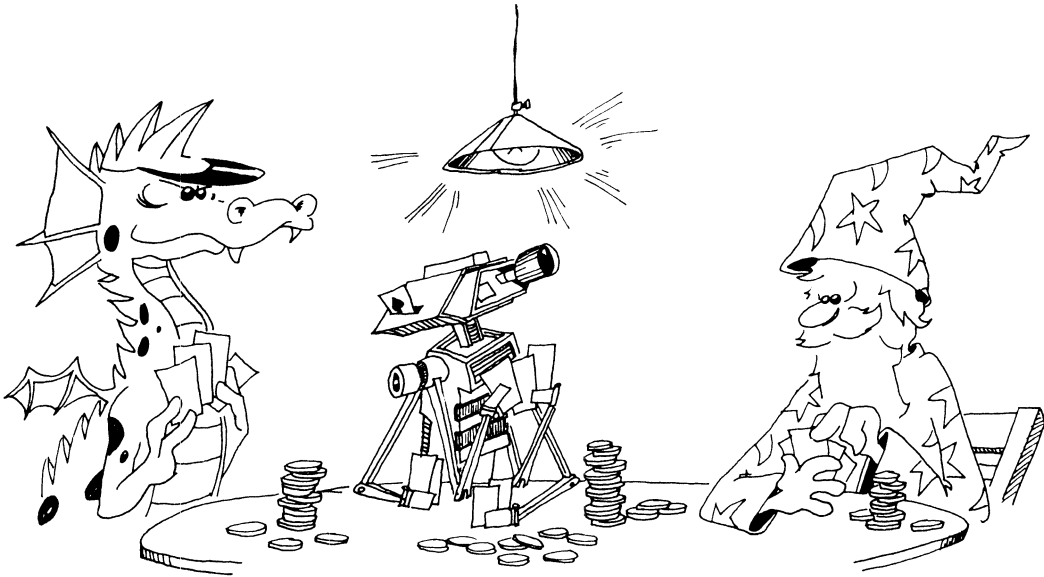
2. Here is another program. What will it print, and how many times?

```
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," "Seven is lucky," and so on.
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 (1-13) 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(). They are 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 through 6 for a die or 1 through 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 0 to 1, 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 considered 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 further clarified in this lesson.

Nesting 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)
```

when box G contains 2, 2.1, 2.95, 3.001, 67, 0, or 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 RND(8) 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

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

When you deal 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 these things. RND() stands for *random*. Like all functions, it will always be followed by a number in parentheses.

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. Run it a few times more. Does it always make the same numbers?

It doesn't matter what number you put in the parentheses just as long as it is greater than 0. In this example, 8 was chosen because it is near the () keys on the keyboard, making it easy to type (8).

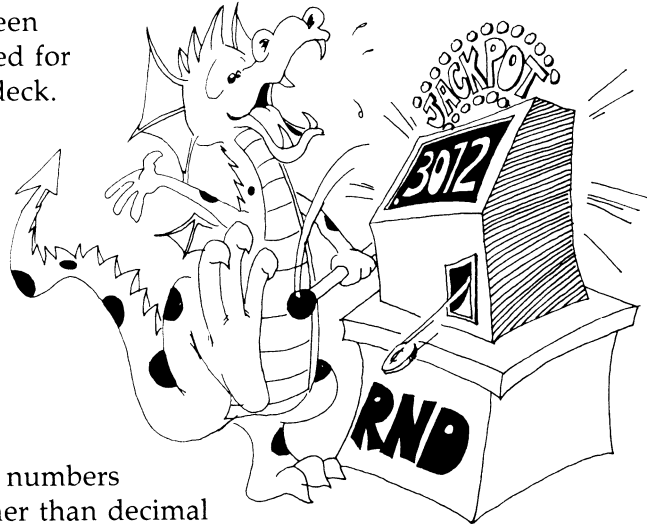
RND() gives numbers that are decimals larger than 0, but smaller than 1. To make numbers larger than 1, you just multiply.

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
```

Run it again a few times.

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



But usually we want whole numbers (integers) like 7 and 23 rather than decimal numbers like 7.03 and 23.62. You can get them by using the INT function.



The INT Function

INT() takes the number in its parentheses and throws away the decimal part, leaving a whole number. Add this line to the program above and run it again:

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

How It Works

Use this one-line program to check how INT() works:

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

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

Rounding Off Numbers

Perhaps you already 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	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 this:

```
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+.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 1000: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's how to save a lot of room.

Instead of this:

```
30 LET R=RND(8)
```

```
50 LET S=R*6
```

```
60 LET I=INT(S)
```

```
70 LET D=I+1
```

Use just one line:

```
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
```

This is how it works:

Expression	Makes numbers from:	
	(small to 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 playing 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 1571 disk drive. (See "Disk Usage," Appendix A.)

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

These commands are introduced:

DSAVE " <i>filename</i> "	save program
DLOAD " <i>filename</i> "	load program
CATALOG	load directory
SCRATCH " <i>filename</i> "	erase program from disk

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

You might show your student how to use the preprogrammed function keys as a shortcut to loading, saving, and calling the directory. Because the function keys can be redefined, we will not discuss them in this book. Refer to the System Guide which came with your computer for more information.

You can use this lesson 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 earlier 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 your program was saved onto disk?
 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 1571 or 1541 disk drive, you can store programs on a disk.

Entering a Program

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

If not, enter `NEW` and then these lines:

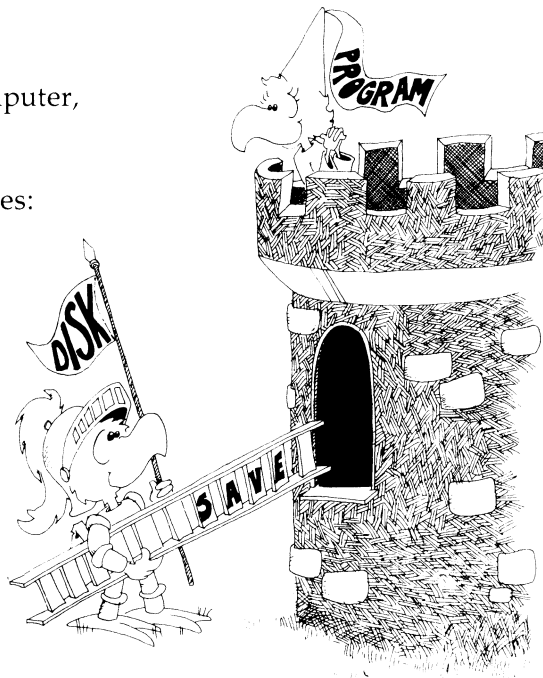
```
10 REM HOT DOG
20 PRINT "NO MUSTARD"
```

Saving a Program

Do you still have your disk in the drive? If not, put your disk in now. Be sure to twist the handle down until it clicks.

Enter:

```
DSAVE "HI "
```

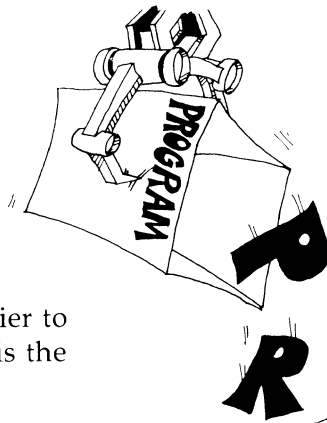


You will hear a whirring and see the green light on the disk drive come on. When the green light goes out and the whirring stops, your program is on the disk.

But if you see the green 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.

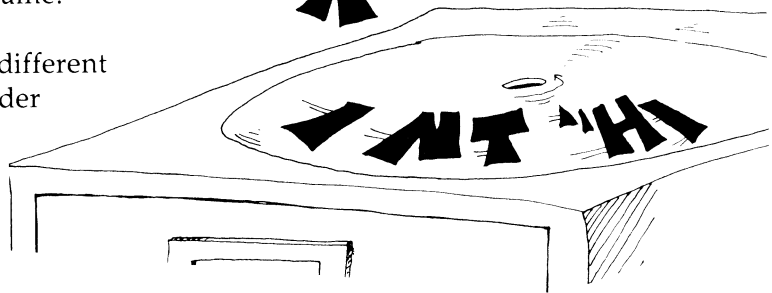
The Filename

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 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: CATALOG

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
```

PRG means the file folder contains a program.

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

Loading the Program

Enter:

```
NEW  
DLOAD "HI "
```

You hear the whirring and see the green 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 away a file?

Enter:

```
SCRATCH "HI "
```

SCRATCH means "erase." You want to scratch out the file.

The computer prints: ARE YOU SURE?

Enter Y (yes), and the disk drive whirs, and the file is gone from the disk. If you change your mind and don't want to erase the file, enter N (no).

Is It Gone?

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

Enter: CATALOG

to see if it is really gone.

Legal Filenames

Filenames:

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

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 &%\$#
128 DISK BACKUP

Wrong names:

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

If you use a name that has a comma in it, only the part of the name before the comma will be used. Try it.

Commands

The 1571 disk drive is smart. It has its own microprocessor chip and memory chips. Read the *1571 Disk Drive User's Guide* to find out more about it.

These four commands are used with files:

DSAVE "*filename*" to save the program
DLOAD "*filename*" to load the program
CATALOG to see the directory
SCRATCH "*filename*" to erase a file

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 CATALOG 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. Practice with the DSAVE, DLOAD, CATALOG, and SCRATCH commands until you are sure that you understand them.

Instructor Notes 15. Some Shortcuts

Now that we've reached RND() and learned how to save programs on disk, all the elements are in place for your student to write substantial programs.

Here are the shortcuts we'll cover in this lesson:

?	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

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 less cluttered program is the result. 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 question mark (?) 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.

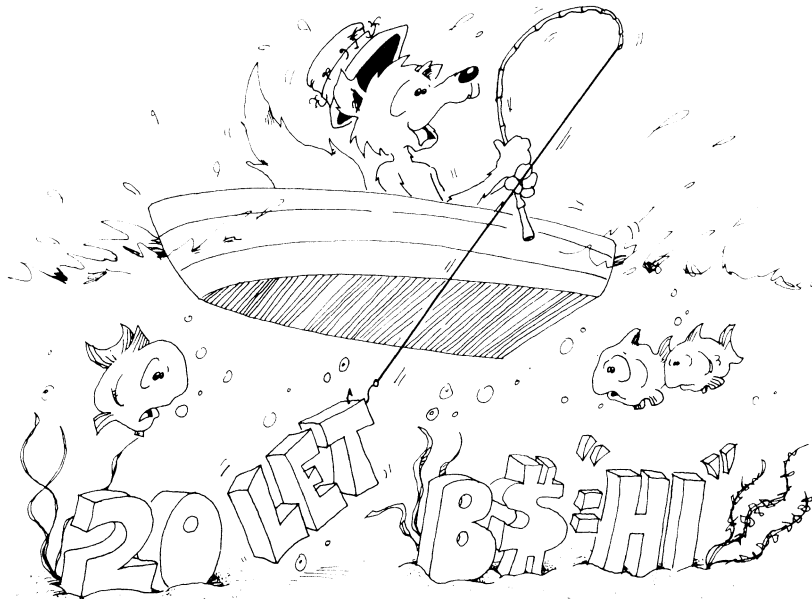
A LET Shortcut

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:

```
1Ø PRINT "ENTER YOUR NAME "  
2Ø INPUT N$
```

You can do this:

```
1Ø 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:

```
2Ø INPUT "LOCATION";X,Y  
LOCATION?
```

You see: LOCATION? on the screen.

You enter two numbers with a comma between them:

```
LOCATION? 5,6
```

Here's another example:

```
3Ø INPUT "MONTH, DAY, YEAR";M$,D,Y
```

After the ?, type: APRIL,29,1986

Error Message in INPUT

If you don't enter enough answers, the computer asks ??. Then you should enter the rest.

Example:

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

```
? MAY, 1  
?? 1986
```

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 this line:

```
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's okay to answer with a number for a string, because the computer says 1986 is a string.)

A LIST Shortcut

There are five ways to use the LIST command:

LIST	Lists the 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:PRINT 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 colon shortcut:

1. To make the program clearer.

Put similar statements on the same line. For 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. For example,

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

The Colon After an IF Statement

You can make neater IF statements by 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: Don't put something on the end of an IF line that doesn't belong.

This 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 whether $A = B$ is true or not. But $Q = R$ in line 37 is done only if $A = B$ is true.

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
```

Incorrect:

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

The computer ignores everything else on a line after reading REM.

Correct:

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

Incorrect:

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

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:

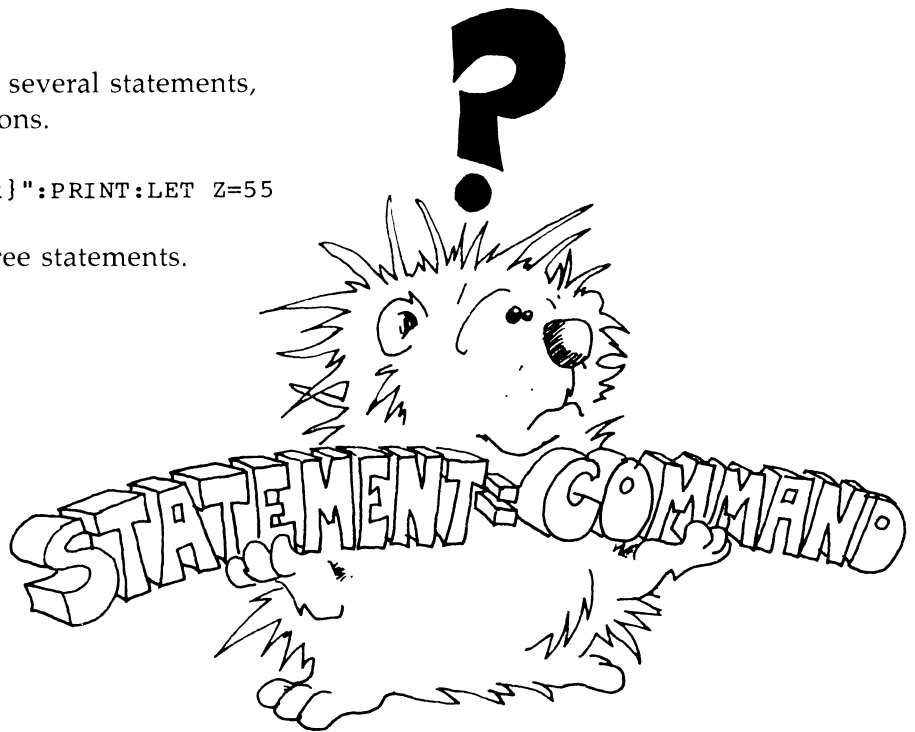
PRINT, NEW, RUN, LIST, REM, COLOR, INPUT, LET, GOTO, IF, DSAVE, DLOAD, CATALOG, and SCRATCH

Statements are instructions to the computer which are most often used in numbered lines. Commands are usually used alone. NEW, DSAVE, DLOAD, 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.



The SLEEP Statement

In place of

```
100 FOR T=1 TO 500:NEXT T
```

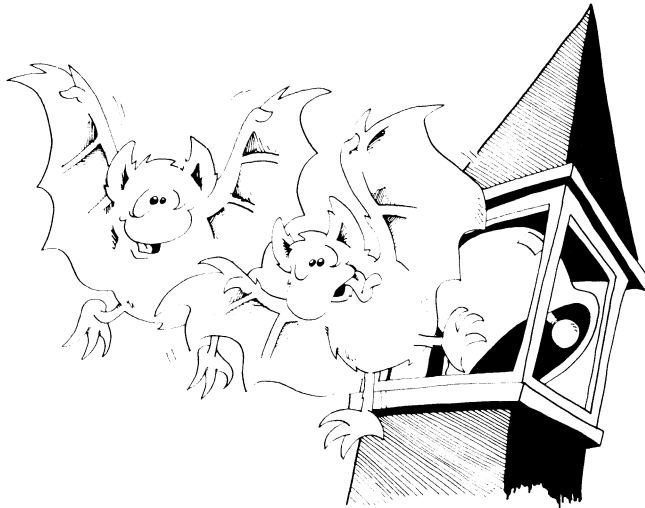
use

```
100 SLEEP 1
```

The SLEEP N statement pauses the computer for N whole seconds. Fractions of a second are ignored, so SLEEP 1.9 pauses only one second.

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 CRSR 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 the cursor will drop down to the beginning of the next line and will 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 "123456789012345678901234567890123456789"
30 GOTO 30
```

But if you add a 0 to the end of the string (making 40 characters), 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 to draw a grid representing the screen, then lightly sketch the characters one by one (and erase 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 that was printed in question 3.

Lesson 16. Moving Pictures

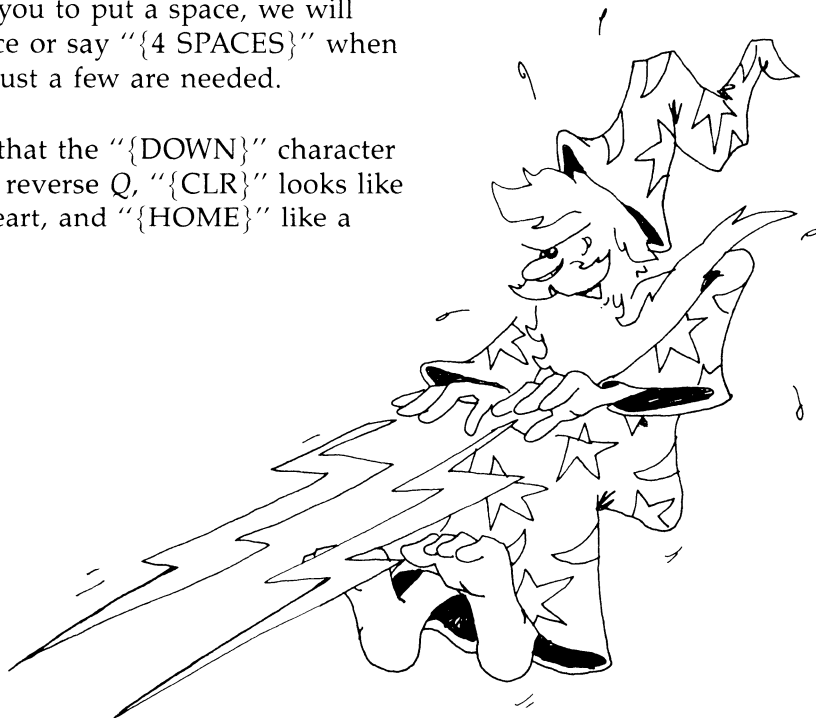
To move the cursor, put CRSR characters in a PRINT statement.

```
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 inside 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 CLR/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 a reverse Q, "{CLR}" looks like a reverse heart, and "{HOME}" like a reverse 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 enter this program 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 there is a semicolon at the end of each PRINT line, and that lines 30 and 50 have the birds, three left-CRSR characters, and no spaces.

How Does It Work?

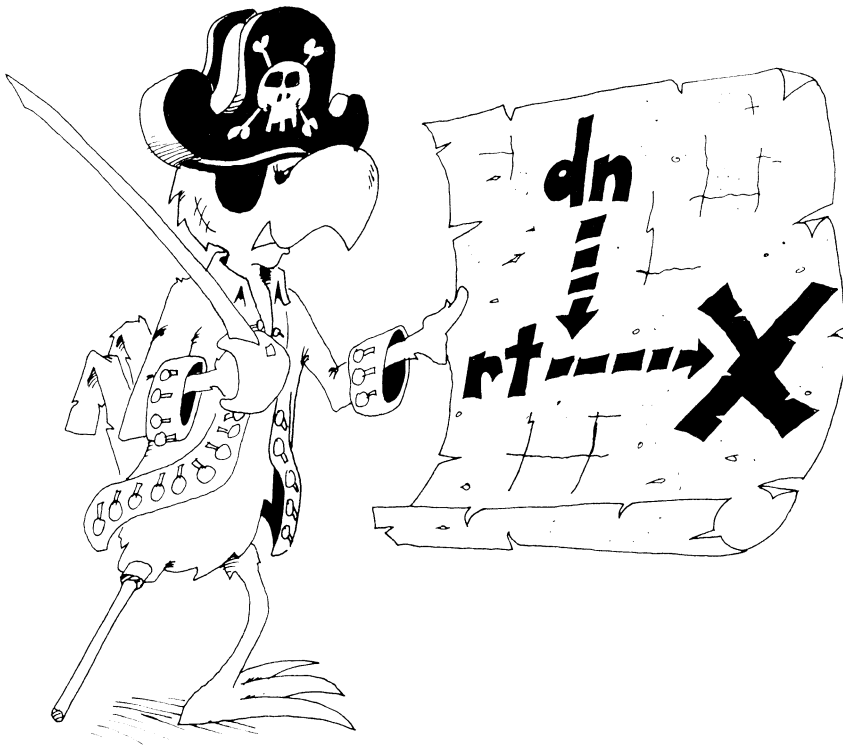
Look at lines 30 and 50 character by character.

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 the bird program, 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 " {2 LEFT}{UP}";:REM ERASE BIRD, GO UP
50 PRINT "JQK{3 LEFT}";:REM PRINT UP WINGS
60 FOR T=1 TO 200:NEXT T
65 PRINT " {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 that the ball moves from right to left.
2. Write a program that makes 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 like any other numeric 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 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 ten down to zero 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
```

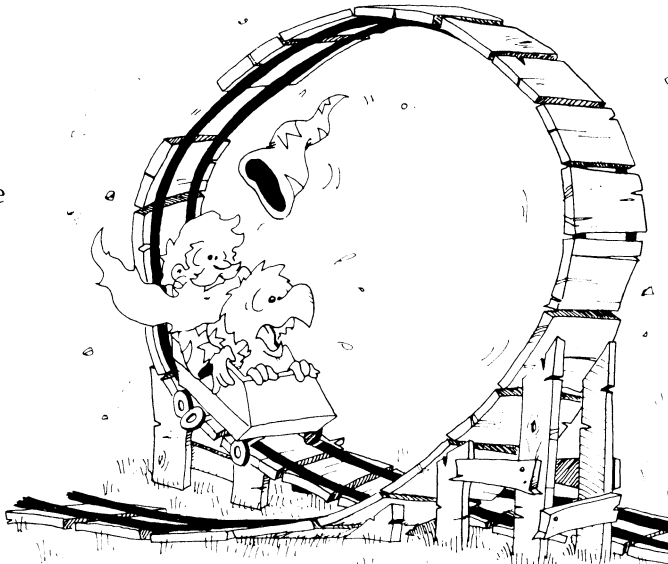
The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

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

Mark Your Listings

Draw arrows to show where the loops are:

```
10 REM ON PAPER
20 PRINT "{CLR}"
30 FOR I=1 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

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 COLOR 0,13
20 PRINT"{CLR}{YEL}"
30 PRINT" T MINUS 12 SECONDS AND COUNTING"
35 FOR I=1 TO 500:NEXT I
40 FOR T=11 TO 0 STEP -1
50 PRINT "{CLR}{DOWN}";T
60 FOR I=1 TO 500:NEXT I:REM TIMING LOOP
70 NEXT T
80 PRINT"{CLR}{DOWN}{RED} ALL ENGINES RUNNING, LIF
T OFF."
81 FOR I=1 TO 500:NEXT I
82 PRINT"{CLR}{DOWN}{BLU} WE HAVE A LIFT OFF."
83 FOR I=1 TO 500:NEXT I
84 PRINT"{CLR}{DOWN}{CYN} 32 MINUTES PAST THE HOUR
."
85 FOR I=1 TO 500:NEXT I
86 PRINT"{CLR}{DOWN}{CYN} LIFT OFF ON APOLLO 11."
90 COLOR 0,1
```

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

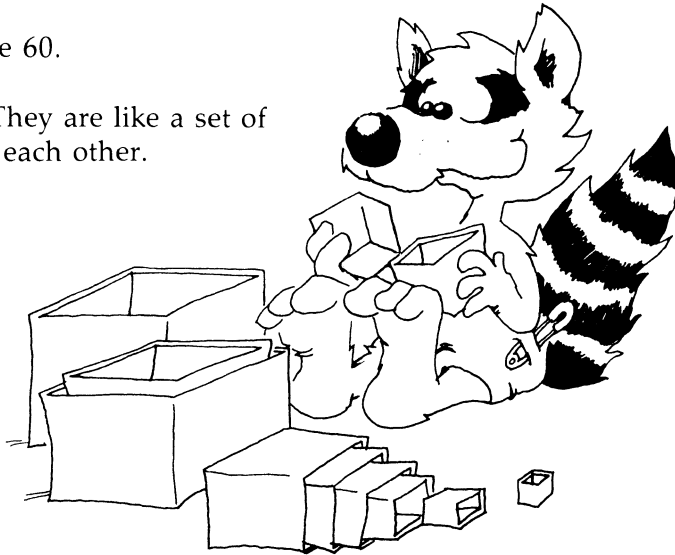
Nested Loops

In this program, we have one loop inside another loop.

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 500:NEXT I:REM TIMING LOOP
```

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

```
40 FOR T=11 TO 0 STEP-1
```

.....

```
70 NEXT T
```

T is the loop variable.

Badly Nested Loops

The inside loop must be all the way inside:

Correct:

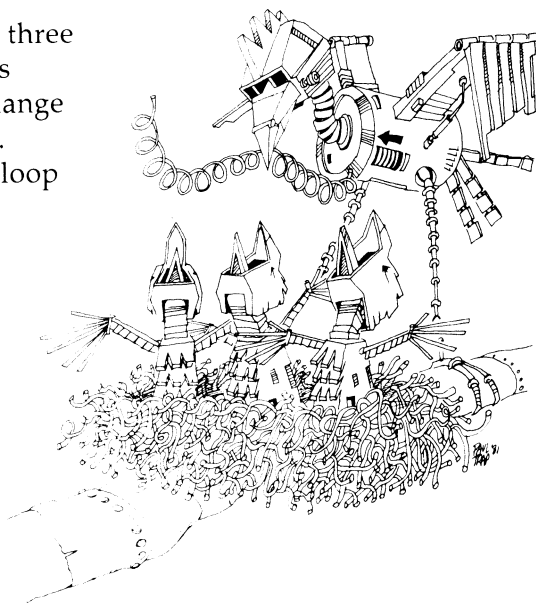
```
25 FOR X=3 TO 7
30 FOR Y=3 TO 7
40 PRINT X*Y
50 NEXT Y
60 NEXT X
```

Incorrect:

```
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 CRSR-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 DATA statements. READ gets data from DATA statements, and RESTORE puts the pointer back to the beginning of the first DATA statement.

Storing data in DATA statements has a few confusing elements when first confronted. You can never change any of the data in a statement unless you rewrite 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 what comes before it. (This procedure is not discussed here. You may mention it 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 much data, because it clearly separates the actual data from the processing of the data. This helps when debugging programs.

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 numeric data and string data in the same DATA statement?

-
-
5. What happens if you try to READ a string into a numeric variable?
 6. Can you change the items in a DATA statement while the program runs?

Lesson 18. DATA, READ, RESTORE

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 items 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,
FRIDAY, 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), and so on.

Strange Rules

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

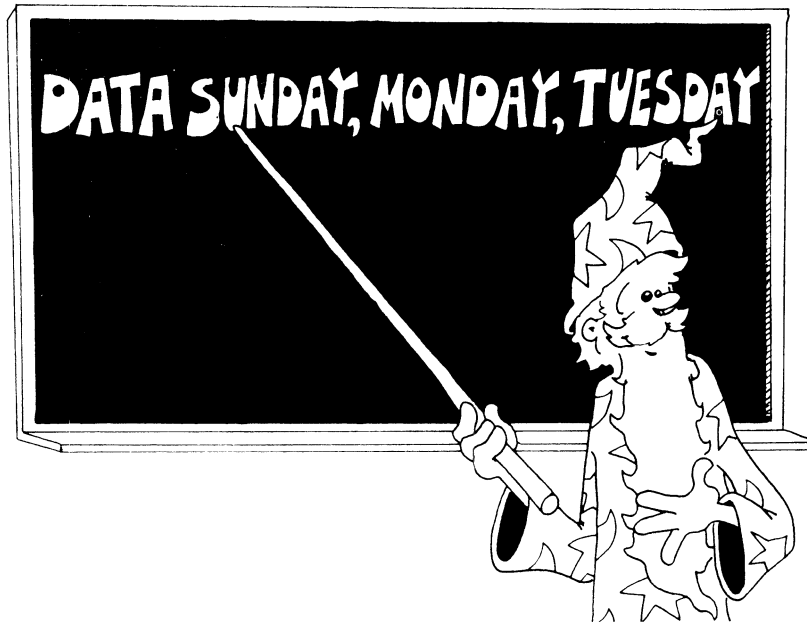
Do this: Change line number 30 in the above program to line number 90. Run the program. It works just the same.

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.

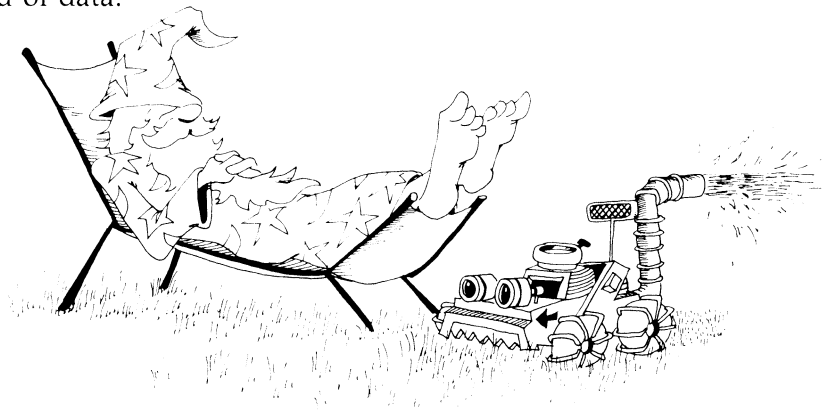
But you must be careful in your READ statement to have the correct kind of variable to match the kind of data.

Correct:

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

Incorrect:

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



You can't put FUZZ in a number box.

Assignment 18

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

Instructor Notes 19. Sound Effects

The Sound Interface Device (SID) chip in the Commodore 128 produces sound and music. This lesson introduces the SOUND statement for making sound effects. Musical applications are covered in a later lesson.

Sound is sent to the 1702 or 1902 color video monitor along the same cables as the video signal. If you use a TV as a monitor, the sound is sent as part of the signal.

The chip can produce up to three independent voices or sounds at the same time. The pitch of each voice is set by a number between 0 and 65535. 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, and so forth. One of the choices is white noise, which is very useful for producing sound effects (drums, gunfire, thunder, wind, waves, and so on).

The length of the note is specified in “jiffies,” between 0 and 65535. Each jiffy is 1/60 second.

Questions

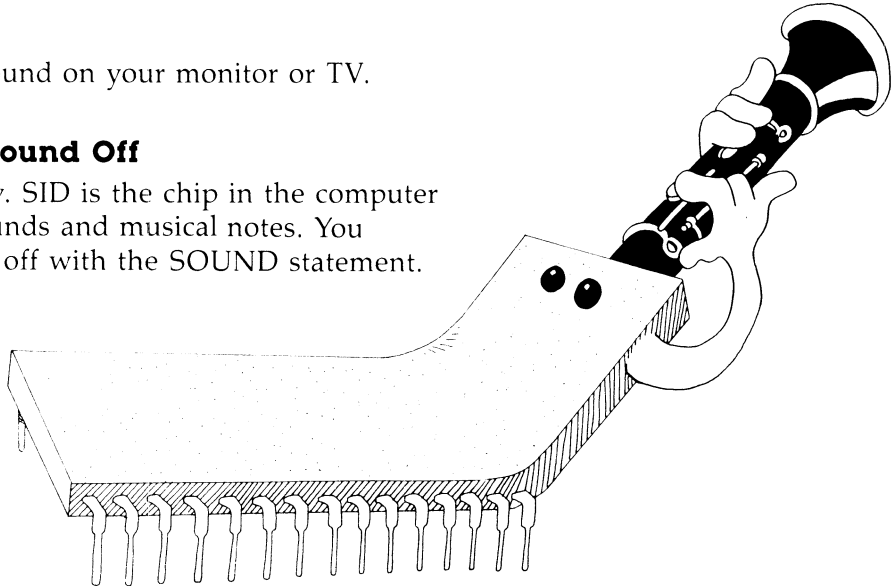
1. What three things must you put in the SOUND statement to hear a sound?
2. How many voices can sound at once?
3. What duration number will give you one second of sound?
4. What waveform number gives a swishing noise?

Lesson 19. Sound Effects

Turn up the sound on your monitor or TV.

Tell SID to Sound Off

No, not Sidney. SID is the chip in the computer that makes sounds and musical notes. You make it sound off with the SOUND statement.



Enter this command:

```
SOUND 1, 2000, 60
```

SID should make a musical tone lasting one second (60 jiffies) with voice 1. Now enter:

```
SOUND 1, 2000, 60 : SOUND 2, 3000, 120
```

SID will sing with two voices, the second one higher and lasting two seconds.

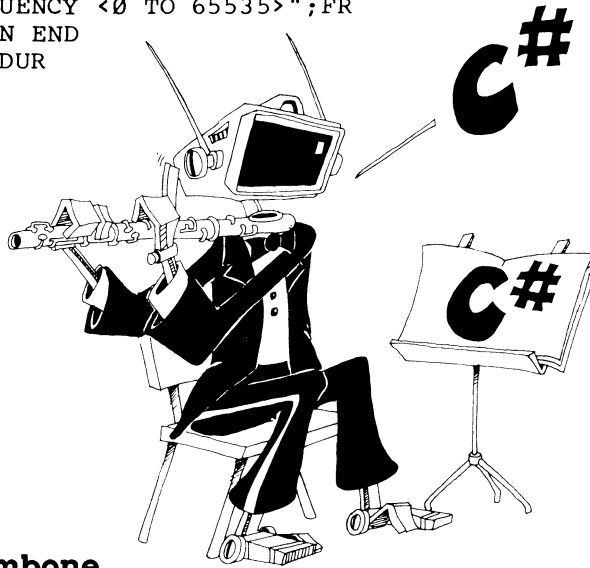
In the SOUND statement, the first number tells SID to use voice 1, 2, or 3. (The voices all sound alike). The second number tells which pitch. Numbers below about 200 sound like a lot of clicks. The pitch number can be as large as 65535, which gives a very high note.

Run this "Peep" program and listen to the different sounds you get from different frequency numbers.


```

10 REM ----- PEEP
15 REM ----- SOUND VC, FR, DUR
21 VC =1      : REM ---- VOICE 1
23 REM ----- FREQUENCY
25 DUR=120 : REM ---- DURATION: 2 SECONDS = 120/60
29 PRINT" ENTER 0 TO END PROGRAM" : PRINT
30 INPUT" FREQUENCY <0 TO 65535>";FR
40 IF FR=0 THEN END
50 SOUND 1,FR,DUR
60 GOTO 30

```



The Slide Trombone

The SOUND statement can take three more numbers beyond the voice, frequency, and duration set of three. They are:

DIR	direction	go up, down, or waver?
MIN	minimum	lowest frequency
SV	step value	how fast the pitch changes

Try this:

```
DIR=0:MIN=5000:SV=50:SOUND 1,9000,300,DIR,MIN,SV
```

Run your cursor back into the line and change DIR=0 to DIR=1. Now try again. Change it to 2, and try again.

In each case the sound swings between the high pitch given by 9000 in the frequency spot (second number) and the lower pitch of 5000 given in the MIN (minimum frequency) spot. How it swings is given by the DIR number.

DIR=0 means it starts at 5000, swings up to 9000, and if the duration has not run out, jumps back to 5000 and swings up again.

DIR=1 means it starts at 9000, swings down to 5000, and jumps back up and keeps repeating.

DIR=2 means it starts at 9000, swings down to 5000, and swings back to 9000. The swinging continues until the duration time runs out.

Now look again at the SOUND statement above. The last number (SV) is the step value. Change it to 10, then 200, then 1000, and hear that the sweeps occur faster when the number is larger. You should always choose a smaller SV number than the MIN number.

Now put it all together in the "Siren" program.

```
10 REM ----- SID IS A SIREN
11 :
15 REM ----- SOUND VC, FR, DUR, DIR, MIN,
    SV, WF, PW
17 VC=1      :REM --- VOICE 1
18 FR=45000  :REM --- FREQUENCY: 0 TO 65535
19 DUR=250   :REM --- DURATION 250 JIFFIES, WHERE
    {SPACE}60 JIFFIES = 1 SECOND
20 DIR=2     :REM --- 0=GO UP, 1=GO DOWN, 2=WAVER
    {SPACE}UP AND DOWN
21 MIN=20000 :REM --- MINIMUM FREQUENCY: 0 TO 6553
    5
24 SV = 3000 :REM --- SWEEP SIZE: 0 TO 65535
25 WF = 0    :REM --- WAVEFORM: 0=TRIANGLE, 1=SAWT
    OOTH, 2=PULSE, 3=SWISH NOISE
50 SOUND VC,FR,DUR,DIR,MIN,SV,WF
```

The Triangle, Saw, and Jug Band

There is one last number you can put on the end of the SOUND statement. It is called WF for "waveform." Try this:

```
WF=0:SOUND 1,5000,120,0,5000,100,WF
```

Repeat with WF = 1, 2, and 3.

Here are the four waveforms and their values:

WF = 0	triangle wave	sounds smooth
WF = 1	sawtooth wave	sounds "buzzy"
WF = 2	pulse wave	sounds less buzzy
WF = 3	white noise	sounds swishy, like wind

Change WF to 1, 2, and then 3 in line 25 of Siren, and listen to the difference in the sound.

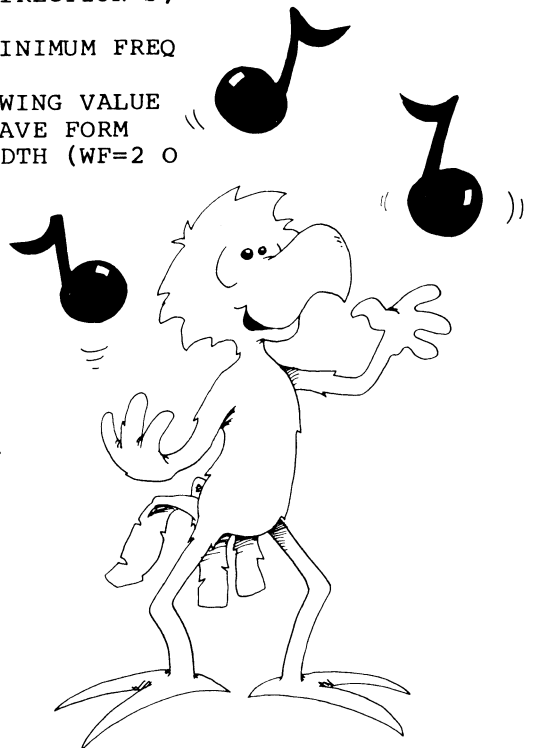
Random Noises

Try this program. Save it to disk before you run it.

```
10 REM ----- RANDOM NOISES
15 HI =65535 : LO=32767
20 FOR VC=1 TO 3 :REM VOICE
22 FR =INT(RND(8)*HI)
24 DUR=INT(RND(8)*180)+60 :REM RANDOM DURATION
26 DIR=INT(RND(8)*3) :REM RANDOM DIRECTION 0,
    1, OR 2
28 MIN=INT(RND(8)*FR) :REM RANDOM MINIMUM FREQ
    UENCY
30 SV =INT(RND(8)*LO) :REM RANDOM SWING VALUE
32 WF =INT(RND(8)*4) :REM RANDOM WAVE FORM
34 PW =INT(RND(8)*4096) :REM PULSE WIDTH (WF=2 O
    NLY)
50 PRINT VC;FR;DUR;DIR;MIN;SV;WF;PW
55 SOUND VC,FR,DUR,DIR,MIN,SV,WF,PW
60 NEXT VC : PRINT : PRINT : GOTO 10
```

Assignment 19

1. Try to make these sound effects:
laser gun, wind (use the white noise waveform), motor boat, telephone ring, explosion.
2. Write a program that shows 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 anywhere in the name—start, middle, or end. Reserved words are the BASIC statements, commands, and functions; they 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 numeric variable name does not.

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 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

Take a look at these names:

Some Right Names

A77
LEFTSIDE
X3AB\$
APE
NAME\$
COWBOY
X3

Some Wrong Names

2X
X#
\$NAME
LIST
COLOR (reserved word)
GIFT\$ (it has IF in it)
TOM (it has TO in it)

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

```
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
X1	is the same as	X10
NAME1\$	is the same as	NAME2\$

Try this program:

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

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

Executing and Running

To *execute* a program means the same as to *run* a program.

It's like a soldier executing the command “Left face!”

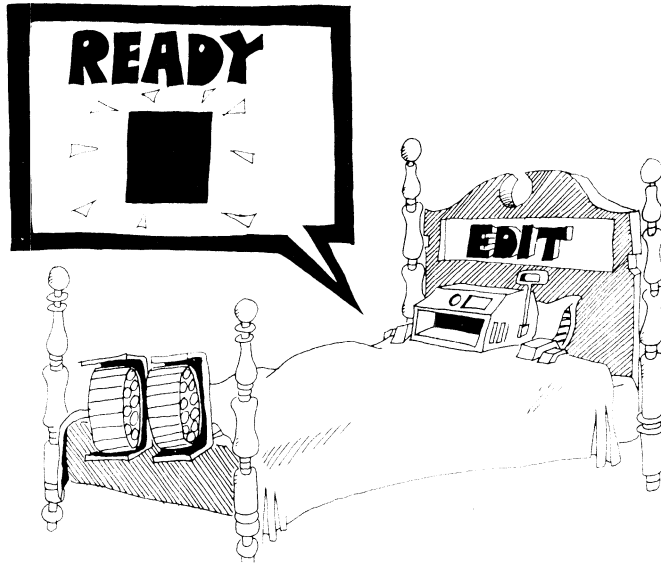
Edit Mode and RUN Mode

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

```
READY.
```

and a flashing cursor.

In edit mode, the computer is almost asleep. It is 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 Jiffy Clock

The computer has a clock that starts when you turn on the computer. It runs all the time.

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

Run this:

```
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.

Now run this:

```
10 REM JIFFY CLOCK
11 PRINT "{CLR}"
12 T=INT(TI/60)
20 IF T<>TT THEN PRINT "{HOME}";T:TT=T
30 GOTO 12
```

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 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 SUPERCALIFRAGILISTICEXPALIDOCIOUS 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 128. It starts with a procedure for systematically adjusting the color controls on the TV or monitor.

The next lesson introduces the POKE statement, which puts graphics characters on the screen in color.

COLOR controls screen border and background colors.

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 command changes the screen background color?
5. What command changes the screen border color?
6. What happens if you press the Commodore key and a color key?

Lesson 21. Color Graphics

Here's how to adjust the colors on your color TV or monitor. Put all the colors on the screen like this:

First press CONTROL and Rvs On. This will cause a space to be a colored spot.

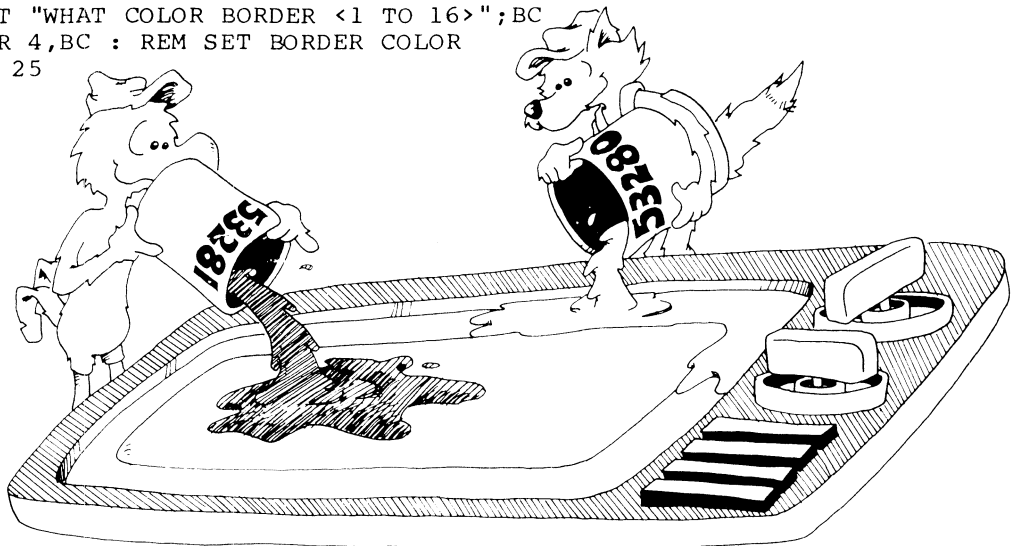
Now Press CONTROL and the Red key. Then hold down the space bar to make a red stripe on the screen.

Repeat these steps with the rest of the color keys, including the colors produced by pressing the Commodore key and a color key. When you're finished, press CONTROL and Rvs Off.

Now you have 16 colored stripes on the screen. Adjust the color controls on the TV or monitor until each color looks correct.

Run this program:

```
10 REM COMBINATIONS
25 PRINT "{CLR}{3 DOWN}"
30 INPUT "WHAT COLOR SCREEN <1 TO 16>";SC
37 COLOR 0,SC : REM SET SCREEN COLOR
38 PRINT "{CLR}{3 DOWN}"
40 INPUT "WHAT COLOR BORDER <1 TO 16>";BC
50 COLOR 4,BC : REM SET BORDER COLOR
90 GOTO 25
```

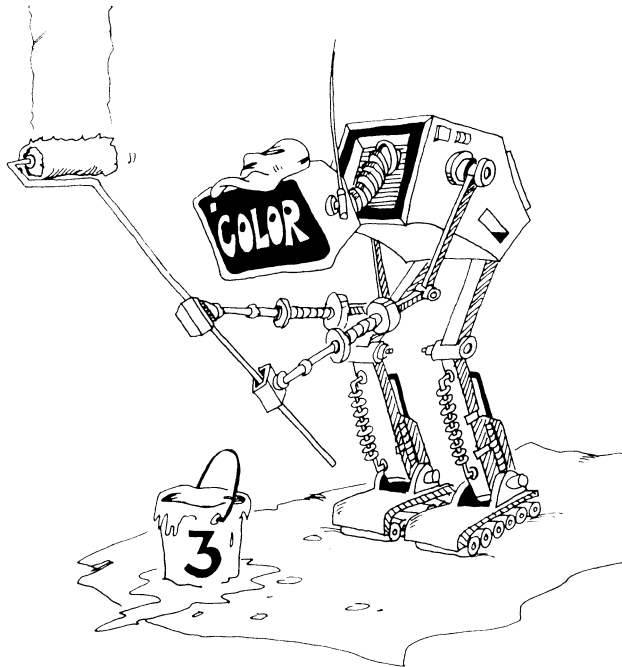


Color Numbers

The screen can have any of these 16 colors.

1 Black	9 Orange
2 White	10 Brown
3 Red	11 Light Red
4 Cyan (Blue-Green)	12 Gray 1
5 Purple	13 Gray 2
6 Green	14 Light Green
7 Blue	15 Light Blue
8 Yellow	16 Gray 3

You can change the printing color, even while the "Combinations" program is running, by pressing the CONTROL key and a color key. Try it.



Character Colors

You can change the color of the letters that are printed while the program is running by putting a number from 1 through 16 in the COLOR 5,N command.

Add these lines to the program and run it:

```
60 INPUT "WHAT COLOR LETTERS<1 TO 16>";CL
64 COLOR 5,CL
```

Careful: If you choose the *same* color for the screen and letters, you won't be able to see the printing. Don't give up. Just press CONTROL and another color key. The printing will appear again in another color.

Jumping Name

Let's look at this program.

```
10 REM JUMPING NAME
15 PRINT "{CLR}{3 DOWN}": COLOR 0,1 :REM CLEAR SCREEN
   EEN TO BLACK
30 INPUT "NAME";N$
32 INPUT "HOW MANY LETTERS IN IT";N
39 PRINT "{CLR}" :REM CLEAR SCREEN
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)*16)+1 :REM RANDOM COLOR
58 IF NC=1 THEN 57 :REM NOT BLACK
60 COLOR 5,NC: PRINT N$ :REM PRINT COLORED NAME
80 FOR T=1 TO 500:NEXT T: 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 don't want black letters on a black screen.

Line 60 prints your name in the new color.

Line 80 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 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 (lesson 16) that ask the user what color bird to print. Then put the color into the COLOR 5,N statement.
4. Write a program that draws your country's flag.

Instructor Notes 22. POKEing Graphics

This lesson explains POKEing characters and colors to the screen.

A 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 using POKE to make graphics 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.

You must exercise great care in POKEing because a wrong address may put junk in some vital part of memory and make the system crash. 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. 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?
4. What danger is there in using POKE?

Lesson 22. POKEing Graphics

Make colored stripes and adjust the TV or monitor color knobs (see the last lesson if you don't remember how to do this).

The Screen Is a Map of Memory Boxes

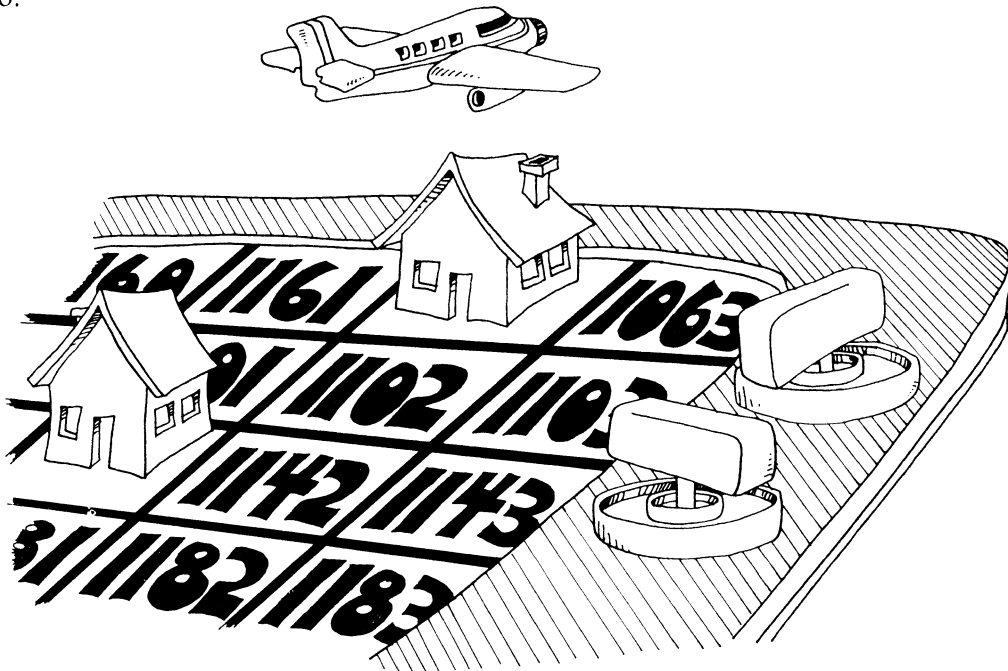
Imagine a little town with 25 streets. Each street has 40 houses on it.

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 line:

```
POKE 1024,83:POKE 55296,2
```

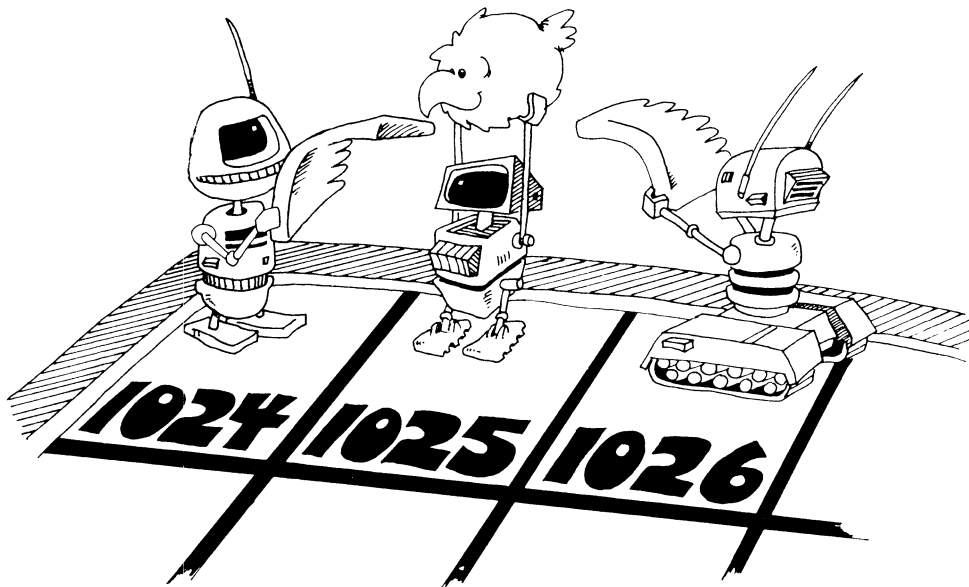
You will see a red heart in the *home* position on the screen (upper-left corner). What happened? We POKEd the heart (character number 83) into memory box number 1024 that shows in the home position on the screen.

POKE means that 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 this program. Save it before you run it.

```
10 REM EVERY OTHER HOUSE
12 PRINT "{CLR}{WHT}"           :REM CLEAR SCREEN
15 COLOR 0,1                   :REM BLACK SCREEN
20 D$=""                        "
22 FOR I=1 TO 25:PRINT D$:NEXT I
25 FOR I=1024 TO 2023 STEP 2:REM WALK ALONG SCREEN
30 CH=INT(RND(8)*64)+64        :REM RANDOM CHARACTERS
60 POKE I,CH                   :REM POKE CHARACTER
80 NEXT I                      :REM END OF LOOP
90 PRINT "{HOME}"             :REM PUT CURSOR HOME
```

This time all the characters were white because the {WHT} character was printed in line 12 and spaces in line 20.

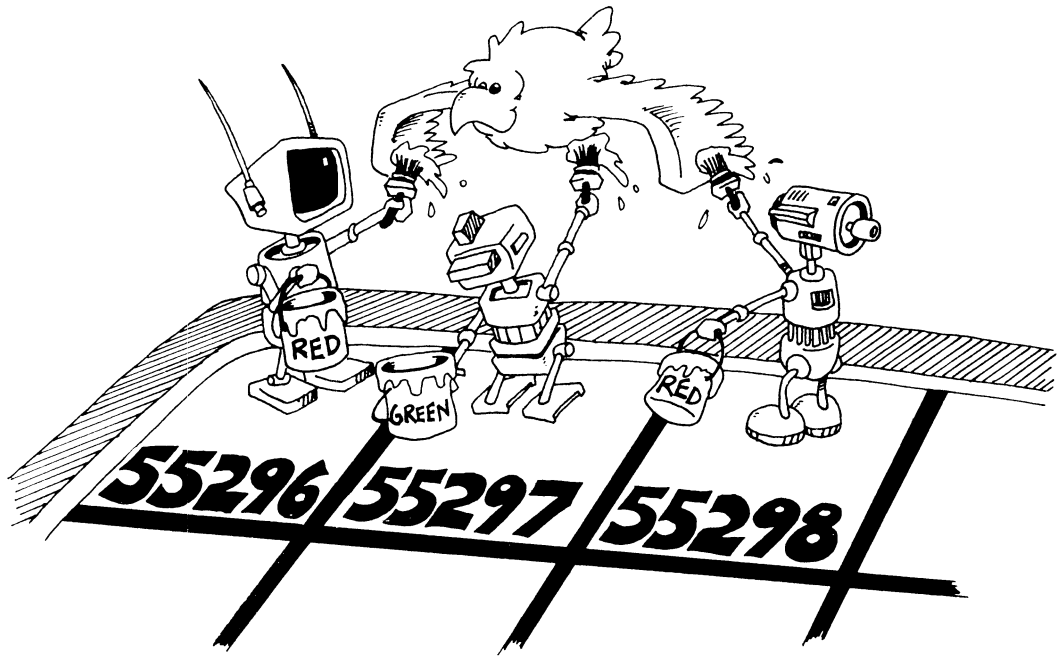
If you want another color, you have to POKE a number into the correct color map box.

Let's change the program. Enter this line:

```
65 POKE I-1024+55296,RND(8)*16+1:REM COLOR BOXES
```

To get just letters and punctuation on the screen, change this line:

```
30 CH=INT(RND(8)*64)           :REM RANDOM CHARACTERS
```



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 formula to get the address of each house and its color box.

Screen address = $1024 + 40 * \text{street number} + \text{house number}$.

Color address = $55296 + 40 * \text{street number} + \text{house number}$.

(Look at line 50 in the "Town Map" program below.)

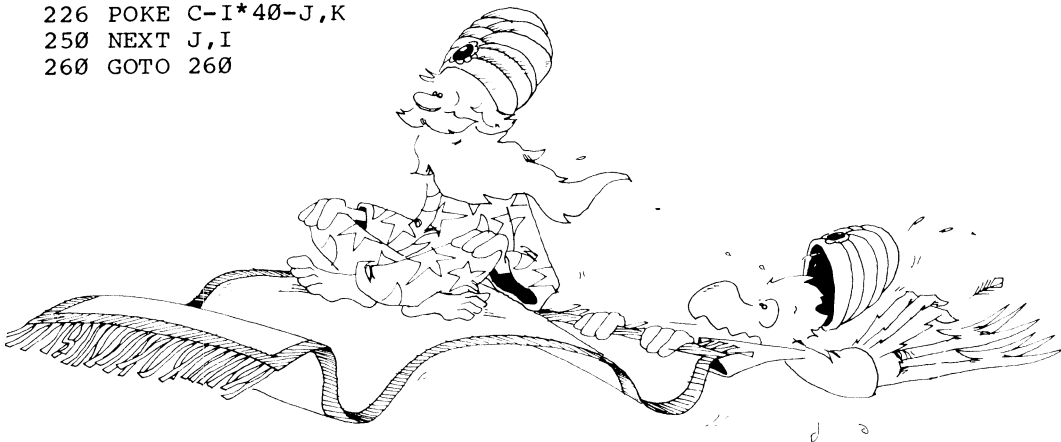
Enter this program. Then save it and run it. (Omit the REM statements when you type it in.)

```
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 500 :NEXT T
80 GOTO 30
```

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 listed 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=1024
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
260 GOTO 260
```



Instructor Notes 23. Secret Writing and the GET Statement

This lesson concerns the GET and GETKEY statements.

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 numeric and the keystroke was not, a ?TYPE MISMATCH ERROR is printed.)

If the line buffer is empty, it returns zero to a numeric variable or "" (the empty string) to a string variable.

In situations where you want time to think before pressing a key, use the GETKEY statement.

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. How is GETKEY different from GET?

Lesson 23. Secret Writing and the GET 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 to press the RETURN key.

```
10 REM ----- GET -----
20 PRINT "{CLR}{2 DOWN}"
30 PRINT "{DOWN}{YEL} PRESS DIFFERENT KEYS"
35 SLEEP 3
40 GET K$:PRINT K$
60 GOTO 40
```

The GET Statement and Secret Writing

The GET statement is different from INPUT. It gets a single character from the keyboard. Here's an 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 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 time to think before you press a key. The computer does not wait for you to think.

Make the Computer Wait

You can ask the computer to keep looking at the keyboard until you press a key. Example:

```
40 GETKEY K$
```

This is a good way to use GET in guessing games. You can enter the word or number to be guessed, and the other player won't be able to see it.

Run this program:

```
10 REM ----- GETKEY -----
20 PRINT "{CLR}{DOWN}"
30 PRINT "{DOWN}{YEL} PRESS ANY KEY"
   {SPACE}:GETKEY K$
45 PRINT "{DOWN}{CYN} WAITING"
   {SPACE}:SLEEP 1
50 PRINT "{DOWN}{PUR} THE KEY YOU PRESSED WAS ";K$
   : SLEEP 1
60 GOTO 20
```

Making Words of Letters

The GET statement gets one letter at a time. To make words, you need to glue the strings.

```
10 REM ## BACKWARDS ##
15 COLOR 0,1 : PRINT "{CLR}{3 DOWN}"           :REM
   {SPACE}BLACK SCREEN, CLEAR SCREEN
30 PRINT " TYPE A WORD {DOWN}"
31 PRINT " END IT WITH A SPACE"
35 W$=""                                         :REM WORD STRI
   NG IS EMPTY
40 GETKEY L$
50 : IF L$=" " THEN 80                           :REM SPACE IS
   {SPACE}TEST FOR END
60 W$=L$+W$                                     :REM ADD LETTE
   R TO FRONT OF WORD
65 GOTO 40                                       :REM GO GET AN
   OTHER LETTER
80 :                                             REM WORD IS F
   INISHED
82 PRINT "{CLR}{4 RIGHT}{PUR} HERE IT IS BACKWARDS"
85 PRINT "{DOWN}{GRN} ";W$;"{YEL}"
```

How does the computer know when the word is all typed in? It sees a space at the end of the word. Line 50 tests for the space and ends the word when it finds the space.

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.

Enter and run:

```
10 GETKEY N
20 PRINT N
30 GOTO 10
```

Press some number keys. Now press a letter key. When you try to GET or GETKEY a letter character in a numeric variable, you see:

```
?TYPE MISMATCH ERROR
```

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 of a program line to make a menu:

```
PRINT "WHICH COLOR? <R=RED,B=BLUE,G=GREEN> "
```

Make the program change 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 GETKEY so that 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 gets the line number from the stack and returns to the statement following the GOSUB.

Subroutines are useful not only in long programs, but also 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 *top-down programming*; it uses various techniques to discipline the programmer.

Call your 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 programs 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

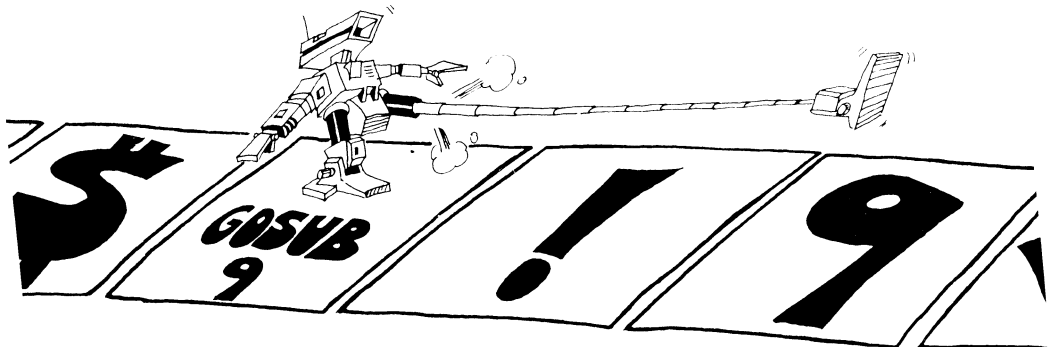
Run this program and then save it:

```
100 REM TAKE A TRIP
101 :
105 COLOR 0,1
110 PRINT "{CLR}{DOWN} HOP TO THE SUBROUTINE
      {DOWN}"
120 SLEEP 1 : GOSUB 200
130 PRINT "{GRN} BACK FROM THE SUBROUTINE {DOWN}"
133 SLEEP 1
135 PRINT "{BLU} HOP AGAIN {DOWN}"
140 SLEEP 1 : GOSUB 200
150 PRINT "{CYN} HOME FOR GOOD {DOWN}{YEL}"
190 END
199 :
200 REM SUBROUTINE
201 :
210 PRINT "{RED} GOT HERE OK {DOWN}"
212 SOUND 1,2000,60
215 SLEEP 1
220 PRINT "{YEL} PACK YOUR BAGS, BACK WE GO
      {DOWN}"
230 SLEEP 1
290 RETURN
```

(Do you remember the SLEEP statement? See lesson 15 if you don't remember.)

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.

Lines 120 and 140 call the subroutine. This means that the computer goes and performs the statements in the subroutine. Then it 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.

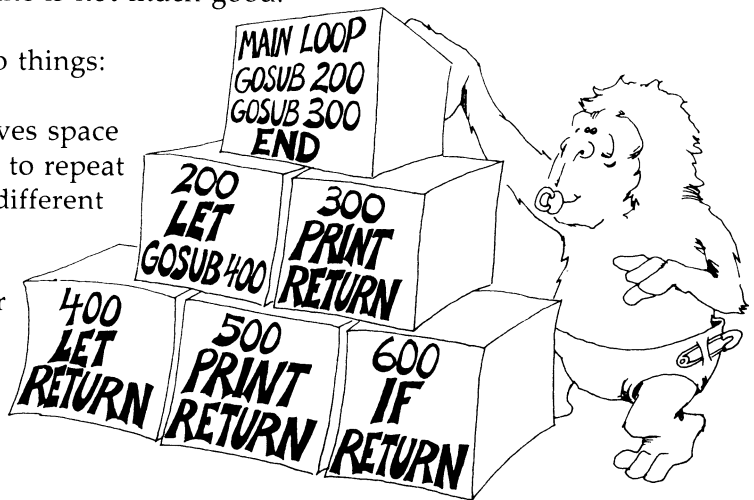
The RETURN statement tells the computer to go back to the statement after the GOSUB.

What Good Is a Subroutine?

In a short program, a subroutine is not much good.

In a long program, it does two things:

1. It saves you work and it saves space in memory. You don't 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

A 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

Save this program to tape or disk before you run it.

```
10 REM -----JUMPING J
15 COLOR 0,1 : PRINT "{RED}{CLR}" :REM B
   LACK SCREEN, CLEAR SCREEN
20 X=18 :Y=7 : D=1
25 FOR J=1 TO 6: FOR I=1 TO 6
29 COLOR 5,7 : GOSUB 100 :REM BLUE COLO
   R, DRAW J
34 COLOR 5,1 : GOSUB 100 :REM BLACK COL
   OR, ERASE J
40 Y=Y-D : NEXT I :REM MOVE TO N
   EXT POSITION
55 D=-D : NEXT J :REM REVERSE D
   IRECTION
60 PRINT "{YEL}" : END
100 REM -----DRAW THE J
104 PRINT "{CLR}{5 DOWN}"
105 FOR L=1 TO Y:PRINT:NEXT L
109 FOR K=1 TO 5
110 PRINT TAB(X);" [+]":NEXT K
120 PRINT TAB(X);" [ ] [ ]"
130 PRINT TAB(X);" [2 +]" :RETURN
```

The picture is the letter *J*. The subroutine starting in line 100 draws the *J*. Before you GOSUB 100, pick the color you want the *J* to be. Look at line 29 and at line 34. If you pick black, the subroutine erases a *J* from that spot because the background is black.

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 40 says that each picture will be drawn at the spot where *Y* is larger than the last *Y* by the amount *D*.

Assignment 24

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 3) through all the colors to yellow (color 8) 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 discusses the AND, OR, and NOT relations and the numeric values for true and false.

There are two abstract ideas in this lesson that may give difficulty. One is that true and false have numeric values of -1 and 0 . Any expression that is of the form of an assertion (*phrase A*) has a numeric value of 0 or -1 . This number is treated just like any other number. It can be stored in a numeric 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 numeric value between IF and THEN. If the number is zero, the assertion is false, but 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

Save this "Teenager" program to disk before you run it:

```
10 REM < AND, OR, NOT >
20 COLOR 0,1 : PRINT "{CLR}{DOWN}"
30 INPUT " NAME{SHIFT-SPACE}";N$ : PRINT
40 INPUT " AGE";A : 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}"
```

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?

OR is used in line 55. 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's 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 does, 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 numeric 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.

IF 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:

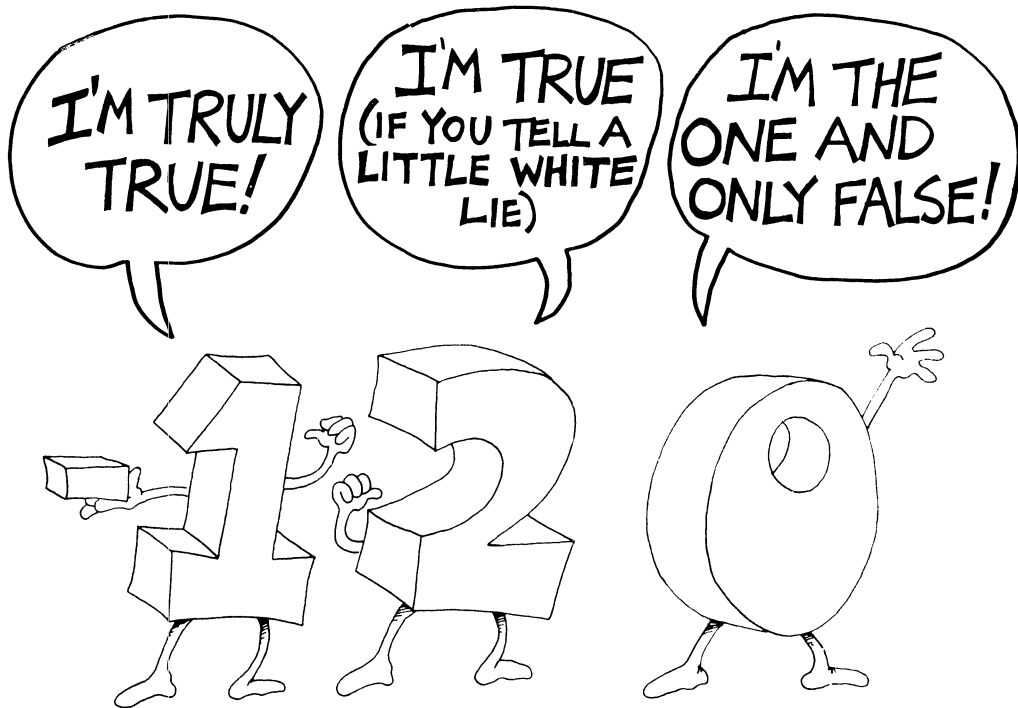
```
IF 22 THEN PRINT "TRUE"
```

Rule: In an IF, the computer looks at *phrase A*.

If it is zero, the computer says "*phrase A* is false." It skips the statement after THEN.

If it is not zero, the computer says "*phrase A* is true." It obeys the statement after THEN.

The IF statement tells little white lies. True is supposed to be the number -1 , but IF stretches the truth to say "true is anything that is not false," that is, any number that is not zero is true.

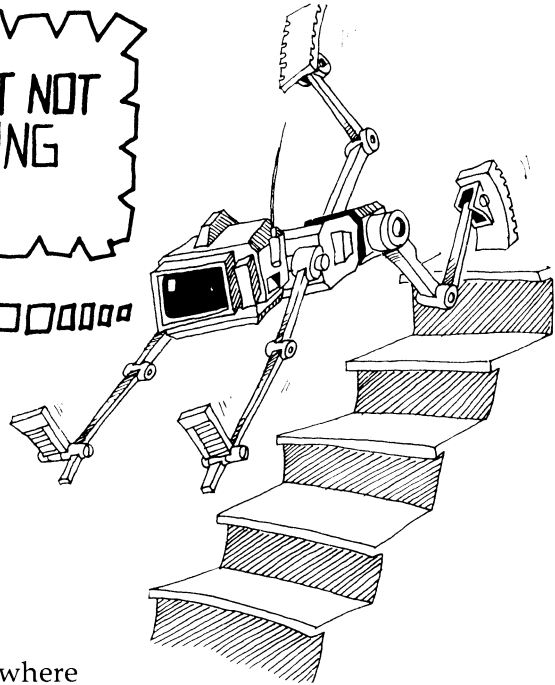
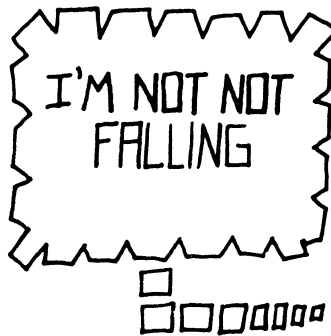


What Does NOT Mean?

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
61 REM "I DON'T HAVE NO..."
62 REM MEANS "I DO HAVE..."
```

Save it to disk.



Careful: NOT (TRUE) = FALSE

and NOT (FALSE) = TRUE

work only for real TRUE, the one where

True = -1.

They don'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'll 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 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. Will the word JELLYBEAN 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*, and *nothing*, and count them. If there are two negative words, there is a double negative. Test the program on this sentence: "COMPUTERS AIN'T GOT NO BRAINS."

Instructor Notes 26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN

This lesson demonstrates the functions

LEFT\$ MID\$ RIGHT\$ LEN

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 special cases, especially those that lead to error messages. It is better that extensive explanations not clutter up the text. If students experience 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$,1)
10 F$=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 changes the word COMPUTER into PUTERCOM.

Lesson 26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, LEN

You already know how to glue strings together:

```
55 A$="CON" + "CAT" + "EN" + "ATION"  
60 PRINT A$
```

The real name for gluing is *concatenation*.

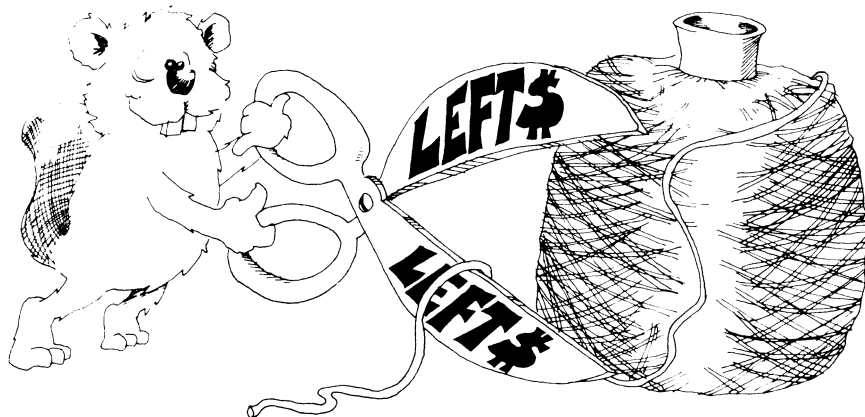
Concatenation means “make a chain.” Maybe we should call them chains instead of strings.

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: LEFT\$() needs two things inside the parentheses:

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)
```

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. The order of the 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:

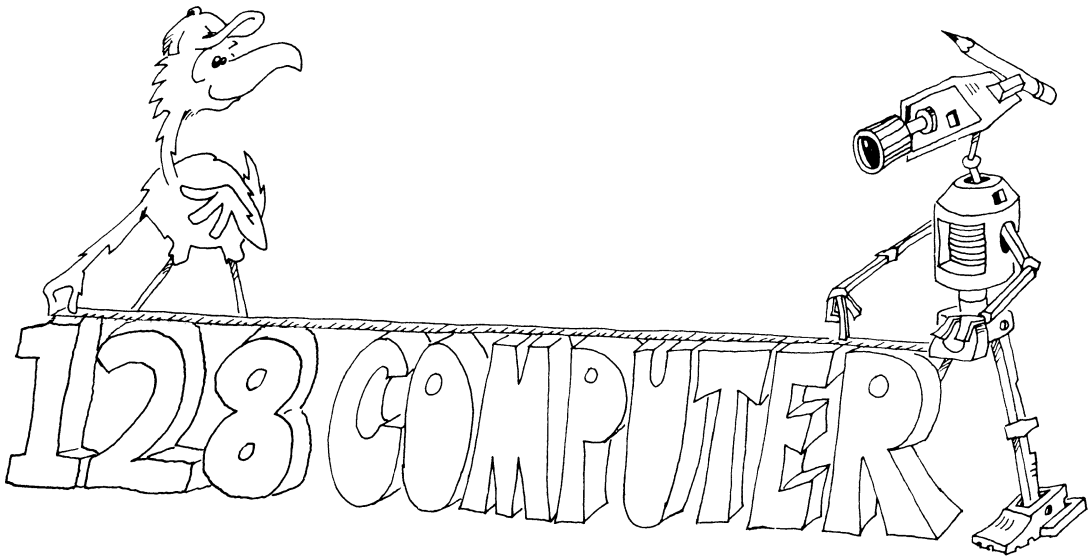
```
55 IF I=4 THEN PRINT:PRINT R$+L$:PRINT
```

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$)
50 PRINT " THE STRING:{DOWN}"
;2 PRINT " '";N$;"' {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$ : 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$:REM SAVE ONLY LETTERS
80 NEXT I : PRINT "{CLR}{DOWN} ";T$
```

Line 60 snips one letter at a time from the middle of the string.

A Real Clock

The Commodore 128 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 this:

```
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 INPUT TI\$. We used large line numbers so that 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 load or save to tape or disk.

Assignment 26

1. Write a secret cipher-making program. 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. Here's an example:

THIS IS A TEST. becomes HTSII S AETTS.

2. Write a question-answering program. 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 from A to C to use in a menu. Change the letter to a number from 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 looks at two functions, `STR$()` and `VAL()`, and also gives a general review of the concept of functions.

`STR$()` takes a number and makes a string that represents it.

`VAL()` does just the opposite, taking a string and making a numeric 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 nonnumeric 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 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 that 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 this line:

```
65 PRINT TI$
```

Lesson 27. Switching Numbers with Strings

This lesson explains two functions—VAL() and STR\$().

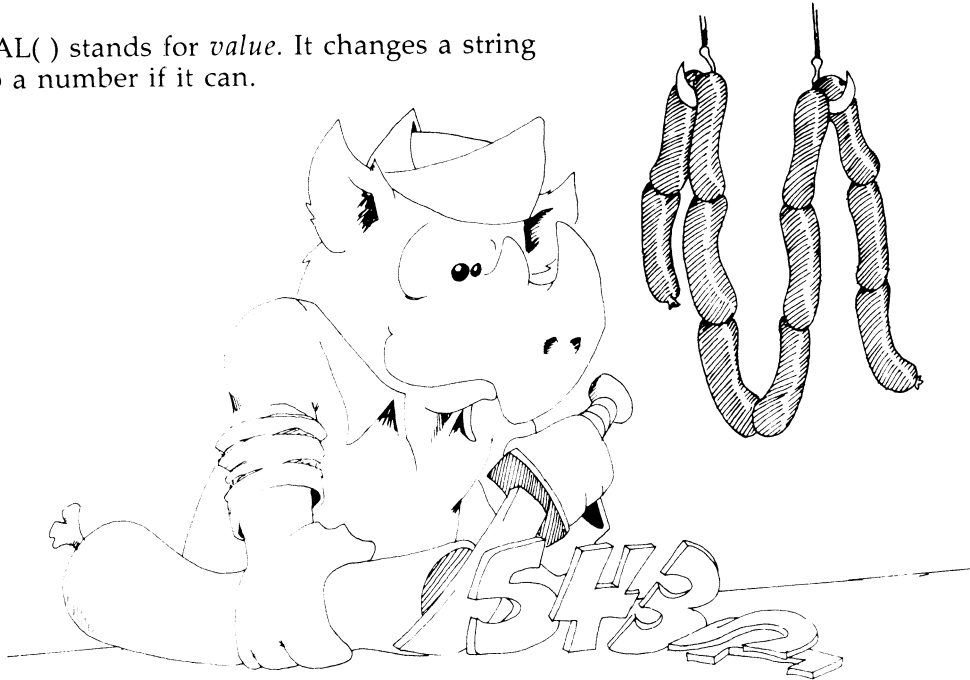
Making Strings into Numbers

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}" : COLOR 0,1
25 PRINT " GIVE ME A NUMBER"
26 INPUT 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. For 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: 1Ø LET D=LEN(CS\$)

Incorrect: 1Ø 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, numeric values go in numeric 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 _____

Explain what is wrong in each of these lines:

10 INT(Q)=65 _____

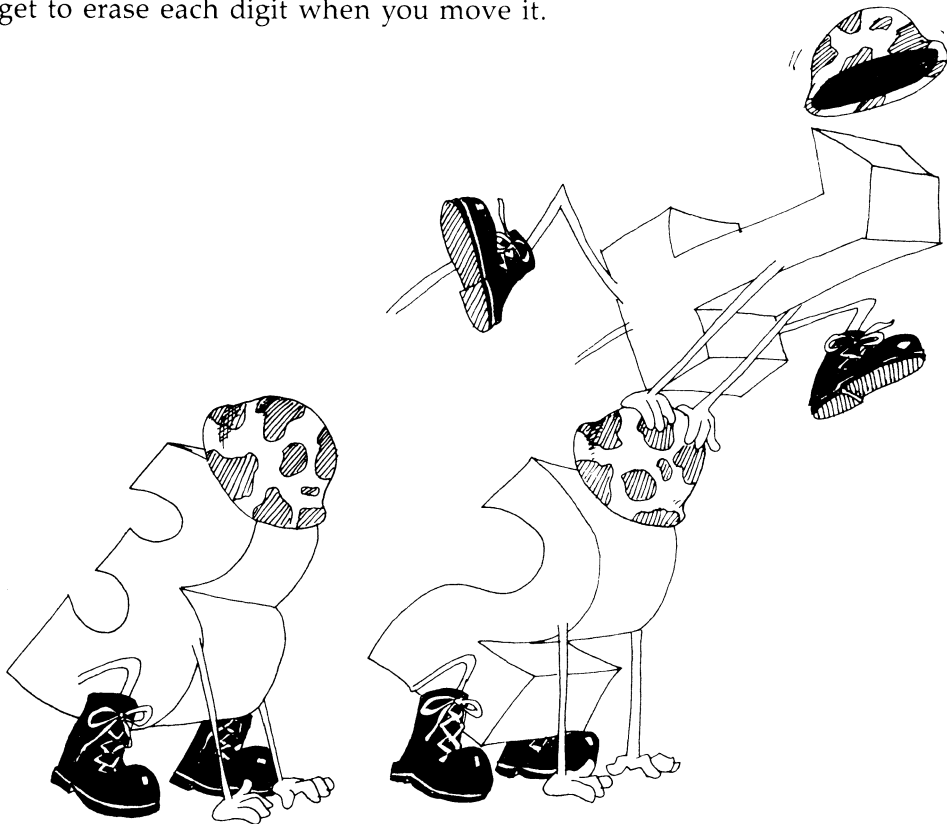
10 D\$=LEFT(R\$,1) _____

10 PW\$=VAL\$(F) _____

10 PRINT CHR\$ _____

Assignment 27

1. Write a program that asks for a number. Then make another number that is backwards from the first, and add them together. Print all three numbers to look 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 Joystick

This lesson introduces the joystick and shows how to control two simultaneously moving objects on the screen.

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 powerful method of moving the arrow shot by a diamond, while the diamond can still move and be controlled by the joystick.

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 whether the square is background. If not, then jump to a subroutine that asks the ASCII number of the target type so that appropriate action can be taken.

Graphics games may grow to be rather long. BASIC is a little slow for such games. You can achieve maximum speed if the working part of the program comes first, and the initialization part is at the end, reached by using GOTO or GOSUB from early in the program. Lesson 32 discusses this format.

For maximum speed, avoid repeatedly converting numerical constants to floating point as the program runs. That is why variables like $Z5=5$, $DD=40$, and so forth, are defined in the Cupid program. This practice is especially important when converting multidigit numbers.

The fastest and most versatile way to make graphics is to use sprites. Lesson 31 covers this topic.

Questions

1. What statement detects joystick 1?
2. How can you detect that the firebutton has been pressed?

-
-
3. How can you detect that the arrow has hit a heart?
 4. What keeps the diamond from moving off the top of the screen?
 5. What keeps the arrow from moving off the left of the screen?
 6. Explain how the computer knows to move the arrow and the diamond at the same time.

Lesson 28. Action Games and the Joystick

Plug the joystick into the socket on the right side of the computer, nearest the front. This is joystick port number 1.

Enter:

```
10 REM JOYSTICK
15 COLOR 0,1
16 K=JOY(1)
20 PRINT K
30 GOTO 16
```

Run it. Wiggle the joystick and see what numbers are printed.

```
1 up
left 7 0 3 right
5 down
```

Press the firebutton while in each of the five positions. "Fire" adds 128 to whatever the joystick was reading before.

A Moving Diamond

Make the joystick control a moving diamond.

Enter this program:

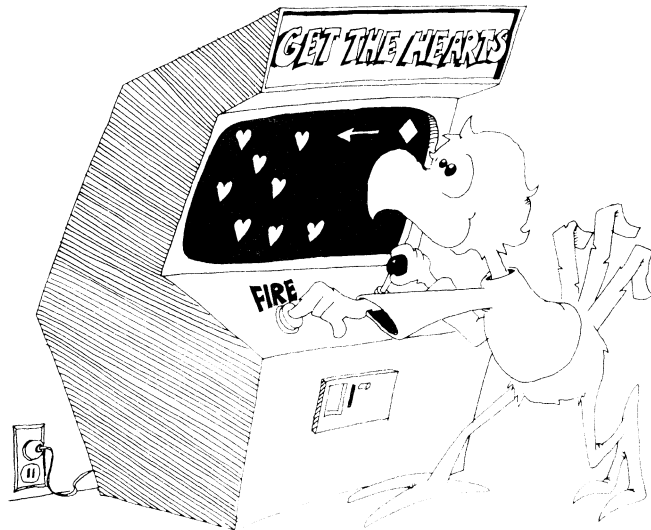
```
10 REM CUPID
11 PRINT "{CLR}"           :REM CLEAR SCREEN
12 COLOR 0,3              :REM RED SCREEN
22 SB=1982                 :REM SCREEN BOTTOM CORNER
24 SC=1102                :REM SCREEN TOP CORNER
26 Y =SC                  :REM START SPOT OF DIAMOND
27 Z0=0:Z1=1:Z5=5        :REM CONSTANTS FOR SPEED
28 DI=90                  :REM DIAMOND CHARACTER
29 SP=32                  :REM SPACE CHARACTER
30 DD=40                  :REM ONE LINE UP OR DOWN
50 K =JOY(1)              :REM LOOK AT THE JOYSTICK
55 IF K=Z0 THEN D=0       :REM DON'T MOVE?
```

```

60 IF K=Z1 THEN D=-DD :REM MOVE UP?
65 IF K=Z5 THEN D= DD :REM MOVE DOWN?
90 POKE Y,SP :REM ERASE OLD DIAMOND
120 Y =Y+D :REM SPOT FOR NEW ONE
130 IF Y>SB THEN Y=SB :REM OFF BOTTOM?
140 IF Y<SC THEN Y=SC :REM OFF TOP?
170 POKE Y,DI :REM POKE NEW DIAMOND
199 GOTO 50 :DO AGAIN

```

Save it before you run it.



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 moves when you shift the joystick. Otherwise, it stops.

Line 55 says, "If the joystick is centered, set D to zero." 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 joystick is tilted up. If it is, D is set to -40 . This means that the next diamond will be put one line higher than the present diamond.

Line 65 sets D to 40 if the joystick is tilted down. 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 firebutton on the joystick.

Add these lines to the "Cupid" program:

```
32 S=0:X=Y           :REM ARROW IS READY
70 IF K>Z5 AND S=0 THEN S=1   :REM SHOOT ARROW
75 IF S>0 THEN GOSUB 300      :REM ARROW STILL GOES
300 REM SHOOT
310 IF S=Z1 THEN X=Y-Z1:POKE X,31:S=2:RETURN
315 POKE X,SP           :REM ERASE OLD ARROW
316 IF S=39 THEN S=0       :RETURN
320 X=X-Z1:S=S+Z1       :REM PUT NEW ARROW
330 POKE X,31           :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 on.

The move arrow subroutine at line 300 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 increases by one, and X, the position of the arrow, decreases by one, so the arrow goes to the left.

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 zero, meaning that the diamond is allowed to shoot again. (The arrow was erased in line 315. No new arrow has been written, so the screen is empty of arrows.)



The Targets Are Hearts

Add these lines to the program:

```
40 GOSUB 400
400 REM HEARTS
405 FOR L=1 TO 20 :REM MAKE 20 HEARTS
410 J=1024 + INT(RND(8)*25)*40
420 I=INT(RND(8)*30)
430 POKE I+J,83 :REM POKE AT RANDOM ON SCREEN
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 on to. 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 28

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 which starts when the arrow is shot and stops when the arrow ends its flight.
3. Add an explosion sound when the heart pops.
4. Why does the diamond dodge around while the arrow flies? It makes sense only if the hearts are shooting back at the the diamond. Add missiles that try to hit the diamond.

Instructor Notes 29. Music

Lesson 19 treated the SOUND statement. This lesson continues the introduction to SID chip features with the PLAY, TEMPO, and VOL commands used in making music.

Review the lesson on sound effects (lesson 19). We need DATA statements, and you may also want to review them (lesson 18).

Read the *Commodore 128 Personal Computer System Guide* for more advanced features of SID, for example, the ENVELOPE and FILTER statements, and the use of voice 3 to control some features of voices 1 and 2, such as the frequency for vibrato or glissando.

Questions

1. How do you play F-sharp?
2. How do you change the tempo of the music?
3. Make the computer play four sixteenth notes followed by a quarter note.
4. How do you make the computer sound like a xylophone?
5. Can you play two or three voices at once? How?

Lesson 29. Music

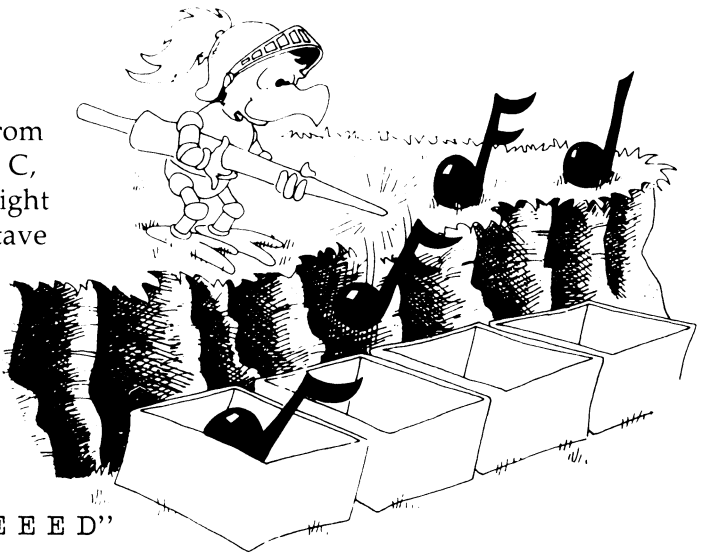
Try this: PLAY "C D E F G A B C"

You hear music, the scale of C major. But the top C repeats the lower one instead of entering the next higher octave. Fix it like this:

PLAY "O3 C D E F G A B O4 C D E F G"

(That's the letter *O*, not zero, in O3.)

The octave number can be any from O0 to O6. The octave changes at C, not at A, as we tin-eared folks might believe. Once you change the octave number, the rest of the notes stay in the new octave until you do another change with the letter *O*.



Put a rest in with the letter *R*:

PLAY "O4 D D D C R E E E D"

Change the volume like this:

VOL 15:PLAY "O4 C C D D R E E F F"

Put any number from 0 to 15 in the VOL statement.

Let's speed it up:

TEMPO 20:VOL 15:PLAY "O4 C C C D"

Use any number from 0 to 255 in the TEMPO statement.

Let's Hear Those Semi-Demi-Quavers

You change note length with these letters:

- W whole note
- H half note
- Q quarter note
- I eighth note (E can't be used because it means the note of pitch E.)
- S sixteenth note

Try this:

```
PLAY "H C C C D R Q C C C W D"
```

When you put H to make half notes, all the following notes are half notes until you pick a new note length.

You're Sharp, I'm Flat

You can make any one note sharp or flat by putting a symbol in front of it.

- C-sharp is #C
- B-flat is \$B (Sorry, there's no true flat sign on the keyboard.)

Note: The sharp or flat is good for only the one note, not like in true music notation where it's good for the whole measure. For example,

```
$B B B is B-flat, B-natural, B-natural.
```

In the same way, you do dotted note lengths by putting a dot in front of the note. Look at this:

```
TEMPO 25:PLAY "O4 H $B $B $B .$B Q A H $B $B $B .$B Q A"
```

A dot on a note makes it half again longer. For example, a dotted half note is as long as a half plus a quarter note.

Guitar or Flute?

You can choose among ten instruments in the PLAY statement:

```
PLAY "O3 T5 C C C D M T4 C C C D"
```

The number following T tells which instrument:

- 0 Piano
- 1 Accordion
- 2 Calliope
- 3 Drum
- 4 Flute
- 5 Guitar
- 6 Harpsichord
- 7 Organ
- 8 Trumpet
- 9 Xylophone

Try each of the numbers in the PLAY line above.

The M in the PLAY statement holds the notes unchanged until the end of the measure. Take it out, and you'll see that the T4 command changes the instrument before the D note can finish playing.

Let's Row That Boat

```
10 REM --- ROW, ROW, ROW YOUR BOAT ---
11 REM NOTES      C,D,E,F,G,A,B,C
12 REM SHARP      #
13 REM FLAT       $
14 REM REST       R
15 REM LENGTHS    W,H,Q,I,S
16 REM DOTTED     .
17 REM WAIT       M (HOLD ALL VOICES TO END OF MEAS
    URE)
18 REM VOICE      V1, V2, V3
19 REM OCTAVE     O0 TO O6
20 REM ENVELOPE   T0 TO T9
21 REM VOLUME     U0 TO U15
22 REM FILTER     0=OFF, 1=ON
23 TEMPO 12
24 FOR I=0 TO 9
26 READ A$: PRINT "{CLR}{3 DOWN} ";A$
27 SLEEP 1
28 PLAY "T"+CHR$(I+48)
30 PLAY "Q.C Q.C QC ID Q.E"
32 PLAY "QE ID QE IF H.G"
```

```

34 PLAY "O5 IC IC IC O4 IG IG IG"
36 PLAY "IE IE IE IC IC IC"
38 PLAY "QG IF QE ID H.C"
39 SLEEP 2
40 NEXT I
98 DATA PIANO,ACORDION,CALLIOPE, DRUM, FLUTE
99 DATA GUITAR, HARPSICHORD, ORGAN,TRUMPET,XYLOPHO
    NE

```

Note: You can put spaces in the PLAY string anywhere you want, or you can have no spaces at all.

Another Round

```

10 REM ----- WHITE CORAL BELLS -----
11 REM NOTES      C,D,E,F,G,A,B,C
12 REM SHARP      #
13 REM FLAT       $
14 REM REST       R
15 REM LENGTHS    W,H,Q,I,S
16 REM DOTTED     .
17 REM WAIT       M (HOLD ALL VOICES TOEND OF MEASU
RE)
19 REM OCTAVE     O0 TO O6
23 TEMPO 8
28 PLAY"T7"
30 PLAY"QA          I#G I#F "          :REM ORGAN
SURE
32 PLAY"Q.E I#C    "                   :REM SECOND
34 PLAY"ID I#F IE  ID  "                 :REM THIRD
36 PLAY"H#C M  T0  "                   :REM FOURTH
38 PLAY"O3 IA  O4  I#C  "               :REM FIFTH (TH
EN CHANGE TO PIANO)
39 PLAY"O3 IB  O4  ID  "                 :REM FIFTH CON
TINUED
40 PLAY"I#C IE  IA  O5  I#C"            :REM SIXTH
42 PLAY"O4  QB  Q#G  "                   :REM SEVENTH
44 PLAY"HA  "                             :REM EIGHTH
49 SLEEP 2
99 REM  ... ..

```

The Three Voices of SID

SID can play two or three instruments at once, each different from the others. The trick is to make them play together, because you have to mix up the notes for the two voices in the same string. There is an example on page 156 of your *System Guide*.

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 to the rectangular array of cells on a TV screen.

The concept of arrays is not too difficult. 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 can also 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 typical use of arrays is to store numbers which cannot neatly be obtained from an equation. An example is 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. Each member of the family is a variable. The F\$ family are string variables:

```
22 F$(0)="DAD"  
24 F$(1)="MOM"  
26 F$(2)="KAREN"
```

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 the parentheses 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 this program.

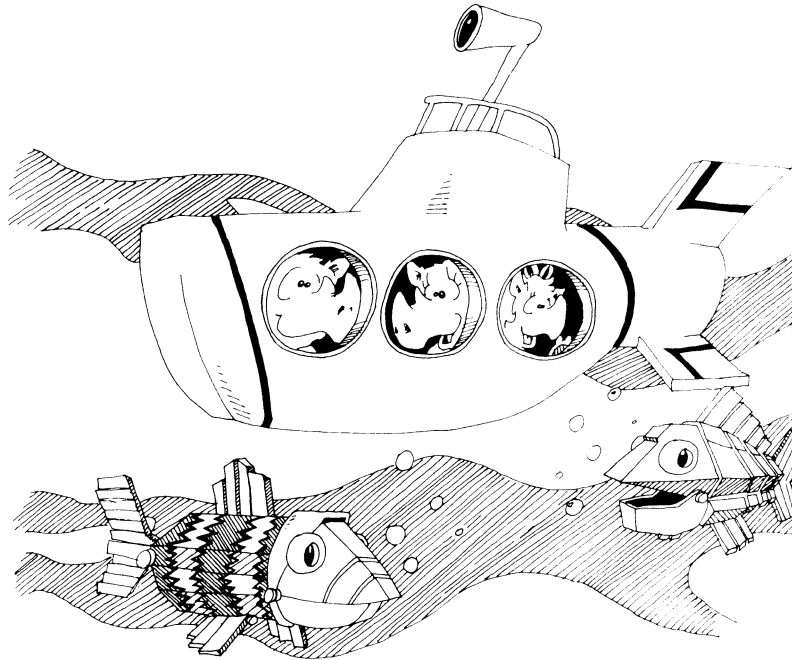
```
10 REM ++ IN A ROW ++
30 DIM A$(4)
35 PRINT "{CLR}{2 DOWN} ENTER A WORD {DOWN}"
40 FOR N=0 TO 4
45 IF N>0 THEN PRINT "ANOTHER {DOWN}"
50 INPUT A$(N)
55 PRINT
60 NEXT N
100 PRINT "IN A ROW {2 DOWN}"
110 FOR I=0 TO 4
120 PRINT I,A$(I)
130 NEXT
```

Now save and run it.

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 this "Phone List" program:

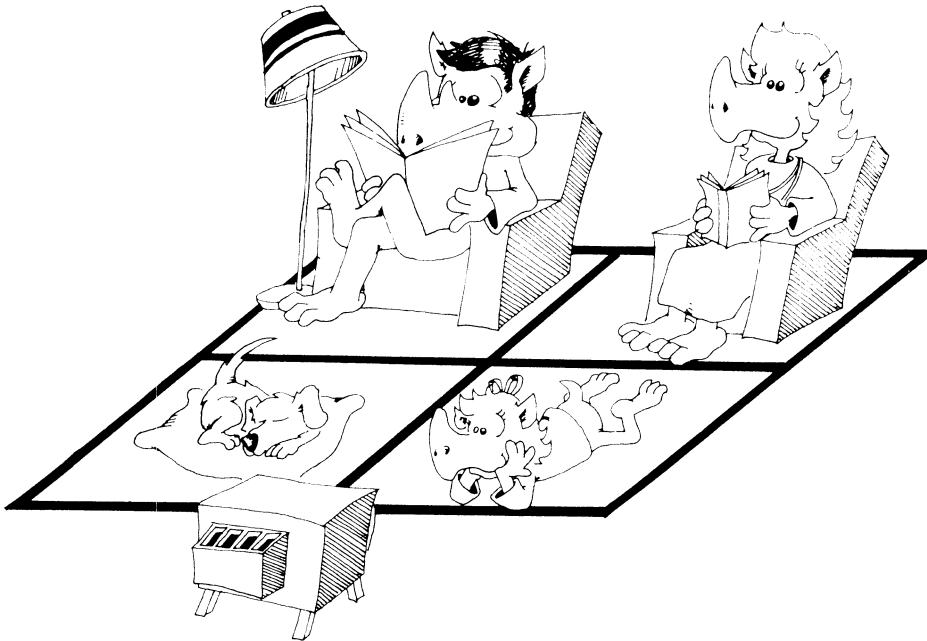
```
10 REM PHONE LIST
15 PRINT "{CLR}"
20 PRINT " ENTER NAMES AND PHONE NUMBERS "
21 PRINT:PRINT " ENTER THE NAME 'END' TO EXIT":PRIN
   T
30 DIM NAME$(20),NUMBER$(20)
35 I=0
50 INPUT " NAME ";N$
55 IF N$="END" THEN END
56 NA$(I)=N$
60 INPUT " NUMBER ";NU$(I)
65 PRINT
70 I=I+1:GOTO 50
```

Run it. Type END to stop the program. Save to 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 listed below asks you to put eight chess queens on the chessboard in such a way that no queen is attacked by any other. If you’re not familiar with chess, look up the queen’s moves 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. "Eight Queens" was written 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 for Action Graphics

The VIC chip in the Commodore 128 controls eight sprites—powerful bitmapped graphics objects (24 bits wide and 21 bits high.)

This lesson demonstrates making, storing, and using sprite figures.

To draw the sprite figure in its 24×21 dot rectangle, call SPRDEF to enter a special sprite drawing mode. There you find a gray rectangle on the screen. Consult your *System Guide* for the commands to draw the sprite in the rectangle, save it to memory, and exit the mode.

In a program, you can call up a sprite with the SPRITE statement, thereby declaring its colors and size. You can set the sprite moving with a MOVSPR statement. It will continue to move at a set speed in a set direction until you command otherwise. You don't need to write a loop that commands the movement of the sprite step by step. The sprite movement is efficient of time and program statements.

Those portions of a sprite in which the bit is zero are transparent, showing whatever objects (other sprites or screen graphics) are behind them. For each sprite, the programmer can choose whether it passes in front of or behind objects in the static screen display.

After this introduction, you can explore further properties of sprites by studying the *Commodore 128 Personal Computer System Guide*.

Questions

1. How do you draw a picture for sprite 1?
2. How do you put sprite 1 in the center of the screen?
3. How do you color sprite 4 red?
4. How do you tell sprite 2 to move slowly to the right?

Lesson 31. Sprites for Action Graphics

The Commodore 128 computer has eight sprites.

A sprite is a little picture you can put on the screen. You draw the picture and tuck it away in memory. In a program, you can call the sprite and move it around.

Sprite One, Obey Our Wishes

Run this program:

```
10 REM ----- SPRITE -----
12 COLOR 0,1 : PRINT "{CLR}" :REM BLACK, CLEAR SCREEN
20 SPRITE 1,1,3,0,0,0,0 :REM SPRITE 1,ON,RED
30 MOVSPR 1,170,140 :REM APPEAR IN CENTER OF SCREEN
40 MOVSPR 1,0#1 :REM MOVE UP SLOWLY
50 SLEEP 3
60 SPRITE 1,0,3,0,0,0,0 :REM TURN SPRITE OFF
```

This program has two new commands:

SPRITE N,O,C,0,0,0,0

MOVSPR N,X,Y or MOVSPR N,A#S

We will look at only the first three numbers that go in the SPRITE statement. The last four will be zero.

- N The sprite number (1-8)
- O O=0 for sprite off, O=1 for sprite on
- C Color of sprite in the usual numbers 1-16
- X Horizontal position 0-319 (may be offscreen)
- Y Vertical position 0-199 (may be offscreen)
- A Angle in degrees to move
- S Speed of movement (0-15)

The MOVSPR command has two forms. In one, you give the X and Y positions of the sprite on the screen. In the other, you tell the sprite to move at angle A with speed S. You can use any crazy angle you want, like 23 degrees. But here are some angles you should know by heart:

0	up
90	right
180	down
270	left

The Artist Takes Over

The sprite we moved around in the “One Sprite” program was drawn by the computer. We want to draw our own. Here is how:

Enter: `SPRDEF`

A gray rectangle appears on the screen. You are supposed to draw a sprite in it. The computer asks “SPRITE NUMBER?” Type 1. The gray screen changes to the bar pattern of sprite 1. You also see the sprite on the right in the correct size.

Press the CLR/HOME and SHIFT keys to erase the bar sprite. Then move the cursor to where you want a dot. Press the 2 key to make the dot.

Use these keys to finish drawing the picture:

CLR/HOME	Erases the sprite picture
Four CRSR keys	Move the cursor around in the sprite
2	Draws a dot in the sprite at the cursor
1	Erases a dot if there was one at the cursor
SHIFTed RETURN	Saves the sprite into memory and prints the SPRITE NUMBER ? prompt
RETURN	At the SPRITE NUMBER? prompt only, returns you to the usual edit mode

Draw a border for the sprite.

When your picture is done, press the SHIFT and RETURN keys to save the picture for sprite 1. Then press RETURN to go back to the edit mode.

Run the "One Sprite" program again and see your new sprite on the screen.

Moving the Sprite

Run:

```
10 REM ----- MOVE SPRITE -----
12 COLOR 0,1 : PRINT "{CLR}"           :REM BLACK, CLEAR
    R SCREEN
14 A=0 : S=1 : O=1 : C=3               :REM INITIAL MOTION
    {SPACE}UP AT SLOW SPEED
30 MOVSPR 1,170,140                   :REM CENTER SPRITE ON
    N THE SCREEN
35 PRINT " MOVE <I,J,K,M,S,G>"
36 PRINT " COLOR <2 TO 9>"
37 PRINT " ON=1, OFF=0"
38 PRINT " END PROGRAM <E>"
50 GETKEY A$                           :REM USER COMMANDS S
    PRITE
51 IF A$="E" THEN SPRITE 1,0,1,0,0,0,0:END
52 IF A$="I" THEN A= 0:D$="UP          "
53 IF A$="K" THEN A= 90:D$="RIGHT      "
54 IF A$="M" THEN A=180:D$="DOWN       "
55 IF A$="J" THEN A=270:D$="LEFT       "
56 IF A$="S" THEN S= 0:D$="STOP        "
57 IF A$="G" THEN S= 1:D$="GO          "
58 IF A$="0" THEN O= 0:D$="OFF         "
59 IF A$="1" THEN O= 1:D$="ON          "
60 CC=VAL(A$)
61 IF CC>1 THEN C=CC :D$="COLOR       "
70 PRINT "{HOME}{5 DOWN} ";: PRINT D$; :REM PRINT ACTION
80 SPRITE 1,O,C                        :REM SPRITE 1, ON OR
    OFF, COLOR C
81 MOVSPR 1,A#S                        :REM MOVE IT AT ANGLE
    E A, SPEED S
90 GOTO 50
```

Line 50 waits until you press a key. If the key is:

S for STOP the sprite stops
G for GO the sprite goes in the same direction as before

These two letters change the speed, S, that we put in line 81.

If the key is:

I	the sprite goes up
K	the sprite goes right
M	the sprite goes down
J	the sprite goes up

These four keys change the angle, A, we put in line 81.

If the key is:

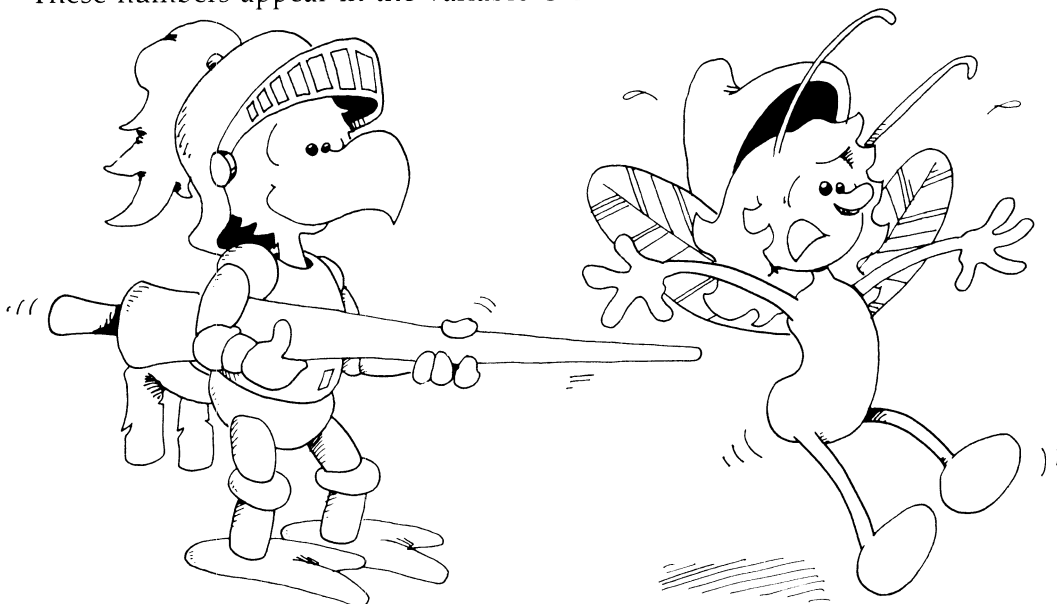
E	the sprite disappears and the program ends.
---	---

If the key is any other letter, nothing happens. But the value of CC in line 60 is 0. (You learned that in lesson 27).

If the key is:

0	the sprite turns off (you can't see it)
1	the sprite turns on again

These numbers appear in the variable O in line 80.



If the key is a number from 2 to 9, the sprite turns the color belonging to the number. This number appears in the variable C in line 80.

Assignment 31

1. Use SPRDEF to make a smiley face for sprite 1. Then draw a frowning face for sprite 2. Write a program to make first the smile, then the frown show in the middle of the screen.
2. Make both sprites appear on the screen at the same time and let them move around randomly.
3. Now that you know how, draw other faces and write a program that changes a face in the middle of the screen slowly from a big smile, to a small smile, then to a puzzled look, then to a frown, then to an angry face.



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 spaghetti programs 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 them.

Instructions and help should be available. Prompts need to be given. Beginners need complete prompts, but experienced users prefer brief prompts.

It's hard to teach the writing of user-friendly programs. Success depends mostly on the programmer's attitude. 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 friendly programming. Becoming acquainted with the desirability of friendly programming and using 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 the 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
- old stuff erased 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 that you are a user of your own programs, too. Be kind to yourself.

Programs Have Three Parts

1. "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

2. Main loop

- controls the order in which tasks are done
- calls subroutines to do the tasks



3. Subroutines

perform 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 this outline and use it to start each new program that you write.

Put the Main Loop at the Beginning

Put the main loop near the front of your program because it will run faster there.

Put Starting Stuff at the End

Put the starting stuff near the end of your program. It may be the biggest part of your program, and you may keep adding to it as you write. This will make your program more user-friendly. The starting stuff doesn't need to run fast.

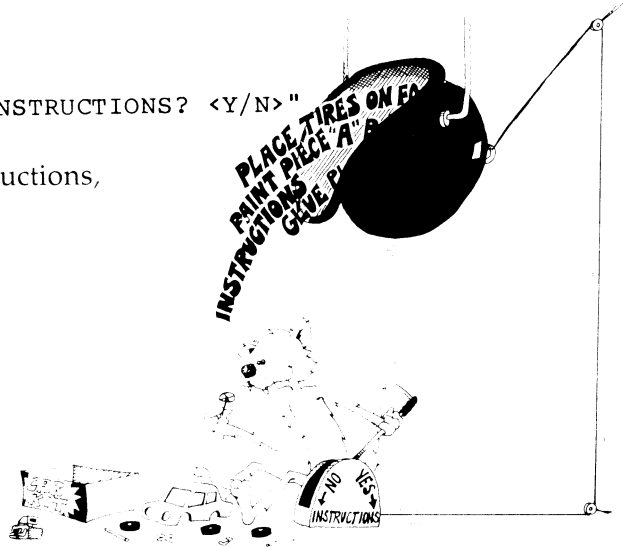
Put Subroutines in Three Places

Put subroutines that must run fast between lines 2 and 99. Put starting stuff subroutines after line 2999. The rest can go between lines 200 and 999.

Information, Please

```
380 PRINT "DO YOU WANT TO SEE INSTRUCTIONS? <Y/N>"
```

This question lets a beginner see instructions, and others can say no.



Tie a String Around the User's Finger

Use a prompt to remind users what choices they have.

In this example

```
<Y/N>
```

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 the user having to press RETURN.

```
380 PRINT "DO YOU WANT TO SEE INSTRUCTIONS? <Y/N>"
382 GETKEY R$
384 IF R$="Y" THEN 600
```

Don't Give Users a Headache

Scrolling gives headaches.

Your screen will scroll if you continue printing after you reach 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.

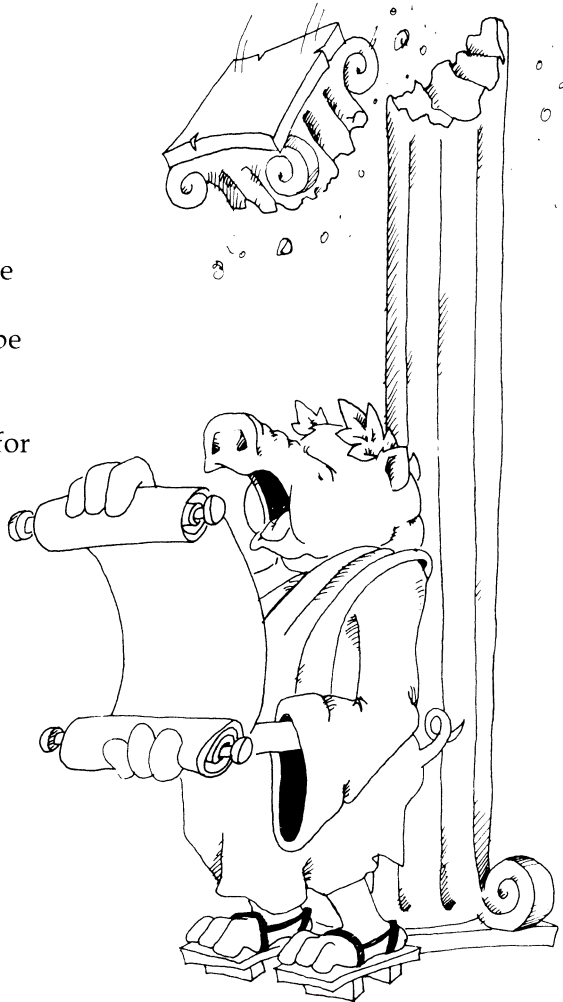
Use delay loops to keep the writing on the screen while the user reads it.

Set Traps for Errors

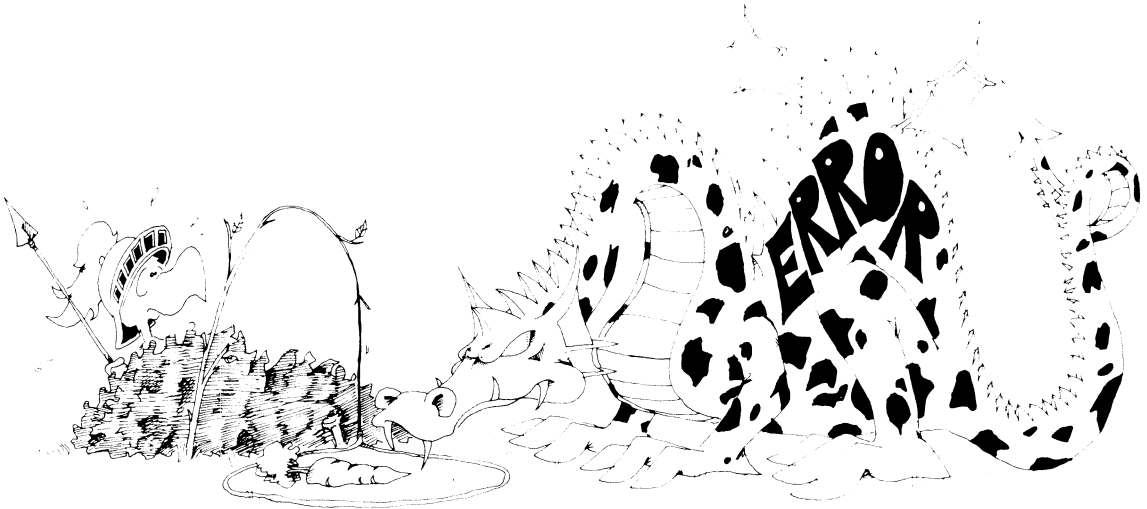
Add this line to the above lines:

```
386 IF R$ <> "N" THEN 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 to try again. The line following line 386 should continue the program if the user chooses not to see the instructions.



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.

`DRAGONETBCFHijklmpqsuvwxyz` cipher alphabet

`ABCDEFGHIJKLMNopqrstuvwxyz` normal alphabet

The user chooses to code or decode from a menu.

Instructor Notes 33. Debugging, STOP, CONT

Since the “sigh and moan” technique of fixing programs 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.

An inexperienced debugger feels hopeless inertia when a program doesn’t work right. Rather than sitting and staring, it is more useful to try some changes. Any changes are better than none, but random changes are 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 helps find these. Delay loops, PRINT statements, and STOP statements help students 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 STOP and 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 a 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 TRIP
20 PRINT "{CLR}"
25 R=INT(RND(8)*200)
30 FOR I=0 TO 200
40 IF I=R THEN STOP
50 NEXT I
```

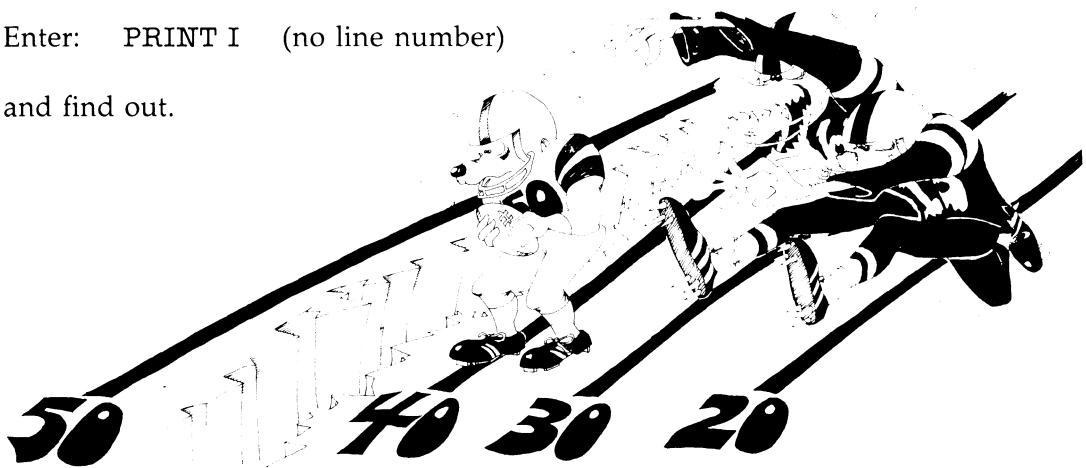
The program will stop, and the computer will print a message:

```
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

Try entering the command CONT.

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.

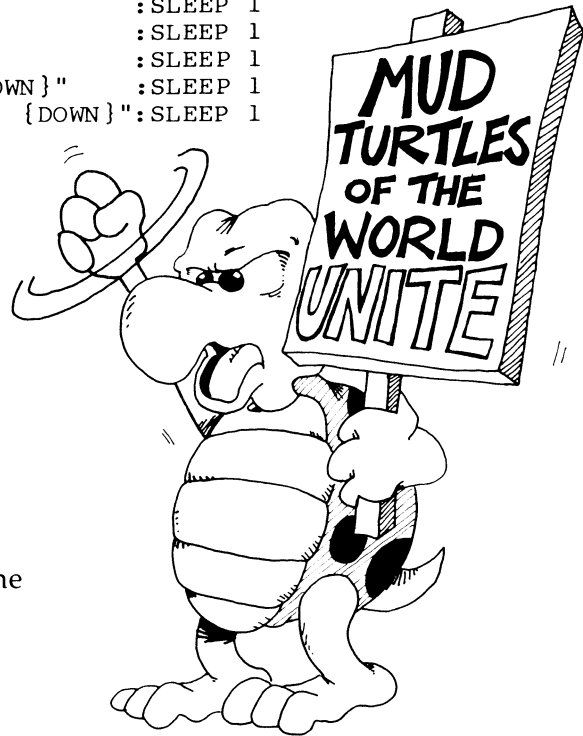
You can use STOP to debug your program.

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}"           :SLEEP
   {SPACE}1
20 PRINT "MUD {DOWN}"           :SLEEP 1
22 PRINT "  TURTLES {DOWN}"     :SLEEP 1
24 PRINT "    OF {DOWN}"        :SLEEP 1
26 PRINT "      THE {DOWN}"     :SLEEP 1
28 PRINT "        WORLD {DOWN}" :SLEEP 1
30 PRINT "          UNITE! {DOWN}":SLEEP 1
40 GOTO 10
```



Pressing the RUN/STOP key stops the program wherever it is. It prints

BREAK IN LINE XX

and enters the edit mode (XX is the line number where it stops).

The command CONT starts the program again at the same spot.

What Do You Do After You Stop?

Put STOP in whatever part of your program is not working right. Then run the program. After it stops, look to see what happened.

(You can 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:

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 a line number) the statement

GOTO XX

(XX is the line number where you want to restart).

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.

CONT can restart a program that was stopped only with a break from a STOP statement or by pressing the RUN/STOP key.

But RUN, RUN XX, and GOTO XX can also start a new program.

Debugging

Little errors in your program are called bugs.

If your program doesn't run right, do these four things:

1. If the computer printed an error message, it tells what line it stopped on. Careful, the mistake may really be in another line.



-
-
2. If the computer just keeps running, but doesn't do the right thing, stop it and add some PRINT lines that will tell what is happening.
 3. Or you can put STOP statements in the program.
 4. If the program runs so fast that you can't tell what is happening, put in some delay loops to slow it down.



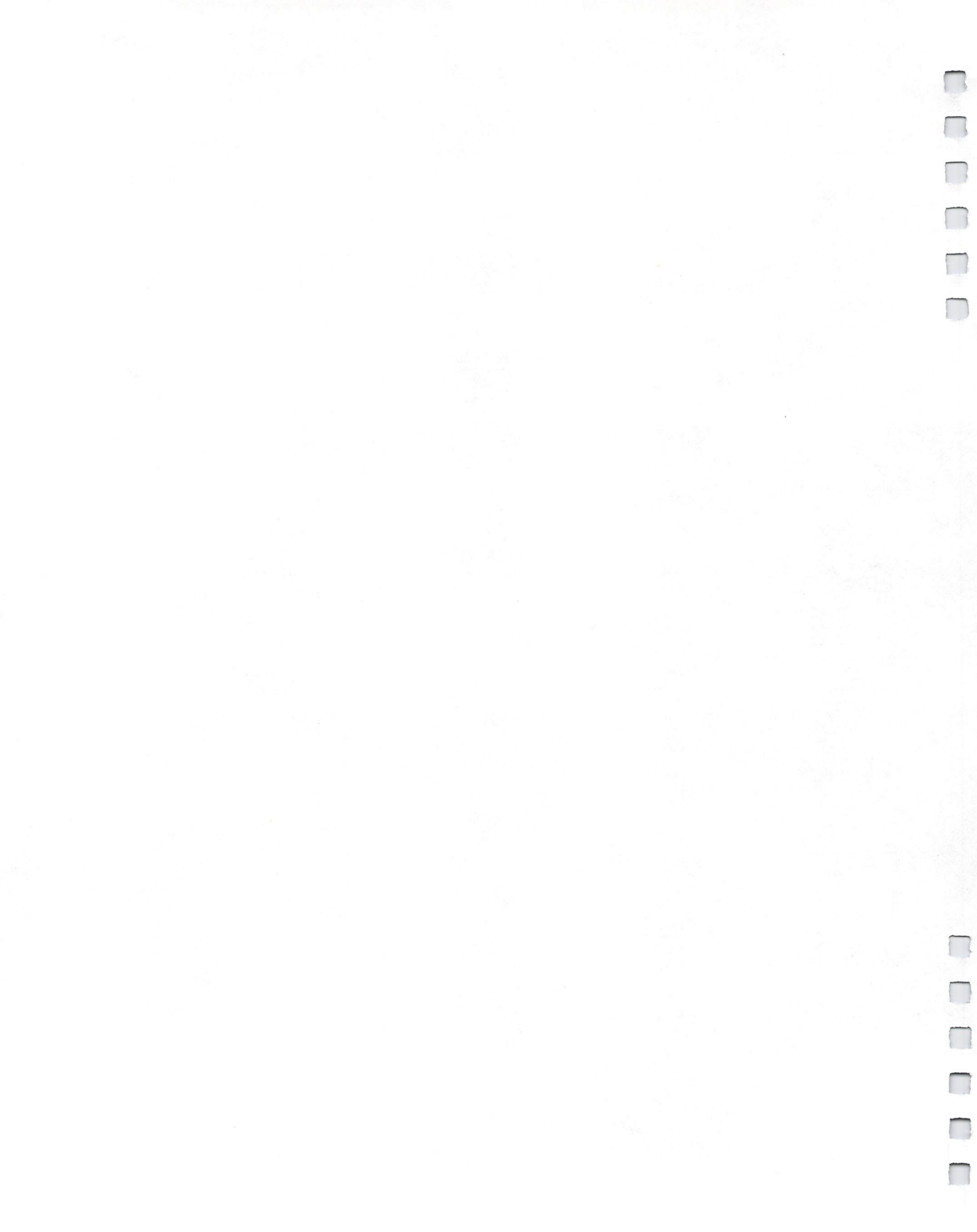
After you've fixed the program, take the PRINT lines, the STOPs, and the delay loops out of the program.



Assignment 33

1. Go back to one of the programs you wrote earlier and debug it. Use the ideas in this chapter. If none of your earlier programs needs debugging, then, congratulations! You are a splendid programmer.
2. Write a "Shape Shifter" program. A shape shifter is a demon that starts in one shape and shifts around its parts to make other shapes.

Appendices

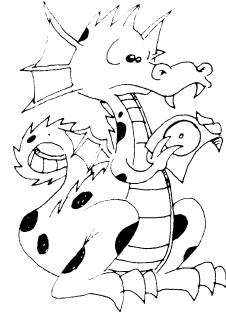


Appendix A

Disk Usage

Instruct your student on care of disks. Especially:

1. Do not touch the brown or gray magnetic disk through the oblong holes.
2. Do not bend the disk.
3. Insert and remove the disk carefully from the drive.
4. Always put the disk back in its protective cover after use.
5. Do not spill food or drink on the disk. In fact, it's better not to snack while at the computer.



Prepare a Disk for the Student

You should prepare a special disk for each student's exclusive use. All the programs that each 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.

These instructions describe the 1571 disk drive. If you use a 1541 drive, the statements about a green light and handle should be changed to a red light and door. The commands are the same for all drives.

You must use the disk HEADER command to format a new, fresh-from-the-box disk before you can store information on it. If you have an old disk which you are willing to erase, you can HEADER it instead of a brand-new disk.

Put the disk in the drive like this: 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 twist the drive handle down until it clicks.

Now enter:

```
HEADER "STUDENT DISK" IS1, D0
```

The drive will clatter and whirl, the green light will come on, and after about a minute, the light will go off and the drive will stop.

If the green 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.

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 no more than 16 characters, and has 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:

```
CATALOG
```

You should see:

```
Ø "STUDENT DISK          " S1 2A
```

All but the zero will appear in reverse letters on the screen. The 2A identifies the kind of Commodore equipment used.

When you twist the handle to take the disk out, the disk should automatically be pushed out an inch. This makes it easy for you to pull it out.

Practice Makes Perfect

This would be a good time to turn to lesson 14 and practice saving and loading programs to the disk.

Appendix B

BASIC Reserved Words

ABS	AND	APPEND	ASC	ATN	AUTO			
BACKUP	BANK	BEGIN	BEND	BLOAD	BOOT	BOX	BSAVE	BUMP
CATALOG	CHAR	CHR\$	CIRCLE	CLOSE	CLR	CMD	COLLECT	COLLISION
COLOR	CONCAT	CONT	COPY	COS				
DATA	DCLEAR	DCLOSE	DEC	DEF FN	DELETE	DIM	DIRECTORY	DLOAD
DO	DOPEN	DRAW	DS	DS\$	DSAVE	DVERIFY		
EL	ELSE	END	ENVELOPE	ER	ERR\$	EXIT	EXP	
FAST	FETCH	FILTER	FN	FOR	FRE			
GET	GET#	GETKEY	GO64	GOSUB	GOTO	GO TO	GRAPHIC	GSHAPE
HEADER	HELP	HEX\$						
IF	INPUT	INPUT#	INSTR	INT				
JOY								
KEY								
LEFT\$	LEN	LET	LIST	LOAD	LOCATE	LOG	LOOP	
MID\$	MONITOR	MOVSPR						
NEW	NEXT	NOT						
OFF	ON	OPEN	OR					
PAINT	PEEK	PEN	PLAY	POINTER	POKE	POS	POT	

PRINT PRINT# PRINT USING PUDEF

QUIT

RCLR RDOT READ RECORD REM RENAME RENUMBER RESTORE RESUME

RETURN RGR RIGHT\$ RND RSPCOLOR RSPPOS RSPRITE RUN RWINDOW

SAVE SCALE SCNCLR SCRATCH SGN SIN SLEEP SLOW SOUND

SPC SPRCOLOR SPRDEF SPRITE SPRSAV SQR SSHAPE

ST STASH STEP STOP STR\$ SWAP SYS

TAB TAN TEMPO THEN TI TI\$ TO

TRAP TROFF TRON

UNTIL USING USR

VAL VERIFY VOL

WAIT WHILE WIDTH WINDOW

XOR

Appendix C

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 DISK

You tried to HEADER a disk without putting a two-character ID after the file-name. Or you have defective disk.

BAD SUBSCRIPT

You made an error using an array, for example,

```
DIM A(5,5):A(1,1,1)=77  Wrong number of subscripts
DIM A(5): A(14)=7      Subscript was larger than 5
```

BEND NOT FOUND

You used IF-THEN BEGIN or IF-THEN-ELSE BEGIN and the computer can't find a BEND to match the BEGIN.

BREAK

The program encountered a STOP command or the RUN/STOP key was pressed.

CAN'T CONTINUE

You used the CONT command when it was not allowed.

CAN'T RESUME

You used a RESUME statement without first using a TRAP statement.

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.

DIRECT MODE ONLY

The line in the program contains a command that can be used only in the edit mode.

DIVISION BY ZERO

You divided by zero, or you divided by a variable whose value was zero.

EXTRA IGNORED

A comma was found in an INPUT statement. The program continues anyway.

FILE DATA

You read bad data from a file stored on a tape or disk.

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 the number of a file already open.

FILE READ

An error during a disk read. Perhaps you opened the door while the green light was on.

FORMULA TOO COMPLEX

You wrote a string expression that needs to be split into parts.

ILLEGAL DEVICE NUMBER

You tried to use a device incorrectly. Refer to your *System Guide* for more information.

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:

Incorrect: L=VAL(R)	Correct: L=VAL(R\$)
Incorrect: TAB(-3)	Correct: TAB(3)
Incorrect: SPEED=400	Correct: SPEED=255

Wrong arguments may be a string where a number is needed, a number where a string is needed, a negative number where a positive one is needed, or a number that is bigger than allowed.

LINE NUMBER TOO LARGE

You used a line number larger than 63999.

LOAD

Something is wrong with the program on tape.

LOOP NOT FOUND

You used a DO statement without a LOOP statement.

LOOP WITHOUT DO

You used a LOOP statement without a DO statement.

MISSING FILE NAME

Your DLOAD, DSAVE, or SCRATCH command had no filename.

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
```

NO GRAPHICS AREA

You must use the GRAPHIC command before using commands like DRAW, BOX, CIRCLE, etc.

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 sub-routines 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 you put a # in a name, etc.

TOO MANY FILES

You may only have 10 files OPEN at one time.

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.

UNIMPLEMENTED COMMAND

The command you used is not supported by BASIC 7.0.

UNRESOLVED REFERENCE

You used a GOTO or GOSUB with a line number for a line which doesn't exist.

VERIFY

The program on tape or disk is not exactly like the program in the computer's memory.

Appendix D

Glossary

argument

The variable, number, or string that appears in the parentheses of a function. For example,

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 128; they are at the bottom right of the keyboard. Each has two arrows. Two other keys have arrows on them and print them as characters. Finally, there are four duplicate cursor keys at the top of the keyboard.

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 numeric value of 0 or -1. *See also* expression, false, logic, phrase A, true. Examples:

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

An acronym for 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

To start up the computer from scratch—an easy thing to do with modern computers that have startup 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, or a printer.

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 a subroutine, or it does the calculation that a function is for. It 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* CRLF, linefeed

character

Letters, digits, punctuation marks, and the space are characters. So are the graphics characters you see as pictures on some keys.

clear

To erase. Used in the terms “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. On a typewriter, it is simply called a carriage return. *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 128 computer:

INPUT cursor A flashing square on the screen
PRINT cursor Invisible, it “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 so that you can 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 re-type parts of it to correct it.

edit mode

When `READY.` and the flashing cursor show, you are in the edit mode. The computer waits for commands or program lines to be entered.

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 that 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

A branch point in a program. *See also* branch.

function

BASIC has a number of built-in functions. Each function has a name followed by parentheses. One or more arguments are in the parentheses. 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
```

function keys

Four keys above the numeric keypad on the right of the keyboard (F1–F8). These keys contain specific preprogrammed instructions after the computer is booted. They can be reprogrammed to perform a set of instructions.

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.

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. Program lines contain the instructions for the computer to follow.

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)	assertion
7<=INT(Z)	expression
LEN (Q\$ + "R") + 2;"RAT"	expression
Q\$;"R"	expression

INT(Z)	function
LEN(Q\$)	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

To retype 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 DLOAD or 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 three kinds of loops: GOTO statements; FOR-NEXT loops, homemade loops that use IF statements with a loop variable; and DO loops.

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.

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

There are two meanings. (1) 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. (2) In machine language programming, a monitor is a control program. Your Commodore 128 has a built-in machine language monitor. To use the monitor from BASIC, enter MONITOR.

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

Arithmetic operations are addition (+), subtraction (-), multiplication (*), and division (/). 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*

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. A program is a set of instructions for the computer to follow to achieve some result.

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
```

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. *See also* Appendix B.

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

An area of memory used by the computer for storing information it needs to complete instructions from the central processing unit. This information can be pictured as a tower of building blocks, with each item of data placed on top of the preceding data item. The last item put on the tower is the first one taken off.

starting stuff

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 used. 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 value which can be changed while a program is running. Each variable is stored in a box in memory which is reserved for that variable. 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. Then it 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.

Appendix E

Answers to Assignments

Lesson 1

```
1 REM      A1-3
10 REM NEW FRIEND
20 PRINT "HI, THERE"
30 PRINT "COMPUTER"
```

Lesson 2

```
1 REM      A2-2
10 REM COLORED NAMES
20 COLOR 0,1
30 PRINT "{WHT}M{RED}I{CYN}N{PUR}D{GRN}A"
35 PRINT "{YEL} ANNE"
40 PRINT "{RVS}{RED} CARLSON{WHT}"
```

Lesson 3

```
1 REM A3-5
10 REM BIRDS
15 COLOR 0,1
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}"
```

Lesson 4

```
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 "        *****"
```

Lesson 5

```
1 REM A5-1 DAN
10 REM BY DAN CLARK, AGE 10
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 PRINT A$;" PLAYS ";B$
```

```
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 PRINT G$;
52 PRINT " PLAYS ";
54 PRINT T$
```

```
1 REM A5-3
10 REM THREE LITTLE WORDS
20 COLOR 0,1
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$
```

Lesson 6

```
1 REM A6-1
10 REM SILLY GOOSE
12 COLOR 0,1
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!"
```

```
1 REM A6-2
10 REM *** FAVORITE***
12 COLOR 0,1
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}{CYN}NOW LET'S SEE WHAT IS IN BOX C
   $"
55 PRINT "{DOWN}IT IS:"
56 PRINT
60 PRINT C$
```

Lesson 7

```
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$
```

Lesson 8

```
1 REM A8-2
10 REM T E E N T I M E S
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 COLOR 0,1
20 PRINT "{CLR}{WHT}"
22 PRINT
24 PRINT
30 PRINT "{YEL} BRIAN"
31 PRINT
32 PRINT
33 PRINT
34 PRINT
35 ?PRINT
40 PRINT "{GRN} STEVE"
50 PRINT
51 PRINT
52 PRINT
53 PRINT
54 PRINT
55 PRINT
90 GOTO 30
```

Lesson 9

```
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 COLOR 0,1
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$
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}"

```

Lesson 10

```

1 REM      A10-1
10 REM BIRTHDAY
15 COLOR 0,1
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 COLOR 0,1
20 PRINT "{CLR}{WHT}{3 DOWN}"
30 PRINT "  GIVE ME A NUMER{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=2000000 AND N2=3000000
96 REM THE ANSWER IS 6E+12 IN SCIENTIFIC
97 REM  NOTATION

```

Lesson 11

```

1 REM      A11A-1
10 REM NICKNAMES
15 COLOR 0,1
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 %$#!"
15 COLOR 0,1
20 PRINT "{CLR}{WHT}{3 DOWN}"
24 PRINT " HEY YOU! WHAT IS YOUR NAME1{DOWN}"
25 INPUT N$
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      A11B-1
10 REM SLOW POKE
15 COLOR 0,1
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      A11B-2
10 REM CLOCK
15 COLOR 0,1
20 PRINT "{CLR}{WHT}{3 DOWN}"
23 PRINT "  PRESENT TIME: HR,MIN,SEC?"
24 PRINT
25 INPUT H,M,S
26 LET F=412
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}"

```

Lesson 12

```

1 REM      A12B-3
10 REM I GOT YOUR NUMBER
15 COLOR 0,1
20 PRINT "{CLR}{WHT}{3 DOWN}"
23 PRINT "  GIVE ME A NUMBER 0 TO TEN{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 COLOR 0,1
16 LET G=0
22 PRINT "{CLR}{YEL}{3 DOWN}"; " PLAYER 2: TURN YOU
R BACK{DOWN}{CYN}"
24 PRINT " PLAYER 1: THINK OF A CARD,
{DOWN}{CYN}"
25 PRINT " PRESS RETURN{DOWN}{CYN}"
26 INPUT A$
32 PRINT "{CLR}{3 DOWN}"; " PLAYER 1: TELL ME THE
{SPACE}SUIT{DOWN}{PUR}"
34 PRINT " HEARTS, DIAMONDS, CLUBS, OR SPA
DES{DOWN}"
36 INPUT S$
42 PRINT "{CYN}{DOWN}"; " NOW TELL ME THE VALUE
{SPACE}1 TO 13"
44 INPUT V
52 PRINT "{CLR}{YEL}{3 DOWN}"; " PLAYER 2: GUESS TH
E CARD SUIT{DOWN}"
54 PRINT " HEARTS, DIAMONDS, CLUBS, OR SPA
DES?{DOWN}"
60 LET G=G+1
61 INPUT GS$
62 IF GS$=S$ THEN 70
64 PRINT "{GRN}{DOWN}"; " WRONG, GUESS AGAIN
{DOWN}{PUR}"
68 GOTO 60
70 PRINT "{CLR}{YEL}{3 DOWN}"; " RIGHT! NOW GUESS T
HE VALUE 1-13{DOWN}"
80 LET G=G+1
81 INPUT GV
82 IF GV=V THEN 90
84 IF GV<V THEN PRINT "{CYN} WRONG, GUESS HIGHER
{DOWN}"
86 IF GV>V THEN PRINT "{PUR} WRONG, GUESS LOWER
{DOWN}"
89 GOTO 80
90 PRINT "{CLR}{YEL}{3 DOWN}"; " CONGRATULATIONS!
{DOWN}"
94 PRINT " YOU GUESSED IT IN ONLY";G;"TRIE
S{DOWN}"
96 INPUT " PLAY AGAIN <Y/N>";A$
97 IF A$="Y" THEN 10

```

Lesson 13

```
1 REM A13-1
10 REM DICE
15 POKE 53281,0
20 PRINT "{CLR}{3 DOWN}"
22 LET D1=1+INT(RND(8)*6)
24 LET D2=1+INT(RND(8)*6)
30 LET D=D1+D2
35 PRINT "{DOWN}";TAB(5);"THE FIRST DIE ";D1
40 PRINT "{DOWN}";TAB(5);"THE SECOND DIE ";D2
45 PRINT "{DOWN}";TAB(5);"THE DICE ";D
60 FOR T=1 TO 1000:NEXT T
99 GOTO 20
```

Lesson 15

```
1 REM{6 SPACES}A15-2
10 REM !!! VACATION !!!
12 COLOR 0,1:PRINT "{CLR}{WHT}"
20 REM HEADING
22 PRINT "{DOWN} VACATION"
24 PRINT "{DOWN}{3 SPACES}CHOOSING"
26 PRINT "{DOWN}{5 SPACES}PROGRAM"
28 PRINT "{DOWN}{GRN} CHOOSES YOUR VA?CATION BY THE
   {SPACE}AMOUNT"
30 PRINT "{DOWN}{YEL} YOU WANT TO SPEND"
32 FOR T=1 TO 1500: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
   {6 SPACES}WALLOW, MI"
62 P$=" ENTER A PICKLE EATING CONTEST IN
   {10 SPACES}SCRATCHYBACK, TN"
63 REM ETC.
69 Z$="BUY A COSY YACHT AND CRUISE THE CARIBBEAN S
   EA"
70 IF D<.5 THEN PRINT M$:GOTO 90
71 IF D<5{2 SPACES}THEN PRINT N$:GOTO 90
72 IF D<50 THEN PRINT P$:GOTO 90
90 REM ENDING OF PROGRAM, SAY SOMETHING
91 REM{3 SPACES}FOR A CLOSING
```

```

1 REM{5 SPACES}A15-3
10 REM ?????? CRAZY ??????
12 POKE 53281,0:PRINT"{CLR}{WHT}{5 DOWN}"
20 INPUT"{UP} WHAT IS YOUR NAME";N$
30 Z=INT(RND(8)*3)+1
31 A$=" YOU ARE ONE BRICK SHORT OF A FULL LOAD"
32 B$=" YOU HAVE BATS IN YOUR BELFRY"
33 C$=" YOU HAVEN'T GOT BOTH OARS IN THE WATER"
37 F$="{HOME}{8 DOWN}"
38 PRINT"{CLR}{5 DOWN}"
41 IF Z=1 THEN PRINT " ";N$;F$;A$
42 IF Z=2 THEN PRINT " ";N$;F$?;B$
43 IF Z=3 THEN PRINT " ";N$;F$;C$

```

Lesson 16

```

1 REM      A16-1
10 REM FLYING BALL
12 COLOR 0,1: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 COLOR 0,4: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

```

Lesson 17

```

1 REM      A17A-1
10 REM RABBITS
12 COLOR 0,1:PRINT"{CLR}{CYN}{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}{27 DOWN}"
15 N$= " STANISLAUS MAZURSKY"
20 FOR I=24 TO 1 STEP -1
30 PRINT N$
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}{CYN}{3 DOWN}"
20 FOR I=1 TO 3
21 COLOR 0,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 COLOR 0,1
```

Lesson 18

```
1 REM 18
10 REM ---- FAMILY ----
15 COLOR 0,1
20 PRINT"{CLR}{3 DOWN} A RELATION:{DOWN}""
22 PRINT" FATHER, MOTHER,{DOWN}"
23 PRINT" SISTER, BROTHER, ETC.{DOWN}"
24 INPUT R$ : PRINT
25 FLAG$="NO"
30 FOR I=1 TO 10
35 READ T$ : READ N$
40 IF T$=R$ THEN PRINT" ";N$;"{DOWN}":FLAG$="YES"
50 NEXT I : RESTORE
60 IF FLAG$="NO" THEN PRINT"{RED} DON'T HAVE ONE
{BLK}{DOWN}"
70 SLEEP 2 : GOTO 20
100 DATA FATHER,EDWARD
101 DATA MOTHER,LOUISE
102 DATA SISTER,ANNE
103 DATA GRANDMOTHER,ADA
104 DATA GRANDMOTHER,CONSTANT
105 DATA GRANDFATHER,REGINALD
107 DATA AUNT,KARN
108 DATA AUNT,SUSAN
109 DATA UNCLE,HARRY
110 DATA COUSIN,BOB
```

Lesson 19

```
1 REM A19-2
10 REM FLY AWAY
20 PRINT"{CLR}{BLK}{22 DOWN}";
25 PRINT"{PUR}"
30 PRINT"UQI{3 LEFT}";
32 SOUND 1,2000,5
40 FOR T=1 TO 100:NEXT T
45 PRINT" {2 LEFT}{UP}";
50 PRINT"JQK{3 LEFT}";
52 SOUND 1,2500,5
60 FOR T=1 TO 100:NEXT T
65 PRINT" {2 LEFT}{UP}";
70 GOTO 30
```

Lesson 20

```
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 2000:NEXT I
15 PRINT"{CLR}"
20 FOR I=1 TO RND(0)*2000+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 2000:NEXT I
99 GOTO 15
```

Lesson 21

```
1 REM          A21-2
10 REM ALL COLORS
20 FOR I=1 TO 16
25 COLOR 0,I
26 PRINT"                SCREEN COLOR";I
30 FOR J=1 TO 16
40 COLOR 4,J
42 PRINT"BORDER COLOR";J
44 FOR T=1 TO 50:NEXT T
50 NEXT J,I
```

Lesson 22

```
1 REM A22-1
10 REM COLORED BOX
12 COLOR 0,1 : PRINT "{CLR}{GRN}"
20 INPUT "WHAT COLOR <2-16>";C
21 IF C<2 OR C>16 THEN 20
30 INPUT "WHAT SIZE <2-16>";S
31 IF S<2 OR S>16 THEN 30
32 T=INT(S*1.4)
35 COLOR 5,C
40 PRINT "{CLR}{6 DOWN}          ";
42 H$="*****"
50 FOR I=1 TO T
52 PRINT "*";:NEXT I : PRINT
55 FOR I=1 TO S-2
57 PRINT TAB(10);"*";TAB(T+9);"*":NEXT I
60 FOR I=1 TO T
62 PRINT TAB(10);"*";:NEXT I
90 PRINT? "{YEL}"
```

Lesson 23

```
1 REM{5 SPACES}A23-1
10 REM MENU MAKER
12 PRINT"{CLR}{3 DOWN}"
20 PRINT"{4 SPACES}WHICH COLOR"
21 PRINT
22 PRINT"{5 SPACES}<R> RED "
24 PRINT"{5 SPACES}<Y> YELLOW"
26 PRINT"{5 SPACES}<G> GREEN "
28 PRINT"{5 SPACES}<B> BLUE "
29 PRINT
30 FOR T=1 TO 300:NEXT T
31 GET C$:IF C$="" THEN 31
35 IF C$="R" THEN C=3
36 IF C$="Y" THEN C=8
37 IF C$="G" THEN C=6
38 IF C$="B" THEN C=7
40 COLOR 0,C
50 PRINT"COLOR"
70 FOR T=1 TO 900:NEXT T
80 GOTO 12
99 REM MAKE A STAR OF THIS COLOR
```

```

1 REM A23-2
10 REM :::: SILLY SENTENCES ::::
12 PRINT "{CLR}{3 DOWN}"
16 PRINT " SILLY SENTENCES {DOWN}"
17 PRINT " WANT INSTRUCTIONS? {DOWN}"
18 GETKEY Y$ : IF Y$="Y" THEN GOSUB 100
20 PRINT "{GRN} THE SUBJECT: {DOWN}"
21 PRINT " END WITH A PERIOD"
24 GETKEY L$ : IF L$="." THEN 30
28 S$=S$+L$ : GOTO 24
30 S$=S$+" "
32 PRINT "{DOWN}{YEL} THE VERB:"
34 GETKEY L$ : IF L$="." THEN 42
38 S$=S$+L$ : GOTO 34
42 S$=S$+" "
50 PRINT "{WHT}{DOWN} THE OBJECT:"
52 GET L$:IF L$="" THEN 52
54 IF L$="." THEN 70
56 S$=S$+L$ : GOTO 52
70 S$=S$+L$
85 PRINT "{CLR}{YEL}{4 DOWN}";S$ : END
100 REM
105 PRINT "{CLR}{3 DOWN}"
110 PRINT "? PARTS OF A SENTENCE"
111 PRINT "{DOWN} A THREE PLAYER GAME."
112 PRINT "{DOWN} EACH PLAYER ENTERS PART OF A SENTENCE."
115 PRINT "{DOWN} NO ONE CAN SEE WHAT THE OTHERS ENTER."
120 PRINT "{DOWN} THE FIRST ENTERS THE SUBJECT"
130 PRINT "{DOWN} THE SECOND ENTERS THE VERB"
140 PRINT "{DOWN} THE THIRD ENTERS THE OBJECT"
148 SLEEP 7 : PRINT "{CLR}{3 DOWN}" : RETURN

```

Lesson 24

```
1 REM A24B-1
10 REM ----- SUBROUTINE DEMO.
12 COLOR 0,1 : PRINT "{CLR}{3 DOWN}{CYN}"
21 GOSUB 100
22 GOSUB 200
23 GOSUB 300
26 IF RND(0)<.5 THEN GOSUB 400
90 PRINT "{YEL}"
99 END
100 REM ----- THE FIRST SUBROUTINE
105 PRINT " NOW IS THE TIME "
150 GOSUB 900 : RETURN
200 REM ----- THE SECOND SUBROUTINE
210 PRINT "{DOWN} FOR {RED} RED {YEL} SMOKE"
250 GOSUB 900 : RETURN
299 RETURN
300 REM ----- THE THIRD SUBROUTINE
310 PRINT "{DOWN} TO POUR OUT"
311 PRINT "{DOWN} OF YOUR COMPUTER!"
399 RETURN
400 REM ----- PERHAPS
410 PRINT "{DOWN}{PUR} <I DON'T MEAN IT!>"
450 GOSUB 900 : RETURN
900 REM ----- TIMER
910 SLEEP 1 : RETURN
```

Lesson 25

```
1 REM A25-3
10 REM ----- "AIN'T GOT NO" -----
12 PRINT"{CLR}{3 DOWN}{YEL}"{21 SPACES}:REM INPUT
   {SPACE}SENTENCE
20 PRINT"{2 SPACES}ENTER A SENTENCE {DOWN}"
22 PRINT"{2 SPACES}NO PUNCTUATION EXCEPT APOSTROPH
   E {DOWN}"
32 INPUT S$: S$=S$+" " : L=LEN(S$)
40 NN=0{5 SPACES}: S1=1{6 SPACES}: S2=1
45 FOR I=1 TO L
50 L$=MID$(S$,I,1)
55 IF L$=" " THEN S1=S2:S2=I+1:GOSUB 200
60 NEXT I
65 PRINT"{2 DOWN}" : REM ----- SEA
   RCH FOR NEGATIVE WORDS
70 IF NN=0 THEN PRINT"{GRN}{2 SPACES}NO NEGATIVE W
   ORDS"
72 IF NN=1 THEN PRINT"{GRN}{2 SPACES}A NEGATIVE SE
   NTENCE"
74 IF NN=2 THEN PRINT"{RED}{2 SPACES}DOUBLE NEGATI
   VE {BLK}"
76 IF NN>2 THEN PRINT"{CYN}{2 SPACES}MANY NEGATIVE
   S"
99 SLEEP 2{2 SPACES}:{2 SPACES}GOTO 12
200 REM ----- TEXT WORD
205 LW=S2-S1-1
210 W$=MID$(S$,S1,LW)
220 READ NW$
222 IF NW$="END" THEN RESTORE:RETURN
224 IF W$ = NW${2 SPACES}THEN NN=NN+1
230 GOTO 220
300 REM ----- NEGATIVE
   {SPACE}WORDS
301 DATA NO,NOT,NEVER,NEGATIVE,DON'T,AIN'T,DOESN'T
   ,NOTHING,SHOULDN'T,WOULDN'T
302 DATA NONE,CAN'T,DIDN'T,END
```

Lesson 26

```
1 REM A26-1
10 REM CIPHER MAKER
12 PRINT "{CLR}{3 DOWN}{PUR}"
20 PRINT " CODE MAKER{DOWN}"
25 PRINT " ENTER A SENTENCE{DOWN}"
30 INPUT S$: S$=S$+" " : L=LEN(S$)
33 PRINT
40 FOR I=1 TO L STEP 2
45 P$=MID$(S$,I,2)
50 Q$=RIGHT$(P$,1)+LEFT$(P$,1)
55 L$=L$+Q$
60 NEXT I
65 PRINT "{2 DOWN} HERE IS THE CODED SENTENCE"
70 PRINT "{DOWN} ";L$;"{YEL}"
```

```
1 REM A26-2
10 REM ANSWERER
12 PRINT "{CLR}{3 DOWN}{WHT}"
20 PRINT " ENTER A QUESTION{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 I
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 INPUT X$
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

```

Lesson 27

```

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

```

Lesson 30

```

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

```

Lesson 32

```

1 REM      A32-1
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 '*' SIGN{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

```

Commands, Statements, Functions, and Operators Index

- AND operator 161–68
- CATALOG command 90, 95, 96
- COLOR command 11, 23, 137, 139–41
- CONT command 217–23
- DATA statement 119, 121–24
- DIM statement 195, 197–203
- DLOAD command 90, 94, 96, 105
- DSAVE command 90, 92–93, 96, 105
- END statement 155, 157–60, 219, 220
- FOR statement 113, 115–118
- GET statement 149–54, 215
- GETKEY statement 149–50, 152–54
- GOSUB statement 155, 157–60, 182
- GOTO statement 47, 49–55, 104, 182, 221–22
- HEADER command 228
- IF statement 47, 56, 58–62, 77–80, 103–4, 165–66
- INPUT statement 36, 38–40, 47, 63, 97, 100–101, 151
- INT function 83, 86–89, 179–81
- LEFT\$ function 169, 171–72, 175, 179–81
- LEN function 169, 171, 173, 175, 179
- LET statement 41, 43–46, 63, 66–70, 99, 221
- LIST command 16, 18
- LOAD command 90, 94, 96, 105
- MID\$ function 169, 171, 173–75, 179
- MOVSPR statement 204–9
- NEW command 1, 4, 105
- NEXT statement 71, 113, 115–18
- NOT operator 161, 166–168
- OR operator 161–68
- PEEK function 142, 179, 182
- PLAY command 189–94
- POKE statement 137, 142–48
- PRINT statement 7, 30, 32–34, 36, 47, 63, 68, 99, 217, 221
- READ statement 119, 121–24
- REM statement 1, 7, 21, 104
- RESTORE statement 119, 124
- RETURN statement 155, 157–60
- RIGHT\$ function 169, 171–72, 175, 179–80
- RND function 36, 83, 85–89, 179–80
- RUN command 1, 6, 105, 134, 221–22
- SAVE command 90, 92–93, 96, 105
- SCRATCH command 90, 94, 96
- SLEEP statement 105–6
- SOUND statement 125–29
- SPRDEF statement 204, 206–9
- SPRITE statement 204–9
- STEP function 113, 116
- STOP statement 217–23
- STR\$ function 176, 178–80
- TAB function 71, 73–75
- TEMPO command 189–94
- THEN statement 58–62, 77–80, 102, 165–66
- VAL function 176, 178–80
- VOL command 189–94

1000
1000
1000
1000
1000
1000
1000

1000
1000
1000
1000
1000
1000
1000

1000
1000
1000
1000
1000
1000
1000

1000
1000
1000
1000
1000
1000
1000

Grownups, Too

Don't let the title fool you. *COMPUTE!'s Kids and the Commodore 128* 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 128. 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 128*. Some of the topics included are:

- What to do if you get an error message
- Shortcuts to make programming faster
- How to add sprites to your programs
- Assignments to help you practice what you've learned (with sample answers)
- Techniques to debug your programs
- Adding sound and music to any program
- Creating color graphics
- A glossary of terms
- And dozens of cartoons to make you laugh as you learn

COMPUTE!'s Kids and the Commodore 128 explains everything you need to know to start using and programming your 128. Its concise, yet refreshing style makes computing fun and exciting for every Commodore user.