

KIDS AND THE ATARI



BY
Edward H. Carlson

Department of Physics and Astronomy
Michigan State University

Illustrated by
Paul D. Trap

First Printing January, 1983
Second Printing March, 1983

©1983 by
RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia
ISBN 0-8359-3670-8

COPYRIGHT © 1983 BY DATAMOST, INC.

This manual is published and copyrighted by DATAMOST, Inc. All rights are reserved by DATAMOST, Inc. Copying, duplicating, selling, or otherwise distributing this product is hereby expressly forbidden except by prior consent of DATAMOST, Inc.

The word ATARI and the Atari logo are registered trademarks of ATARI Inc. Atari Inc. was not in any way involved in the writing or other preparation of this manual, nor were the facts presented here reviewed for accuracy by that company. Use of the term ATARI should not be construed to represent any endorsement, official or otherwise, by ATARI Inc.

TABLE OF CONTENTS

	Page
Acknowledgements	2
To The Kids	3
To The Parents	4
To The Teachers	5
About Programming	6
About The Book	7

INTRODUCTION

1 PRINT, NEW, REM, and RUN	9
2 Buzz, Inverse, and String Constants	15
3 List and Memory Boxes	21
4 Backspace, Cursor Keys, Insert, Delete	28
5 String Boxes, DIM, LET	33
6 The INPUT Command	38
7 Tricks with PRINT	43
8 The GOTO Command and Break Key	47
9 The IF Command	51
10 Introducing Numbers	57
11 Delay Loops, Sound	64
12 The IF Command with Numbers	68
13 Random Numbers and the INT Function	73
14 Saving to Tape	78

GRAPHICS, GAMES, AND ALL THAT

15 Some Shortcuts	83
16 Graphics Characters, POSITION	88
17 FOR-NEXT Loops	92
18 Edit and Run Modes, the Calculator	97
19 DATA, READ, RESTORE	102
20 SOUND	107
21 Color Graphics	112
22 ASCII Code	118
23 Secret Writing and the GET Command	122
24 Pretty Programs, GOSUB, RETURN, END	126

ADVANCED PROGRAMMING

25 Keyboard, ON . . . GOTO	132
26 Snipping and Gluing Strings	136
27 Switching Numbers with Strings	142
28 Joystick for Action Games	145
29 Shooting Stars	149
30 Arrays	155
31 Logic: AND, OR, NOT	160
32 User Friendly Programs	166
33 Debugging, STOP, CONT	172
Reserved Words	178
Answers to Assignments	179
Glossary	203
Commands	214
Error Messages	215
Index	217

ACKNOWLEDGEMENTS

My sincere thanks go to Paul Sheldon Foote for suggesting I write this book.

I helped prepare and teach in "The Computer Camp" summer camp at Michigan State University for these last two summers. I am deeply grateful to my fellow teachers and board members at the summer camp: Mark Lardie, Mary Winter, and John Forsyth, each of whom shared their teaching experiences with me and suggested techniques for communicating the material in an effective way.

Mark Lardie has been especially generous in reading the typescript and offering suggestions from his extensive experience in teaching computing to children under a variety of formats.

Remembering the enthusiastic pleasure of the summer camp students has encouraged me during the months spent in preparing this book.

Several families have used the first version of this book in their homes and offered suggestions for improvement. I especially wish to thank Steve Peter and his girls Karen and Kristy; 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.

John Decarli of the Computer Mart in East Lansing has helped in many ways.

My own family has respected my need for long periods in the writing den and for quiet in the house. So my final and heartfelt thanks go to my wife, Louise, and our children Karen, Brian and Minda.

TO THE KIDS

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

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

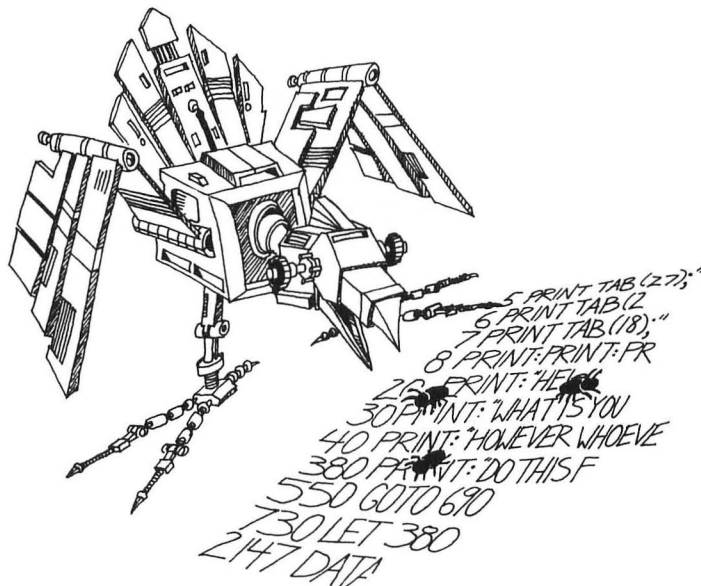
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

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

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

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

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



TO THE PARENTS

This book is designed to teach ATARI BASIC to youngsters in the range from 10 to 14 years old. It gives guidance, explanations, exercises, reviews and "quizzes." Some exercises have room for the student to write in answers that you can check later. Answers are provided in the back of the book for assignments. Your child will probably need some help in getting started and a great deal of encouragement at the sticky places.

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

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



TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches ATARI BASIC on cassette ATARI systems. The lessons contain explanations (including cartoons), examples, exercises and review questions. Notes for the instructor which accompany each lesson summarize the material, provide helpful hints and give good review questions.

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

I view this book as teaching programming in the broadest sense, using the BASIC language, rather than teaching "BASIC." Seymour Papert has pointed out in MINDSTORMS 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 these: "chunking" ideas into "mind sized bites," organizing such modules in a hierarchical system, looping to repeat modules, and conditional testing (the IF...THEN statement).

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



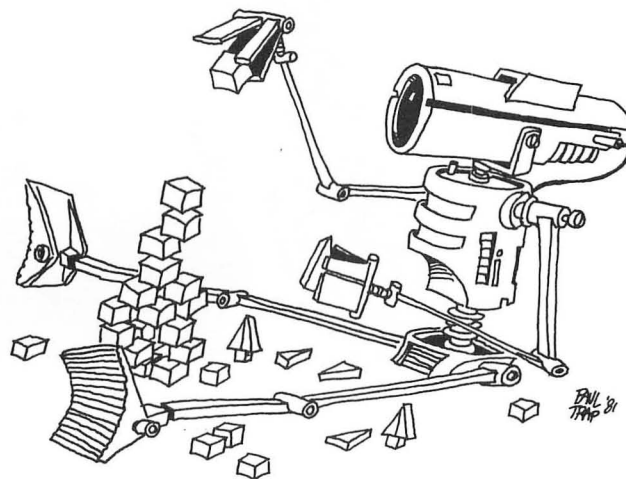
ABOUT PROGRAMMING

There is a common misconception that programming a computer must be very similar to doing arithmetic. Not so. The childhood activities that computing most resembles may be playing with building blocks and writing an English composition.

Like a set of blocks, BASIC uses many copies of a small number of elements (commands) that are combined in rather standardized ways to achieve an original end result. As familiarity with the system grows, a "bag of tricks" is collected by the programmer that allows each command to serve a larger number of functions, just as the child first uses the "triangle block" in making roofs but later finds that two of them make a splendid fir tree.

Like an essay, a program is a finished product that fulfills a specified need. The child writing to the theme "How I spent my summer," adopts one of several working styles. The beginner may be hung-up in how to hold the pencil and how to spell. The same child a few grades later will just start writing, not spending much time in forming good paragraphs much less in planning the overall structure of the composition. With maturity comes freedom to move back and forth among the levels of concern, now thinking about the overall form, and a few moments later paying attention to word choice or punctuation.

Computing does have some similarities to arithmetic as seen by most children. There are rigid rules to learn: procedures in arithmetic but only syntax in programming. Even the tiniest mistake makes the whole result "wrong." (A more effective attitude in programming is that "wrong" results are partly right, and need debugging, a normal and expected activity.) However, the limited scope for creativity in arithmetic contrasts sharply with the emphasis on creativity in program writing.



Programming offers general education advantages not easily found elsewhere in a child's experiences. The plasticity of the form, words on a screen that are created and destroyed by the touch of a key, allows effort to be concentrated on the central features to be learned. These features are balanced between analysis (why doesn't it work as I want) and synthesis (planning on several size scales, from the program as a whole down through loops and subroutines to individual commands). Learning on the computer is efficient of effort. Errors of syntax are automatically pointed out by the computer.

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

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

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

The book starts with a bare bones introduction to programming, leading quickly to the point where interesting programs can be written. See the notes for lesson 5, THE INPUT COMMAND, for an explanation. The central part of the book emphasizes more advanced and powerful commands. The final part of the book continues this, but also deals with broader aspects of the art of programming such as editing and debugging, and user friendly programming.

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

The ATARI computer has many sophisticated features, such as player-missile graphics and the Central I/O subsystem, that cannot be covered in this manual. These are discussed very clearly in the book De Re Atari.

Lessons 14: SAVING TO TAPE, and 18: EDIT AND RUN MODES can be studied anytime after the first lesson.

INTRODUCTION

INSTRUCTOR NOTES 1 PRINT, NEW, REM, AND RUN

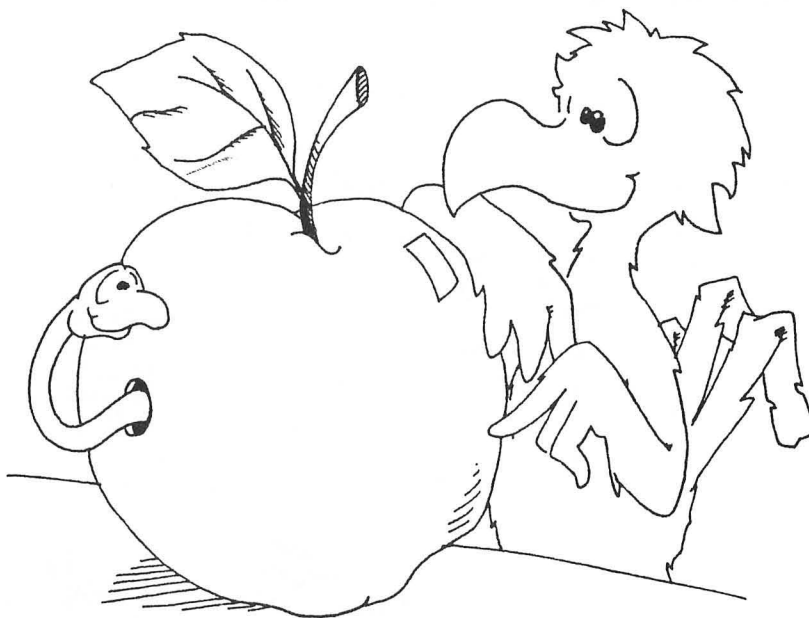
This lesson is an introduction to the computer.

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. RETURN key.
3. The computer understands only a limited number of commands.
4. In this lesson, NEW, PRINT, REM, RUN.
5. What a program is. Numbered lines.
6. Clearing the screen.
7. Memory can be cleared with NEW.
8. What is seen on the screen and what is in memory are different. This may be a hard concept for the student to grasp at first.
9. RUN makes the computer go to memory, look at the commands in the lines (in order) and perform the commands.
10. One can skip numbers in choosing line numbers, and why one may want to do so.

QUESTIONS:

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



LESSON 1 PRINT, NEW, REM, AND RUN

GETTING STARTED

Put the BASIC COMPUTING LANGUAGE cartridge in the ATARI computer. The label side is towards you. If you have an ATARI 800, put the cartridge in the left slot.

Turn on the computer. You will see:

READY

Below READY is a square. This square is called the "cursor." When you see it on the screen, the computer wants you to type something.

"Cursor" means "runner." The square runs along the screen showing where the next letter you type will appear.

TYPING

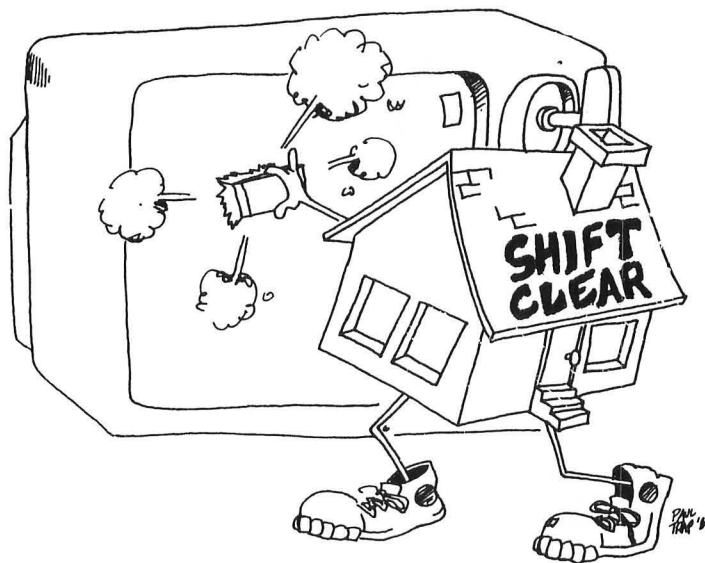
Type some things. What you type shows on the TV screen.

ERASING THE SCREEN

Two keys together erase the TV screen.

Hold down one of the SHIFT keys and press the CLEAR key. The screen is erased.

CLEAR stands for "clear the screen." "Clear" means the same as "erase."



COMMAND THE COMPUTER

Try this. Type: `GIVE ME CANDY`

and press the RETURN key.

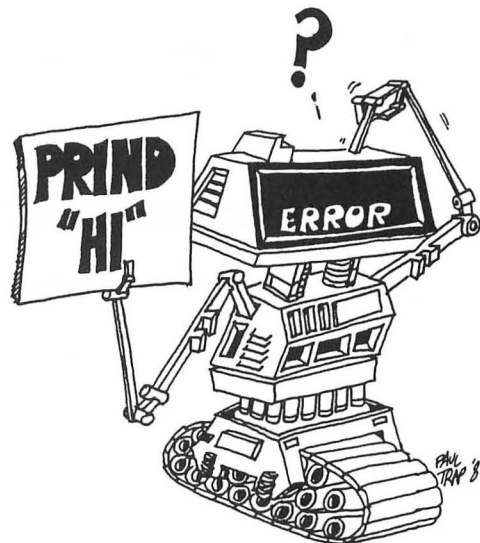
The computer says:

`ERROR- GIVE ME CANDY`

The computer only understands about 80 words. You need to learn which words the computer understands.

Here are the first 4 words to learn:

`NEW, PRINT, REM, and RUN.`

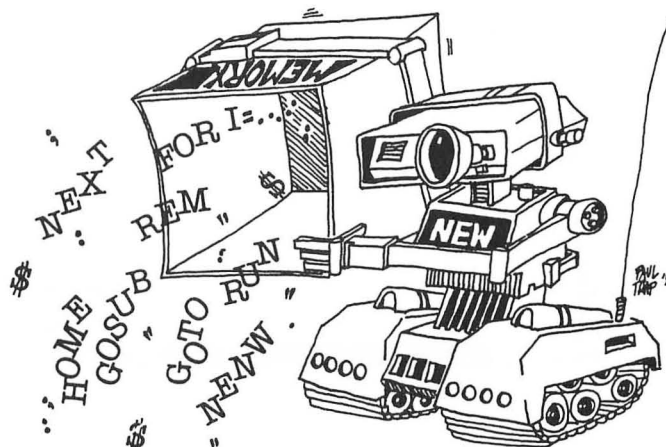


THE NEW COMMAND

Type: `NEW`

and press RETURN.

`NEW` empties the computer's memory so you can put your program in it. It doesn't erase anything from the TV screen.



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.

Enter this line: `10 PRINT "HI"`

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

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

Now line number 10 is in the computer's memory. It will stay in memory until you enter the NEW command, or until you turn off the computer. Line 10 is a very short program.

THE NUMBER ZERO AND THE LETTER "O"

The computer always writes the zero like this:

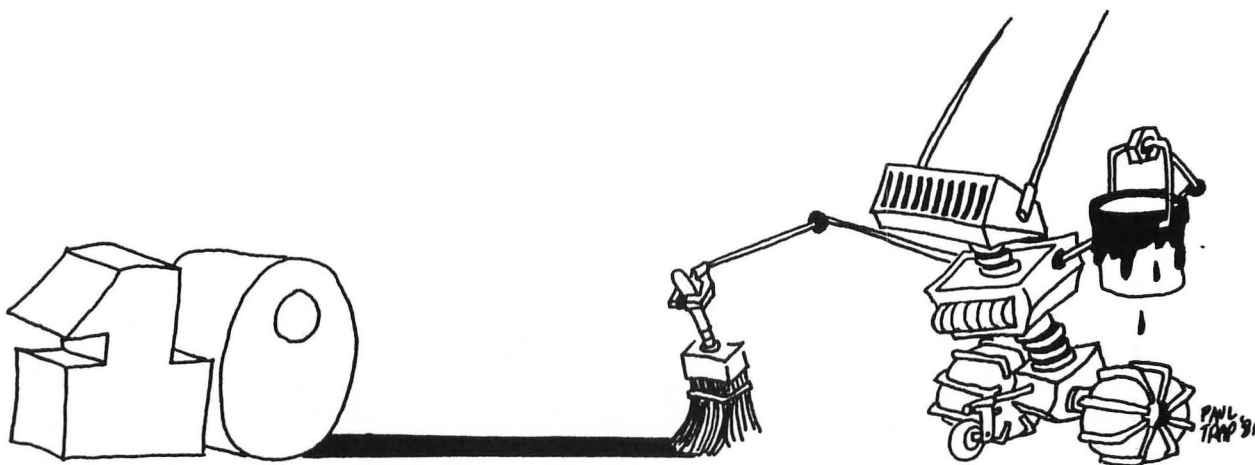
zero Ø

and the letter O like this:

letter O O

You have to be careful to do the same.

right 10 PRINT "HI"
wrong 10 PRINT "HI"



WHAT IS A PROGRAM?

A program is a list of commands you wish the computer to do. The commands 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:

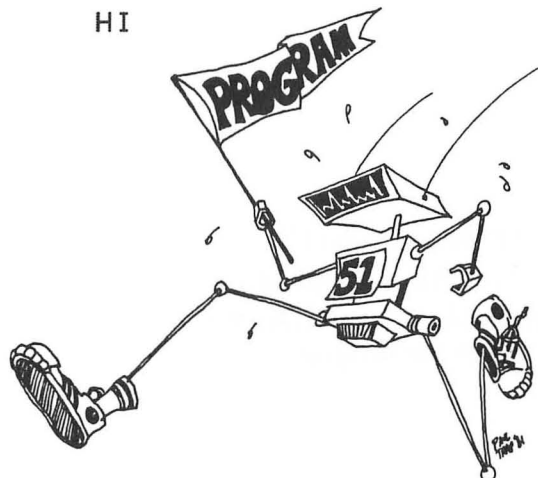
```
RUN
```

(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 commands it reads in the lines.

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

HI



A LONGER PROGRAM

Clear the memory with NEW

(Did you remember to press RETURN afterward?).

Enter this program:

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

This program has 3 lines. Each line starts with a command.

Enter: RUN

Line 1—the computer skips this line because it is a REM. Line 2—the computer prints “HI.” Line 3—then computer prints “FRIEND.” The REM command lets you put little notes to yourself in the program. REM means “remark” or “reminder.”

In line 1 we used REM to give a name to the program. The name is “PROGRAM 1” The computer does the commands in the lines. It starts with the lowest line number and goes down the list in order.

HOW TO NUMBER THE LINES IN A PROGRAM

Usually you will skip numbers when writing the program.

Like this:

```
10 REM PROGRAM 1
20 PRINT "HI"
30 PRINT "FRIEND"
```

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

Assignment 1:

1. Show how to “clear the screen”.
2. Use the command NEW. Explain what it does.
3. Write a program that uses REM once and PRINT twice. Then use the RUN command to make the program obey the commands.

INSTRUCTOR NOTES 2 BUZZ, INVERSE, AND STRING CONSTANTS

This lesson opens with the CTRL 2 key sequence which makes the buzzer sound. We wish to make plenty of "bells and whistles" available to the student to increase program richness.

The idea of a "string constant," used in Lesson 1, is explained. The numbers appearing in a string, for example the "19," cannot be used directly in arithmetic.

The INVERSE command puts a little pizzazz on the screen.

Although the ATARI can print in lower case letters, we will not do so in this book. There are several reasons: we need lower case to indicate special commands like "buzz, inverse, normal, clear" in PRINT commands. The lower case letters are so small that they are not very clear anyway. And it just adds an unnecessary complication. If the instructor wishes to include lower case typing, do so by giving extra explanations, especially emphasizing that "clear" in a PRINT line does not mean type the word "clear," etc.

QUESTIONS:

1. How do you do each of these things: Make the ATARI "buzz"? Erase the screen? Empty the memory?
2. What is a "string"?
3. What special key do you press to "enter" a line?
4. What is a command? Give some examples.
5. How could you print "FIRE" in inverse letters and make the computer buzz?



LESSON 2 BUZZ, INVERSE, AND STRING CONSTANTS

Enter: NEW (remember: RETURN key)

Then clear the screen. (remember: SHIFT CLEAR keys)

You are ready to start this lesson.

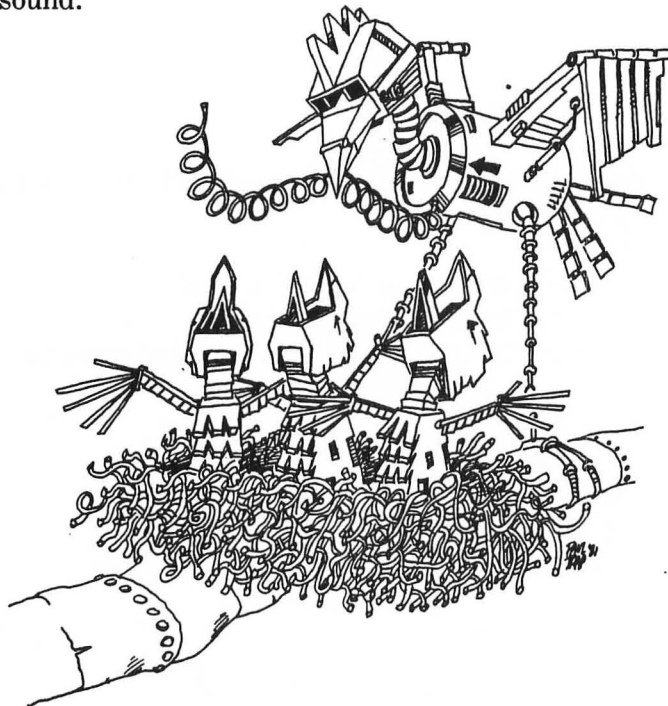
SOUNDING THE BUZZER.

The ATARI has a buzzer. Use two keys to make it sound.

Hold down the CTRL key. Then press the "2" key.

You hear a loud buzzing sound.

CTRL stands for "control." This key helps control things on the computer. CTRL 2 makes the buzzer sound.



PRINTING AN EMPTY LINE

Run this:

```
10 REM LINES
20 PRINT "HERE IS A LINE"
30 PRINT
40 PRINT "ONE LINE WAS SKIPPED"
```

Line 30 just prints a blank line.

STRING CONSTANTS

Look at these PRINT statements:

```
10 PRINT "JOE"  
10 PRINT "#D47*%"  
10 PRINT "19"  
10 PRINT "3.1416"  
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 " "
```

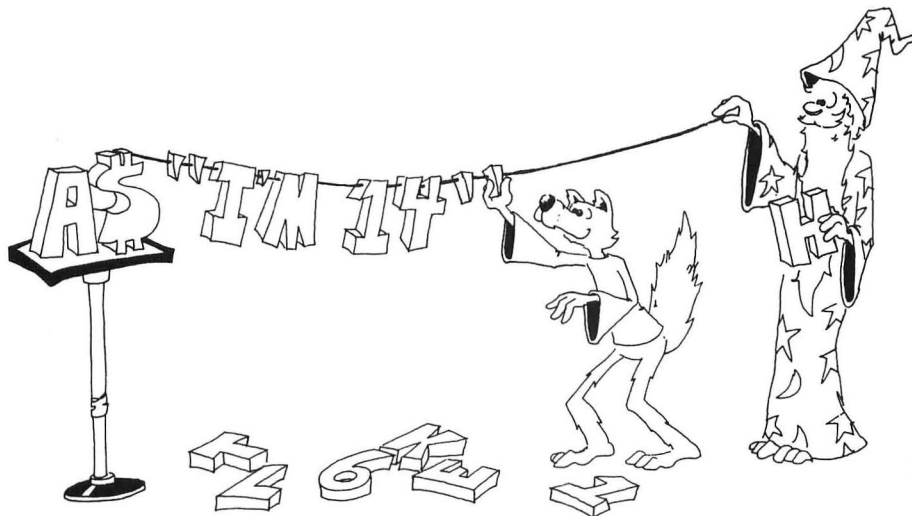
(Later we will have graphics characters too, and some special characters like "buzz" and "clear.") Characters in a row make a "string."

The letters are stretched out like beads on a string.

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

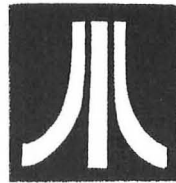
It is a string because it is made of letters, numbers and punctuation marks in a row.

It is a constant because it stays the same. It doesn't change as the program runs.



INSIDE OUT PRINTING

There is a special key on the ATARI computer. It looks like this:



It is called "the ATARI key."

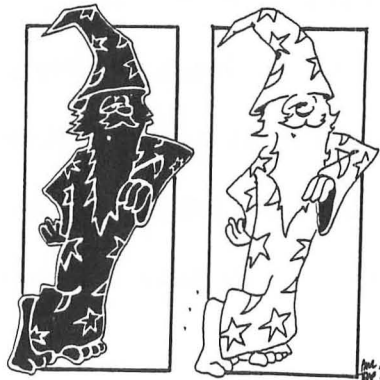
Press it. Type.

Now all the letters you type are in the "inverse" mode.

Press the ATARI key again. Type.

You are back to normal (non-inverse) letters.

Use inverse to make the stuff printed on the screen look more interesting.



INVERSE AND NORMAL IN PRINT COMMANDS

In this book we have a special way of telling you to press the ATARI key.

We will say "inverse" or "normal." It looks like this:

```
10 REM BLACK AND WHITE
20 PRINT" inverse WHITE normal
BLACK "
```

This means:

Type:	20 PRINT"
Press the ATARI key	
Type:	WHITE
Press the ATARI key again	
Type:	BLACK "
Press the RETURN key.	

Run the program.

CAPITAL LETTERS AND SMALL LETTERS

The ATARI can print in small letters (the ones called "lower case").

But in this book, we will not use any lower case on the computer.

IMPORTANT! Whenever you see lower case writing inside the quotation marks of a PRINT command, it will have a special meaning.

```
20 PRINT inverse      is not 20 PRINT "INVERSE"  
20 PRINT normal       is not 20 PRINT "NORMAL"
```

We saw one special meaning above with the ATARI key. Here is another.

THE ESC KEY

Find the ESC key. It is in the upper left corner of the keyboard. ESC is short for ESCAPE.

This key helps us give special instructions like "clear" and "buzz" to the computer.

Whenever you see "clear" in a PRINT command, do this:

Press the ESC key once.

Then hold down the SHIFT key and press the CLEAR key. Got it?

```
Enter:          10 REM SPECIAL PRINT  
                20 PRINT "clear"  
                30 PRINT "CLEAN SLATE"
```

Of course you do not see "clear" in line 20. You see a funny bent arrow instead.

Whenever you see "buzz" in a PRINT command, do this:

Press the ESC key once.

Then hold down the CTRL key and press the 2 key.

```
Add this:      40 PRINT "buzz"
```

Of course, you do not see "buzz" in line 40. You see a funny inverse bent arrow instead.

Run the program. It should clear the screen, print CLEAN SLATE, and sound the buzzer.

Assignment 2:

1. Write a program that prints your first, middle and last names. Make the first and middle names in normal letters, and the last name in inverse letters.
2. Now make the program clear the screen before writing anything.
3. Now make the buzzer sound before printing each name.



INSTRUCTOR NOTES 3 LIST AND MEMORY BOXES

In this lesson:

- LIST, LIST 30
- REM for titles, remarks
- memory boxes holding lines
- erase one line from memory
- add a line between old lines
- replace a line
- drawings using PRINT commands

The difference between “command” and “statement” is artificial. The BASIC interpreter does not distinguish between them. Our wishes are called “commands” when used in the edit mode and “statements” when used in a program line. PRINT is a good example of a command used both ways. In the first part of this book I will call all of these things “commands” and later on explain what is meant by a statement (when talking about colons and having several statements on one line).

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 use LIST 100,300 will be taken up later.

The memory as a shelf of boxes is a key model of the computer 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.

REM as a remark command can be a little confusing to new students. It needs to be distinguished both from PRINT and from just typing to the screen.

Using print to draw pictures is demonstrated. It is better to draw some at the end of each lesson than to do a lot now. Drawing after lesson 4 helps develop line editing skills.

QUESTIONS:

1. How do you erase a line you no longer want?
2. Clear the screen. Now how do you show the program in memory on the screen?
3. How can you replace a wrong line with a corrected one?
4. Suppose you want to put a line in between two lines you already have in memory. How do you do this?
5. Explain how the computer puts program lines in “boxes” in memory. What does it write on the front of the box?

LESSON 3 LIST AND MEMORY BOXES

Start each lesson by clearing the screen and the memory.

Now enter:

```
10 REM HEARING  
20 PRINT "clear"  
30 PRINT "LISTEN"  
40 PRINT "buzz"  
50 PRINT "DID YOU HEAR IT?"
```

Run this 5 line program. Then clear the screen.

The program is no longer visible on the screen.

But the program is not lost. The computer has stored the program in its memory. We can ask the computer to show us the program again.

LISTING THE PROGRAM

To ask the computer to show line 30 of the program, enter:

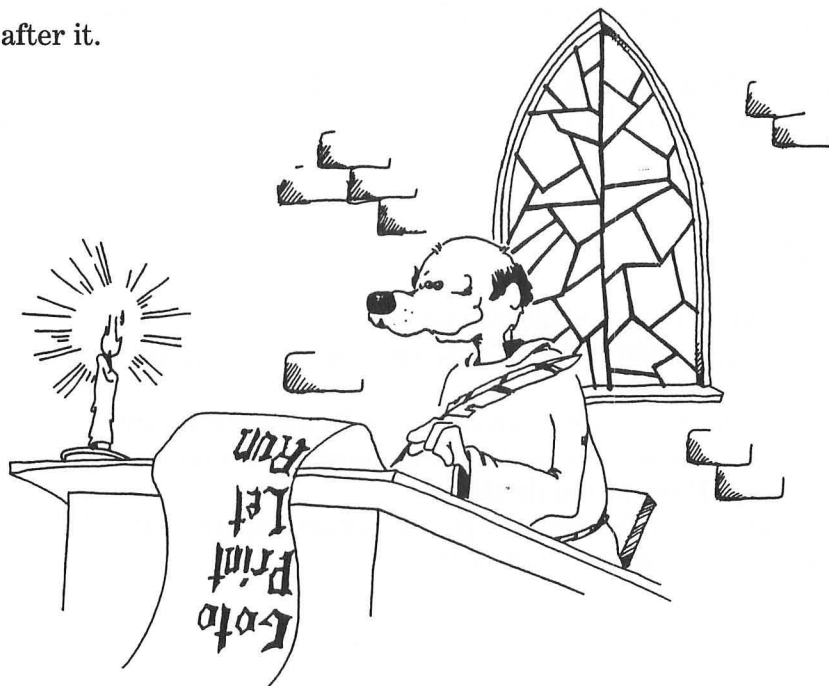
```
LIST 30
```

The computer will list whatever line you ask for by number. If you want to list the whole stored program just enter

```
LIST
```

with no number after it.

Try it.



THE MEMORY

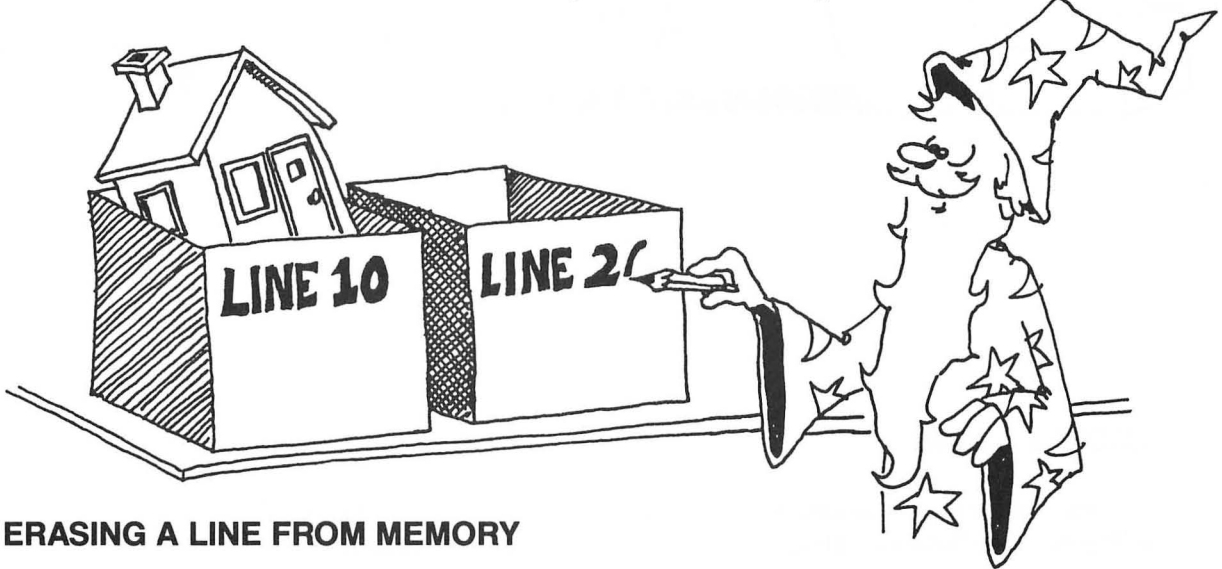
The computer's memory is like a shelf of boxes. The name of the box goes on the front of each box. At the start, all the boxes are empty and no box has a name.

When you entered: `10 REM HEARING`

the computer took the first empty box and wrote the name "Line 10" on the label. Then it put the command `REM HEARING` into the box and put the box back on the shelf.

When you entered: `20 PRINT "clear"`

the computer took the second box and wrote "Line 20" on its label. Then it put the command `PRINT "clear"` into 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. For example,

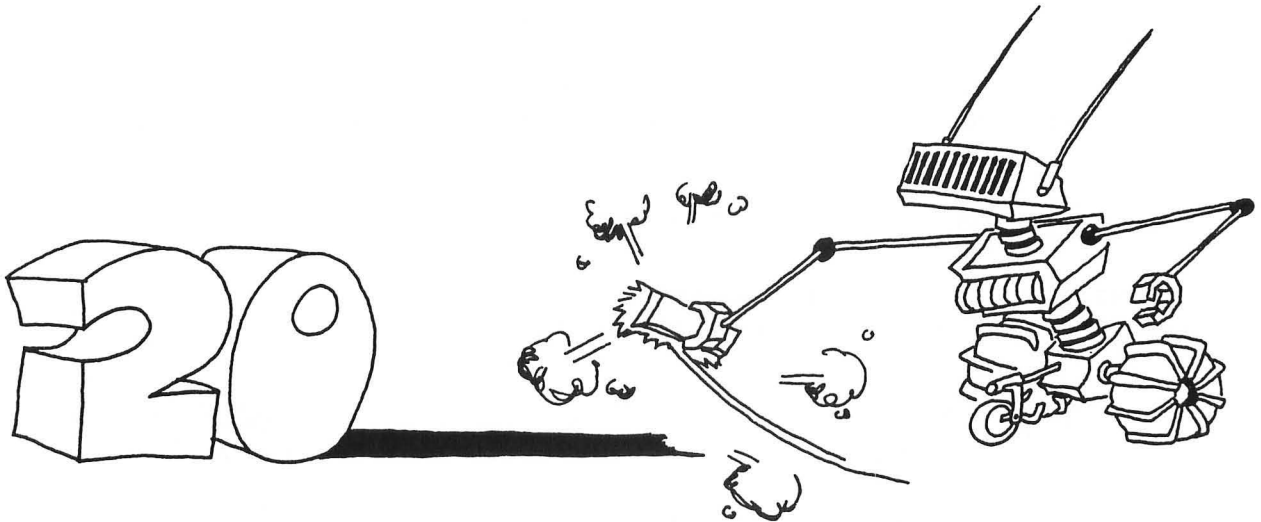
to erase line 20, enter: `20`

You still see the line on the screen, but do a `LIST` and you 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 off the front of the box.

How do you erase the whole program? (Look at lesson 1 to see the answer.)

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 those of the old lines, and type your line in. The computer puts it in the correct place.

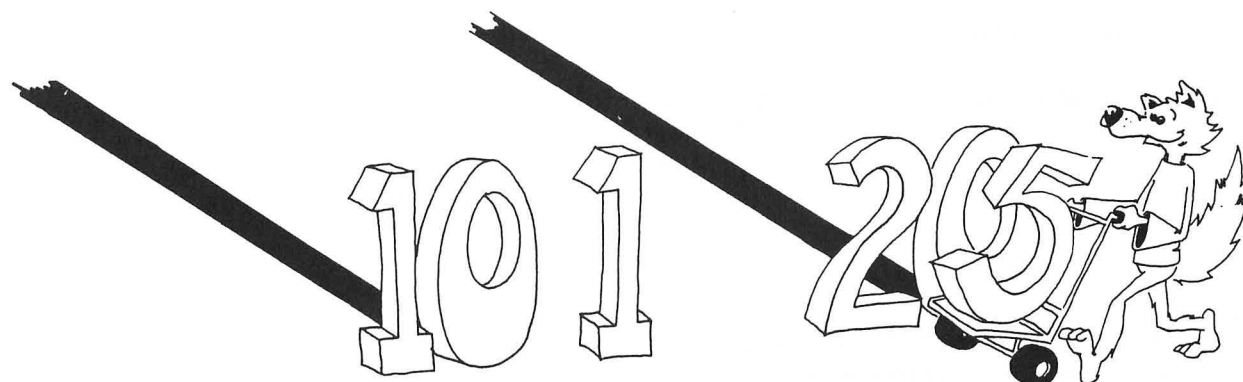
Enter NEW and this:

```
10 REM MORE AND MORE
20 PRINT "MORE LINES WANTED"
40 PRINT "HERE THEY ARE"
```

List it and run it. Now add this line:

```
15 PRINT "STILL"
```

List and run it again.

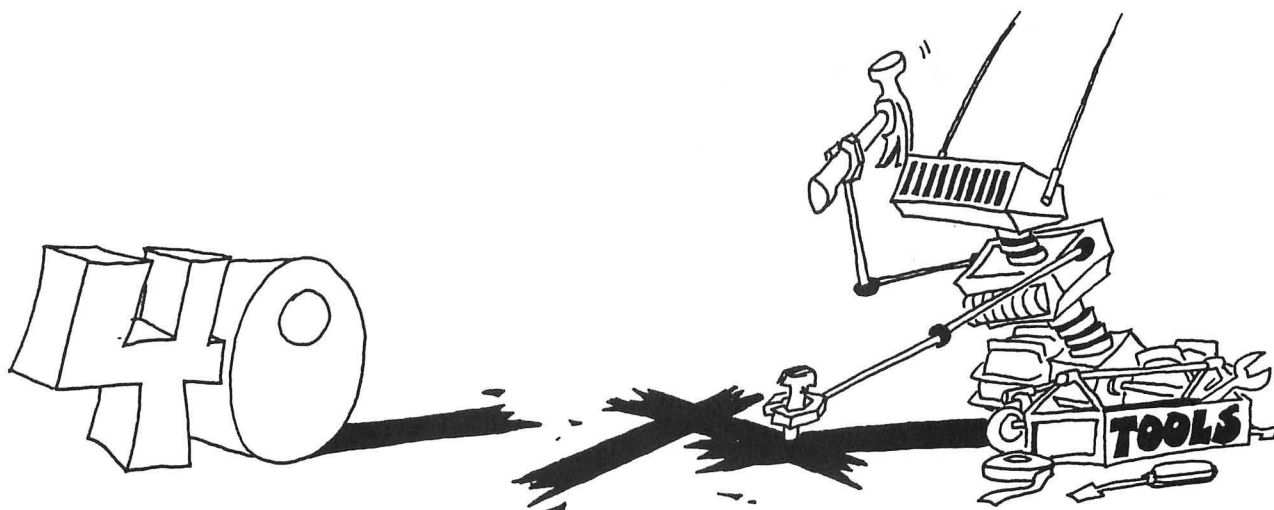


FIXING A LINE

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

```
40 PRINT "NEEDS FIXING"
```

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



THE REM COMMAND

Use a REM command to put remarks in your program.

Enter NEW and this:

```
10 REM PROGRAM 2
20 PRINT "clear"
30 PRINT "LINE 10 DOES NOTHING"
35 REM THIS LINE DOES NOTHING
40 LIST
RUN
```

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



PICTURE DRAWING

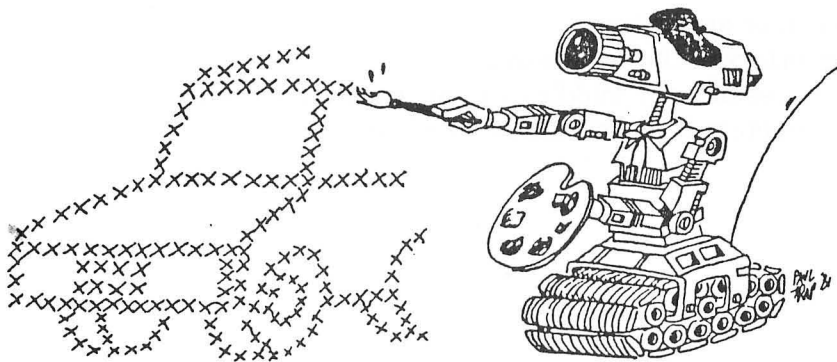
You can use the PRINT command to draw pictures. Here is a picture of a car. Enter NEW before drawing the car.

```
10 PRINT"clear"  
20 PRINT  
30 PRINT" XXXXXX"  
40 PRINT"XXXXXXXXXXXXX"  
50 PRINT"  "      " "
```

Don't forget to put the spaces in the PRINT lines! They are part of the drawing.

Assignment 3:

1. What command will list line 10 of the 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 command?
4. What is the REM command used for?
5. Use "clear," "buzz," REM, and PRINT to draw 3 flying birds on the screen. Make each bird squawk after it is drawn.



INSTRUCTOR NOTES 4 BACKSPACE, CURSOR KEYS, INSERT, DELETE

This lesson shows how to use the cursor movement keys and the backspace, insert and delete keys.

The BACK S key is a “rubout” key. Pressing it erases the character to the left of the cursor. The cursor moves onto the empty space. Hold down the BACK S key and after a half second pause, it starts repeating, erasing as it goes.

In fact, most keys on the keyboard repeat if you hold them down.

The arrow keys are used in moving the cursor around on the screen. The CTRL key must be held down while using the arrow keys.

Wherever the cursor stops, you can type in new characters.

The INSERT and DELETE keys also require holding down the CTRL key. The DELETE key deletes the character the cursor is on. The INSERT key inserts a space to the left of the cursor, then the cursor moves onto the space.

QUESTIONS:

1. What is a “cursor”? What is it good for?
2. Have your student demonstrate how to edit a line:
 - Using the BACK S key to rubout recent errors,
 - Using the cursor arrow keys,
 - Correcting letters in the middle of a line,
 - Inserting spaces into the middle of a line, then typing into them,
 - Deleting characters from the middle of a line.

LESSON 4 BACKSPACE, CURSOR KEYS, INSERT, DELETE

BACKSPACE

Type a long line. Then press the BACK S key. It erases one letter and moves the cursor back one space.

Now hold the BACK S key down. It whizzes along, erasing!

(BACK S stands for "backspace.")

THE ARROW KEYS

There are 4 arrow keys. They are in a little square at the right of the keyboard.

On top, one has an up arrow. The other has a down arrow.

On the bottom, one has a left arrow, the other a right.

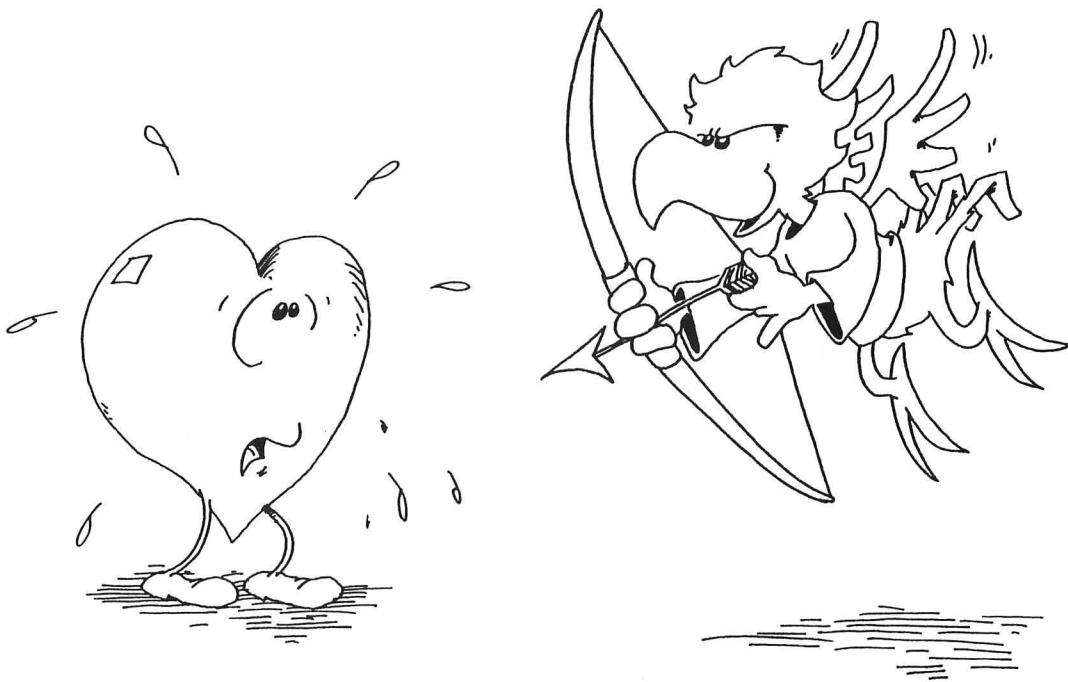
Hold down the CTRL key.

Press any of the cursor keys.

The cursor goes in the direction of the arrow.

Then press another cursor key.

When you get the cursor where you want it, type something.

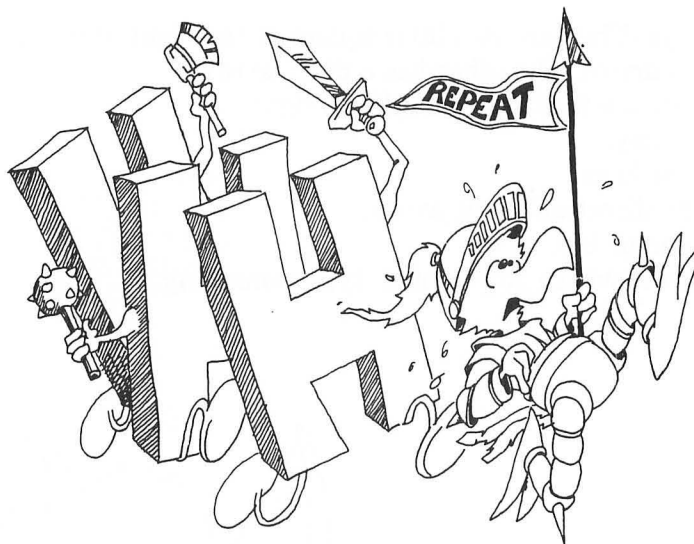


WHIZZING ALONG

Hold down the CTRL key. Now hold down an arrow key. The cursor goes whizzing along!

This works with other keys too. Hold down the "A" key. After a pause, you get a row:

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA



FIXING MISTAKES

Enter: `10 REM ATERI`

Move the cursor key over the letter E.

Type an "A" instead.

Now the line is correct, reading:

`10 REM ATARI`

Press RETURN to store the line in the memory.

INSERT KEY

Type this: `20 REM DRAON`

We want a G in the word DRAGON. Move the cursor over the O.

Hold down the CTRL key and press the INSERT key. The line opens up and a space is put to the left of the cursor. The cursor moves onto the new space. Type a G.

Now we have `20 REM DRAGON`

Of course, if you hold down the CTRL and INSERT keys, a whole row of spaces is inserted.

DELETE KEY

This key is the opposite of the INSERT key.

Move the cursor back over the G in DRAGON.

Hold down the CTRL key and press the DELETE key. The G disappears and the line closes up from the right. It again reads:

`20 REM DRAON`

HEY LOOK! IT IS REALLY EASY.

You use the CTRL key for all kinds of fixing.

Hold CTRL down. Then:

Press the cursor arrow keys to get where you want.

Press the DELETE key if you want to erase.

Press the INSERT key to add room for letters.

Then let up on the CTRL key and type what you want.

Assignment 4:

1. Type a line and use the arrow keys to move around in the line. Change letters in the line. When you are done, press RETURN to enter the line. Use LIST to look at the line.
2. Type a line, use the arrow keys to get to the middle of it. Use the DELETE key to remove letters. Press RETURN. LIST.
3. Type a line and use the arrow keys to get to the middle. Use the INSERT key to add spaces. Type something in the spaces. Press RETURN. LIST.
4. Draw a "smiley face."



INSTRUCTOR NOTES 5 STRING BOXES, DIM, LET

The concept of memory boxes is introduced. DIM is explained and LET is used to fill the box with a string constant.

The box model is used to emphasize that LET is a replacement command, 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, and new contents inserted.

More exactly, when a variable is used, a copy of the contents is made and used. The original contents remain intact. This point is explained.

Used so far:

```
PRINT, NEW, REM, RUN, LIST, DIM, LET
```

Special keys discussed so far:

SHIFT, RETURN, CTRL, ESC, BACK S, DELETE, INSERT, CLEAR, four arrow keys.

QUESTIONS:

1. Why do you have to use the DIM command?
2. If you run this little program:

```
10 DIM A$(25),B$(25)
20 LET A$="HI"
30 LET B$=A$
```

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

3. In this program:

```
10 DIM Q$(5)
20 Q$="MOM"
```

What is "MOM" called?

What is the name of the string variable in this program?

What is the value of the string variable after line 10 runs?

What is the value of the string variable after line 20 runs?

4. What is wrong with this program?

```
10 DIM H$(5)
20 LET H$="FAT SAUSAGE"
```


LESSON 5 STRING BOXES, DIM, LET

STRING CONSTANTS

You know what a string constant is. For example:

"MOPSEY"

You can use the string constant in a PRINT statement:

Enter: `20 PRINT "MOPSEY"`

BOXES IN MEMORY

We want to have a box in memory for holding any string we choose.

First we must tell the computer how big a box we need. We use the DIM command. Add these lines:

```
10 REM STRING BOX
15 DIM W$(10)
```

Line 15 says:

Get a box big enough to hold 10 letters.
Write the name W\$ on the front.
Leave the box empty for now.

DIM stands for "dimension" or "size." The DIM command gives the name of the box and how large it is.

STRING VARIABLES

The DIM command made a string variable.

Its name is W\$.

Rule: String variable names always end in a dollar sign, "\$." Pick any letter you like for the name and then put a dollar sign after it.

W\$ is called a variable because you can put different strings in the box at different times. The box can hold only one string at a time.

The memory box can be any size from one letter up to thousands of letters (if your memory is not yet full).

FILLING THE BOX

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

```
10 REM STRING BOX
12 PRINT "clear"
15 DIM W$(10)
20 LET W$="MOPSEY"
40 PRINT W$
```

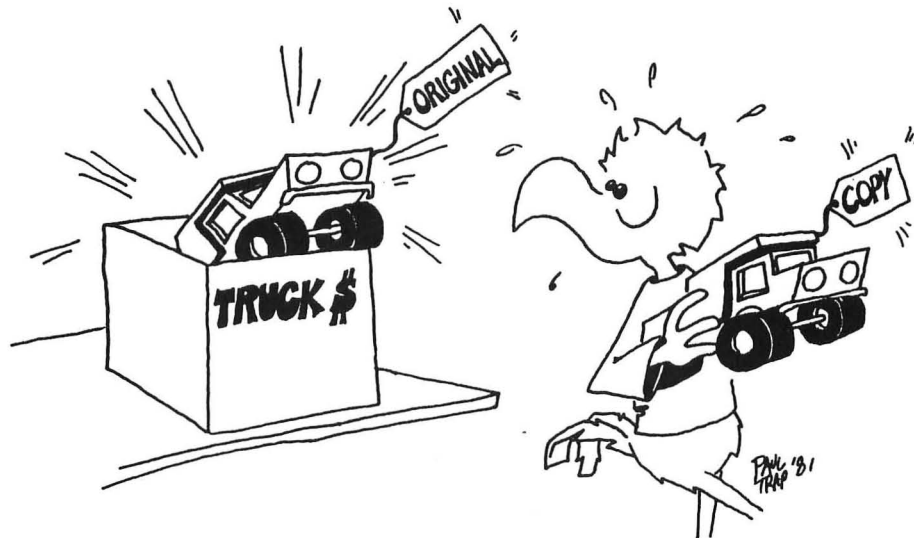
Here is what the computer does:

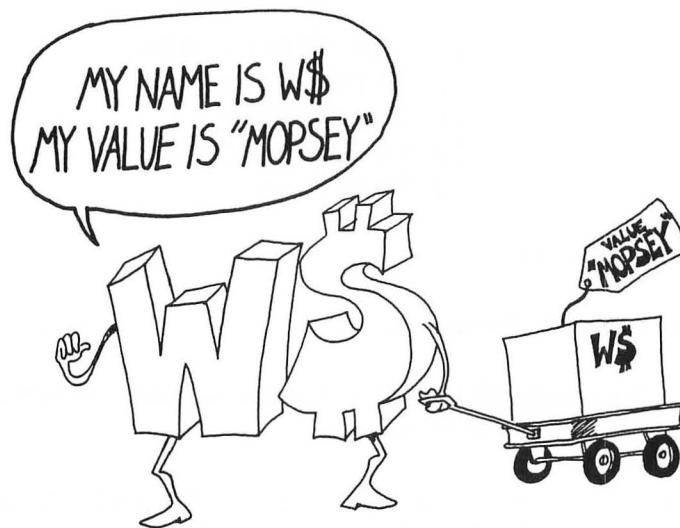
Line 12 The computer clears the screen.

Line 15 It picks a box big enough to hold 10 letters and puts the name W\$ on the front. The box is empty for now.

Line 20 The computer takes the string "MOPSEY" and puts it into the box W\$. (The box holds 10 letters, MOPSEY fits because it has only 6 letters.)

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 "MOPSEY" that it finds there. It puts the copy on the TV screen. The string "MOPSEY" is still in box "W\$."





NAMES AND VALUES

The name of the variable was W\$. It is put on the front of the box.

The value of the variable was "MOPSEY." The value is put into the box.

MANY BOXES

The DIM command can create more than one box at a time. Look at this:

```
10 DIM A$(8), Z$(40), D$(10)
```

Line 10 made three string variables. Their names are A\$, Z\$, and D\$.

RULES for boxes:

1. You must make a box with DIM before you put something into it.
2. Each box has a different name. Make each box only once.
3. You can have as many DIM commands as you want, but each makes boxes with different names.

ANOTHER EXAMPLE:

Enter and run:

```
10 DIM A$(40)
12 DIM Z$(40), D$(40)
15 LET D$="PICKLES"
20 LET A$=" AND "
30 PRINT "WHAT GOES WITH PICKLES?"
35 INPUT Z$
40 PRINT "clear"
50 PRINT D$;A$;Z$
```

Explain what the computer does in each line.

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



Assignment: 5

1. Write your own program that uses the DIM and the LET commands and explain how it stores things in "boxes."
2. Write a program that makes a box called N\$. Then use LET to put your first name into the box. Take your first name out of the box and PRINT it. Finally, put your friend's name into the same box. Take it out and print it.

INSTRUCTOR NOTES 6 THE INPUT COMMAND

This lesson reviews the idea of a string variable and memory boxes and the DIM and LET commands. It introduces the INPUT command and the SETCOLOR command for GRAPHICS 0 screens.

We will give only the central features of each command for the whole of the introduction of the book (through lesson 14). This allows us to quickly outline the essentials of programming so that the student “sees the forest” and is able to write meaningful programs. The commands essential for interesting programs are:

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

Back to this lesson. String variables are introduced using the “box” concept again. Variable names are restricted to one letter for the time being. This does not lead to confusion in short programs and allows faster typing.

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

The SETCOLOR command is introduced using the idea of “paint pots.” The student is shown how to make colored screens and borders in the GRAPHICS 0 screen mode.

QUESTIONS:

1. What two different things does the computer put into boxes?
2. How does the program ask a person to type in something?
3. How do you know the computer is waiting for an answer?
4. What is a letter with a “\$” after it called?
5. Write a short program that uses DIM, LET, PRINT and INPUT.
6. How do you change the color of the screen to green?

LESSON 6 THE INPUT COMMAND

MAKING THE BOXES

We make a box before putting a string into it.

Enter:

```
10 REM STRING BOXES
15 DIM B$(30), C$(40)
```

Rule: You must do a DIM for each variable before you use the variable.

If you forget, the computer will print:

ERROR - 9 when you run the program.

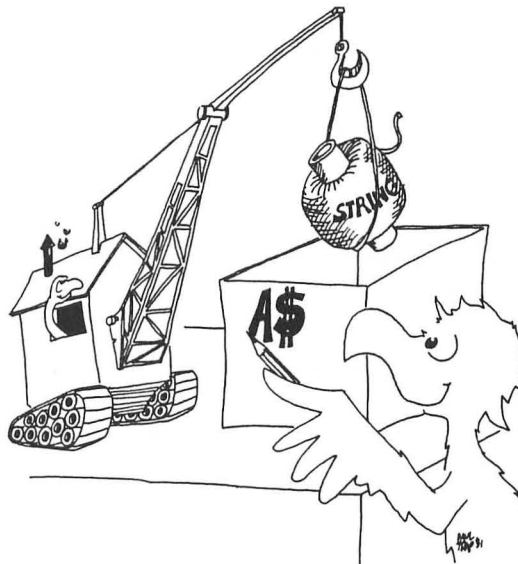
FILLING THE BOXES

The boxes B\$ and C\$ are still empty.

There are two ways to put something into a box:

The first way uses the LET command.

The second way uses the INPUT command.



THE LET COMMAND

Add this line:

```
20 LET B$="SAY SOMETHING"
```

Line 20 takes the string constant "SAY SOMETHING" and puts it into the variable B\$.

B\$ is the "name" of the string variable.

"SAY SOMETHING" is the "value" of the string variable.

The value has 13 characters. (One space, and 12 letters.)

"SAY SOMETHING" easily fits because the box is 30 characters long. (It was made that big by DIM in line 15.)

THE INPUT COMMAND

The INPUT comand makes the computer ask you for something.

The INPUT command is the other way to put something into a string box.

You have:

```
10 REM STRING BOXES
15 DIM B$(30), C$(40)
20 LET B$="SAY SOMETHING"
```

Enter:

```
22 PRINT B$
25 INPUT C$
30 PRINT
35 PRINT "DID YOU SAY "
40 PRINT "' ' ; C$ ; "' ?"
```

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 cursor means the computer expects you to type something in.

When you type "HI," the computer stores this word in the box named C\$.

Run the program again and this time say something funny.



PAINT POTS WITH 16 COLORS

The ATARI has 5 paint pots numbered 0, 1, 2, 3, and 4.

We need to use pots 2 and 4.

ATARI numbers its colors from 0 (grey) to 15 (orange)

Try this:

```
10 REM COLORED SCREEN
20 PRINT "PAINT POT 2 IS THE SCREEN COLOR"
30 PRINT "WHAT COLOR SCREEN DO YOU WANT"
33 PRINT "NUMBERS FROM 0 TO 15"
36 INPUT C
40 PRINT "HOW BRIGHT <0 TO 14>"
45 INPUT B
50 SETCOLOR 2, C, B
60 GOTO 30
```

Line 50 tells what color and brightness to make paint pot 2.

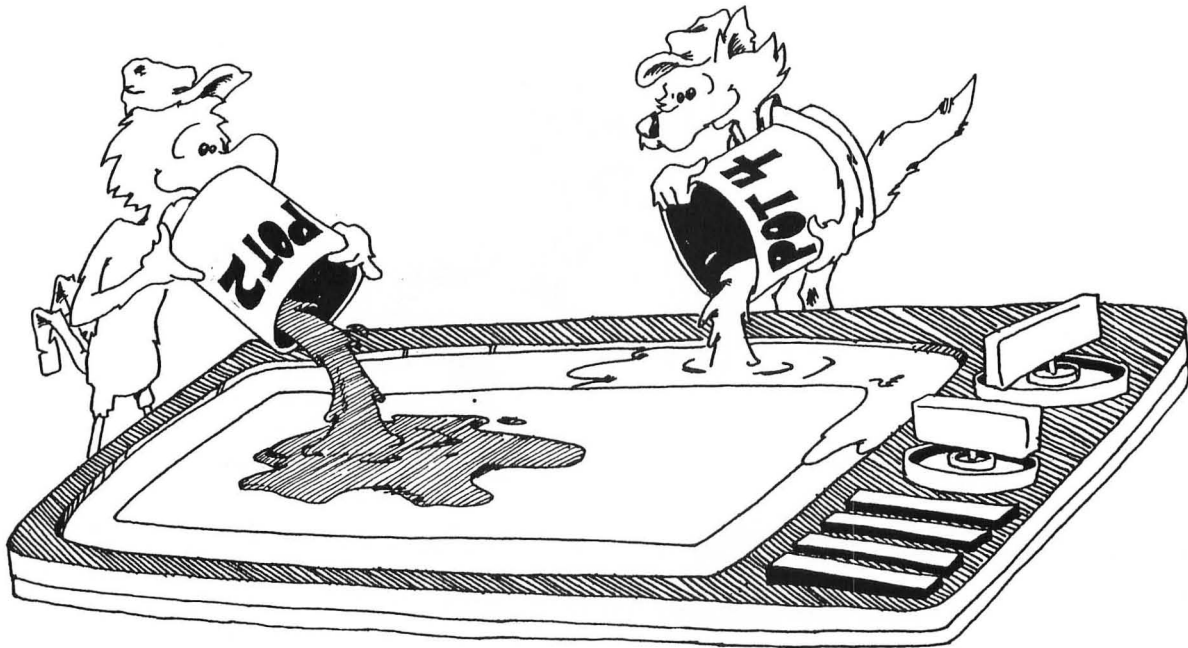
It is like choosing a clean, empty, paint pot and pouring a color (by number) into it so you can paint the screen.

Run the program and try all the colors from 0 to 15. For each, try even numbers from 0 to 14 for the brightness.

Black is color 0 with brightness 0.

(This program uses number variables B and C. We will learn more about numerical variables in lesson 12.)

Line 60 has the GOTO command. The next lesson explains this command.



COLORED BORDERS

Make these changes in the program above:

```

10 REM COLORED BORDER
20 PRINT "PAINT POT 4 IS THE BORDER COLOR"
30 PRINT "WHAT COLOR BORDER DO YOU WANT?"
33 same
36 same
40 same
45 same
50 SETCOLOR 4, C, B
60 same

```

Assignment 6:

1. Write a program that asks for a person's name. Then have it change the screen and border colors and print something silly to the person, by name.
2. Write a program that asks you to INPUT your favorite food and put it into a box called F\$. Now the program asks you your favorite animal and puts this into box F\$ too.
Have the program print F\$. What will be printed? Run the program and see if you are right.

INSTRUCTOR NOTES 7 TRICKS WITH PRINT

In this lesson:

- PRINT with a semicolon at the end
- PRINT with semicolons between items
- the "invisible" PRINT cursor
- PRINT with commas between items
- GRAPHICS 1,2 screens, an introduction

The lesson introduces the output cursor which is invisible on the screen. It marks the place where the next character will be placed on the screen by a PRINT command. (The input cursor is the square you can see on the screen. It is familiar from the edit mode and the INPUT command.)

When a PRINT statement ends with a semicolon, the output cursor remains in place. The next PRINT will put its first character exactly into the spot following the last character printed by the current PRINT command.

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

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

The series of printed items will have their characters in contact. If spaces are desired, as in the "HAM AND EGGS" example, the spaces have to be put into the strings explicitly.

The 11 graphics modes of the ATARI require some awkward sequences of commands for best use. We introduce these modes slowly.

QUESTIONS:

1. Which cursor is a little flashing square? What command puts it on the screen?
2. Which cursor is invisible? What command 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?

5. What do commas do in print lines?

LESSON 7 TRICKS WITH PRINT

ONE LINE OR MANY?

Enter this program:

```
10 REM FOOD
20 PRINT
30 PRINT "HAM"
40 PRINT "AND"
50 PRINT "EGGS"
```

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

Now enter:

```
30 PRINT "HAM ";
40 PRINT "AND ";
```

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

Run it.

What was different from the first time?

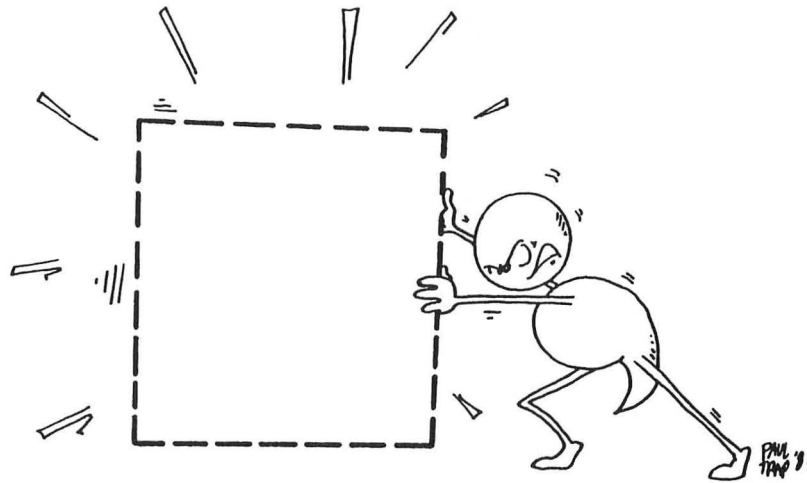
THE HIDDEN CURSOR

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

The PRINT command 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 command adds on to what has already been written on the same line.



FAMOUS PAIRS

Enter:

```
10 PRINT "clear"
15 DIM A$(30),B$(30)
20 PRINT "ENTER A NAME"
30 INPUT A$
35 PRINT "clear"
40 PRINT "ENTER ANOTHER"
50 INPUT B$
60 PRINT "clear"
70 PRINT"PRESENTING THAT FAMOUS TWOSOME:"
75 PRINT
80 PRINT A$;" AND ";"B$
```

Be sure to put a space before and after the "AND".

SQUASHED TOGETHER OR SPREAD OUT?

Enter NEW then try this:

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

after you have run it, try also:

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

Rule: When you put a comma between two items in a PRINT list, the computer spreads them out in columns.

BIG LETTERS

The ATARI has 11 different ways to use the TV screen. Here are the first three of them.

GRAPHICS 0	what you have been using
GRAPHICS 1	wide colored letters
GRAPHICS 2	large colored letters

Run:

```
10 GRAPHICS 1
20 PRINT #6; "HI THERE"
30 PRINT "clear FRIEND"
```

Press the SYSTEM RESET key to get the screen back to normal.

Use the #6 in PRINT when writing the big letters.

Don't use #6 when writing to the little screen below.

Try it again with a "2" in place of the "1."

SPLIT SCREEN

Both GRAPHICS 1 and GRAPHICS 2 have a "split screen" below the large letter screen.

If you want all large letters, do this

```
10 GRAPHICS 1+16
20 PRINT #6; "NO SPLIT"
30 GOTO 30
```

Try it again with GRAPHICS 2+16

Line 30 is an "infinite loop." It keeps the program from ending. When the program ends, the screen goes back to GRAPHICS 0 automatically.

We will study the GOTO command in lesson 8.

Assignment 7:

1. Write a program that asks for the name of a musical group and one of their tunes. Then using just one PRINT command, print the group name and the tune name, with the word "plays" in between.
2. Do the same, but use 3 print commands to print on one line.
3. Now have it printed out in large letters (GRAPHICS 2).

INSTRUCTOR NOTES 8 THE GOTO COMMAND AND BREAK KEY

The GOTO command allows a “dumb” loop that goes on forever. It also helps in flow of command in later programs, after the IF is introduced. It provides a slow and easy entrance for the student into the idea that the flow of command need not just 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 BREAK key does this nicely.

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

QUESTIONS:

1. In this little program:

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

what will appear on the screen when it is run?

2. And this one:

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

3. How do you stop the program in question 2?
4. Write a short program that buzzes, asks you your favorite movie star's name, and then does it over and over again.

LESSON 8 THE GOTO COMMAND AND BREAK KEY

JUMPING AROUND IN YOUR PROGRAM

Try this program:

```
10 REM WHIZZING
12 DIM N$(40)
15 PRINT "clear"
20 PRINT "YOUR NAME?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

RUN this program. It never stops by itself! To stop your name from whizzing past your eyes, press the BREAK key.

Line 40 uses the GOTO command. 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 in a lot of programs.



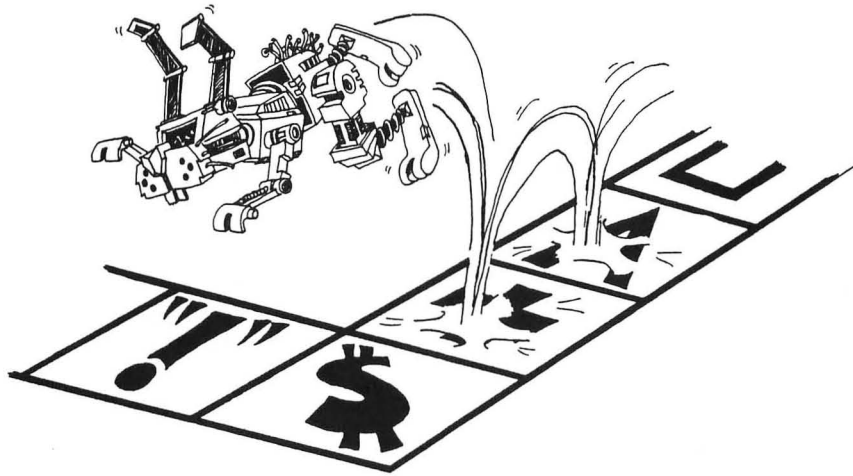
MORE JUMPING

Enter:

```
10 REM NON-STOP TALKER
15 DIM S$(40)
20 PRINT "SAY SOMETHING"
30 INPUT S$
40 PRINT "DID YOU SAY '" ; S$ ; "'?"
45 PRINT
50 GOTO 30
```

Run the program. Type a different answer every time you see the "?" and the cursor. Press the BREAK key to end the program.

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



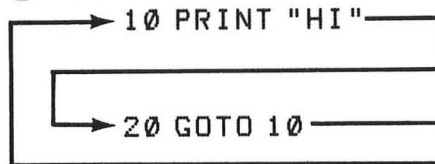
KINDS OF JUMPS

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

Jumping back gives a LOOP.

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

The path through the program is like this:



The computer goes around and around in this loop. Press the BREAK key to stop.

Jumping ahead lets you skip part of the program. It is not useful yet, but we will use it later in the IF command.

THE BREAK KEY

The BREAK key is a "life saver." When you are in trouble, press BREAK and the computer will "peep" and start over, waiting for your next command. Your program is still safe in memory.

If the computer still is messed up, for example if it "hangs," press the SYSTEM RESET key.

(When the computer just sits there like a dolt, and doesn't do anything no matter what key you press, we say that the computer is "hanging.")



Assignment 8:

1. Write a program that prints your first name over and over.
2. How do you stop your program?
3. Write another that prints your name on one line, then a friend's on the next, over and over. Stop the program with the RESET key.
4. Write a program that uses each of these commands: PRINT, INPUT, LET, GOTO.

INSTRUCTOR NOTES 9 THE IF COMMAND

IF is a powerful but intricate command that is at the very heart of the computer as a logic machine.

Both verbal (the “cake” cartoon) and visual (the “fork in the road” cartoon) metaphors help in understanding the IF command.

The GOTO command has already introduced the idea that the flow of control down the program list may be altered. To that idea is now added the conditional test: if an assertion is true, one thing happens; if it is false, another.

The phrase “something A” is used for the assertion being tested for truth. The phrase “command C” is used for the command 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”:

$$\begin{array}{l} A\$ = B\$ \\ CX\$ < > D\$ \end{array}$$

The next level up we have the TRUTH or FALSITY of the assertion.

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

When you see “A = B” it may not REALLY be true that A equals B because the assertion may actually be FALSE.

The larger set of relations:

$$< , \quad > , \quad = , \quad = < , \quad = > , \quad < >$$

will be treated in later lessons.

QUESTIONS:

1. How do you make this program print “THAT’S FINE”?

```
10 DIM T$(5)
15 PRINT "DOES YOUR TOE HURT?"
17 INPUT T$
20 IF T$="NAH" THEN GOTO 90
40 IF T$="SOME" THEN GOTO 15
90 PRINT "THAT'S FINE"
```

2. Write a short program which asks if you like chocolate or vanilla ice cream. Answers to be “C” or “V.” For the “C” print “Yummy!” For the “V” answer, print “Mmmmmm!”

LESSON 9 THE IF COMMAND

Clear the memory and enter:

```
10 PRINT "clear"  
15 DIM A$(3)  
20 PRINT "ARE YOU HAPPY? (YES OR NO)"  
30 INPUT A$  
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 something A THEN command C
```

First the computer looks at "something A."

If it is true, the computer does the command C.

If "something A" is not true, then the computer goes on to the next line without doing the command C.



It looks like this:

```
10 IF    something A is true THEN  do command C
      and then go on to the next line.
```

or

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

SOME EXAMPLES OF IF

```
10 REM IF TEST PROGRAM
50 IF  A$="YES"      THEN  PRINT "GOOD"
55 IF  "YES"=A$      THEN  PRINT "GOOD"
60 IF  A$=B$        THEN  LET C$="NO WAY!"
70 IF  N$="BIRD"     THEN  PRINT "buzz"
75 IF  A$="READY"    THEN  PRINT "inverse"
```

Line 50 and line 55 do exactly the same thing.

Assignment 9A:

1. Add these lines to the program:

```
12 DIM A$(10), B$(10), C$(10), N$(10)
15 INPUT A$
17 LET C$="WHAT?"
20 LET B$="PAY UP"
25 LET N$=A$
85 PRINT C$
87 PRINT "inverse"
99 GOTO 15
```

Run the program and enter these words:

YES, BIRD, READY, NO WAY

and some other words you choose yourself. Look at what the program prints to see that IF commands are working as you expect. (Remember, if "something A" is true, then the command after the THEN is executed.)

2. Clear memory and write a program that asks if you are a "BOY" or "GIRL." If the answer is "BOY," the program prints "SNIPS AND SNAILS." If the answer is "GIRL," print "SUGAR AND SPICE."

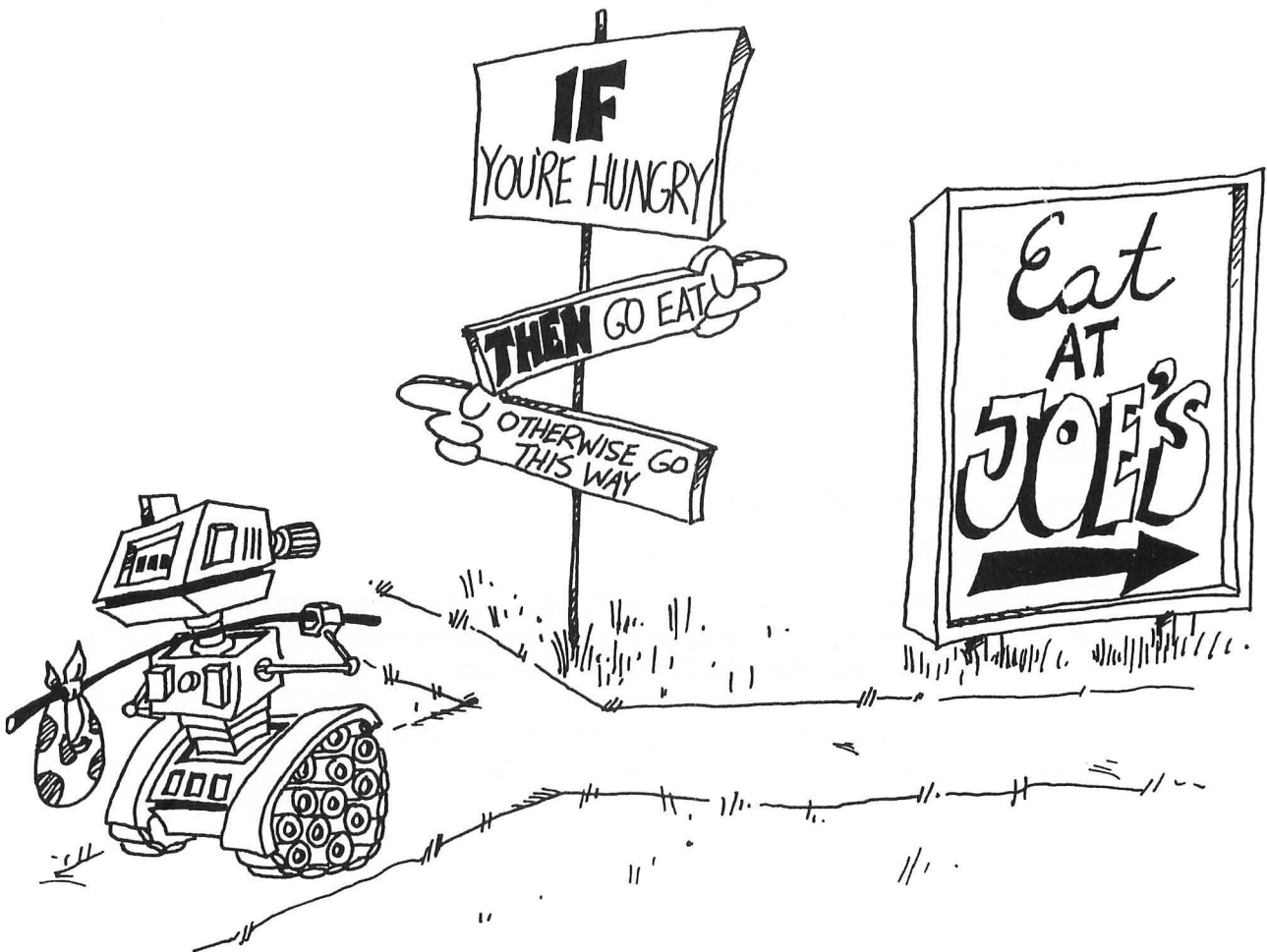
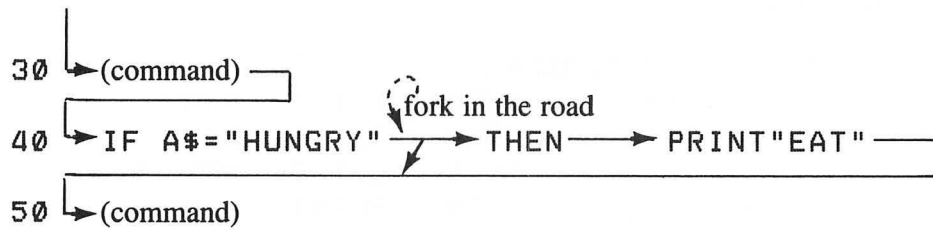
A FORK IN THE ROAD

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

If "something A" is true, it must go past the "THEN" and obey the command it finds there. Then it goes down to the next line.

If "something A" is false, it goes down to the next line right away.

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



THE "<>" SIGN

The "<>" sign means "not equal." It is the opposite of the "=" sign when used in an English phrase.

To make the "<>" sign, press the "<" key, then the ">" key.

```
40 IF      something  A      THEN      PRINT "NO SMOKE"
```

"Something A" is a phrase that is TRUE or FALSE. If it is true, then the computer prints "NO SMOKE." Look at this "something A" phrase:

```
Q$<>"FIRE"
```

and put it in an IF command:

```
40 IF      Q$<>"FIRE"      THEN      PRINT "NO SMOKE"
```

If the Q\$ box contains "COLD" then Q\$ is not equal to "FIRE" and the expression Q\$<>"FIRE" is TRUE. The computer will print "NO SMOKE."

If the Q\$ box contains "FIRE" then the phrase:

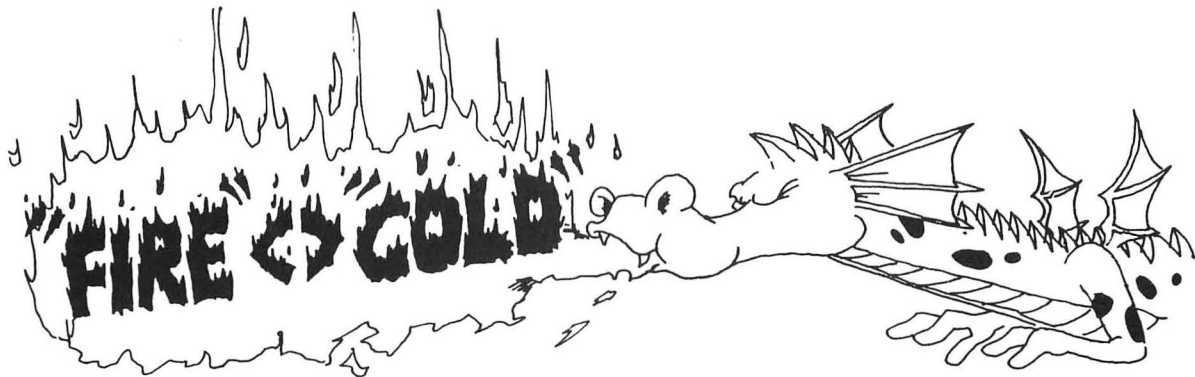
```
Q$<>"FIRE"      is FALSE      and
```

the computer will not print anything.

Here is how it looks in a program:

```
10 DIM Q$(4)
15 PRINT "IS YOUR HOUSE ON FIRE?"
20 PRINT "(ENTER 'FIRE' OR 'COLD')"
```

```
30 INPUT Q$
40 IF Q$<>"FIRE" THEN PRINT "NO SMOKE"
50 IF Q$= "FIRE" THEN PRINT "HELP"
```

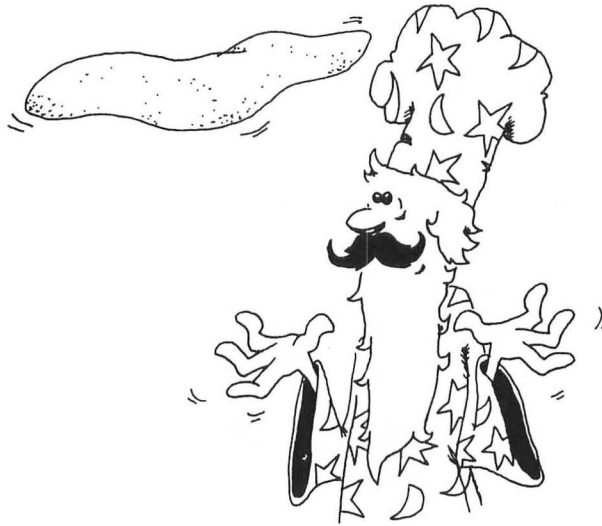


Assignment 9B:

1. Write a "pizza" program. Ask what topping is wanted. Make the computer answer something silly for each different choice. You can choose mushrooms, pepperoni, anchovies, green peppers, etc. 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 "something A"

G\$ < > C\$

for when the guess is wrong, and the other with an "=" sign for when the guess is right. The "command C" prints "wrong" or "right."



INSTRUCTOR NOTES 10 INTRODUCING NUMBERS

Numerical variables and operations are introduced. The LET, INPUT and PRINT commands are revisited.

The idea of memory as a shelf of “boxes” is extended to numbers. Again, variable names are limited to one letter for the time being.

The arithmetic operations are illustrated. The “*” symbol for multiplication will probably be unfamiliar to the student. Division will give decimal numbers, so it is nice if your student is familiar with them. But most arithmetic will be addition and subtraction, with a little multiplication, and a student unfamiliar with decimal numbers will not experience any disadvantage.

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

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

The non-standard use of “=” in BASIC, that it means “replace” and not “equal,” shows up strongly in the statement:

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

The cartoon uses the box idea to illustrate this meaning of “=”.

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 “ $7 = 7 + 1$ ” in arithmetic.
3. Give another example of “bad arithmetic” in a LET command. Use the * or / symbols.
4. Explain what is meant by the “name of a variable” and the “value of a variable” for numerical variables. For string variables.
5. If the boxes A and B have the numbers 5 and 8 in them, then PRINT A;B will print 58. How do you make the computer print 5 8 instead of 58?

LESSON 10 INTRODUCING NUMBERS

INPUT, LET AND PRINT

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

```
10 REM BIGGER
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
```

MEMORY BOXES

Numbers go into memory boxes, just like strings. There is one difference:

Strings: You must use DIM to tell how large the box is.

Numbers: All number boxes are the same size, and the DIM command is not used.

Here is what happens in the BIGGER program.

Line 10 A REM, so the computer ignores it.

Line 20 PRINTS the string.

Line 30 The computer waits for a number to be INPUTed. It takes the number and looks for a box named N. The computer does not find a box named N because N has not been used in this program before. So the computer takes a new box, writes N on the front, and puts the number into it.

Line 40 The computer takes the number in box N, adds one to it, and puts it into another box and puts the name A on the front.

Line 45 PRINTs an empty line.

Line 50 PRINTs a message.

Line 60 Goes to box A, takes a copy of the number out, and PRINTs the number. (The original number is still in box A.)

ARITHMETIC

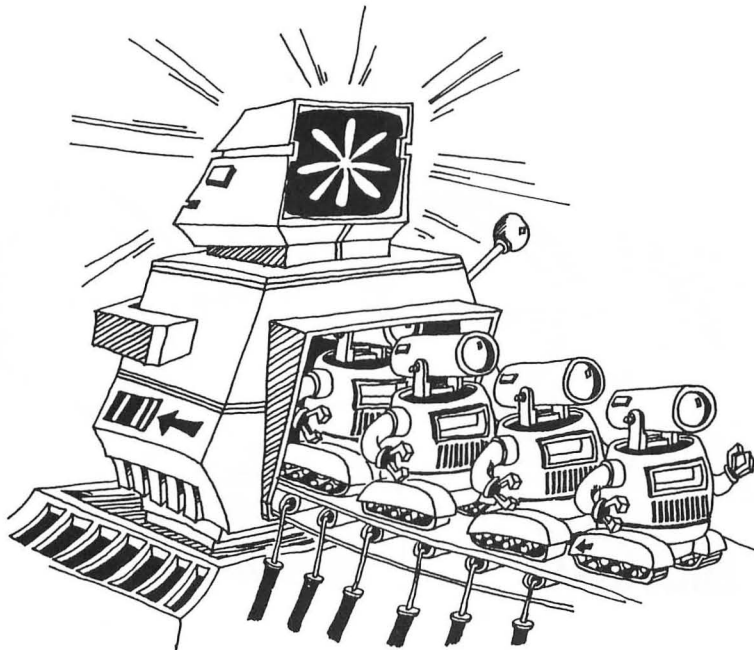
The minus sign, the equal sign, the plus sign, and the multiplication sign are on the cursor arrow keys.

The computer uses “*” 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 as the question mark.

Answers to division problems are given as decimal numbers.



VARIABLES

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

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

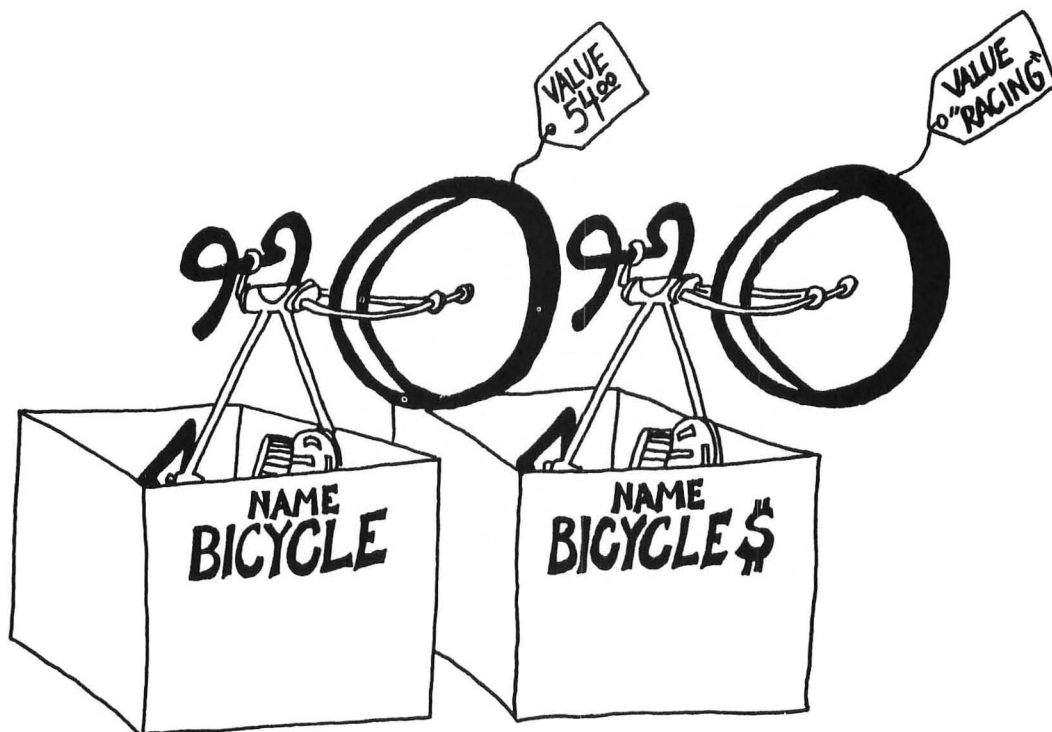
The thing that is put into the box is called the “value” of the variable.

ARITHMETIC IN THE LET COMMAND

```
10 LET A=2
20 LET B=3
30 LET C=B-A
40 PRINT A;" ";B;" ";C
```

Some more examples:

```
10 LET B=15
20 LET A=B/5
30 LET X=A*4+2
40 PRINT X;" ";A
```



CAREFUL!

Numbers and strings are different. Example: "1984" is not a number. It is a string constant because it is in quotes.

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

Some numerical constants: 5, 22, 3.14, -50

Some string constants: "HI", "7", "TWO", "3.14"

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

Correct: `10 LET A = 3 + 7`

Wrong: `10 LET A$ = 3 + 7`

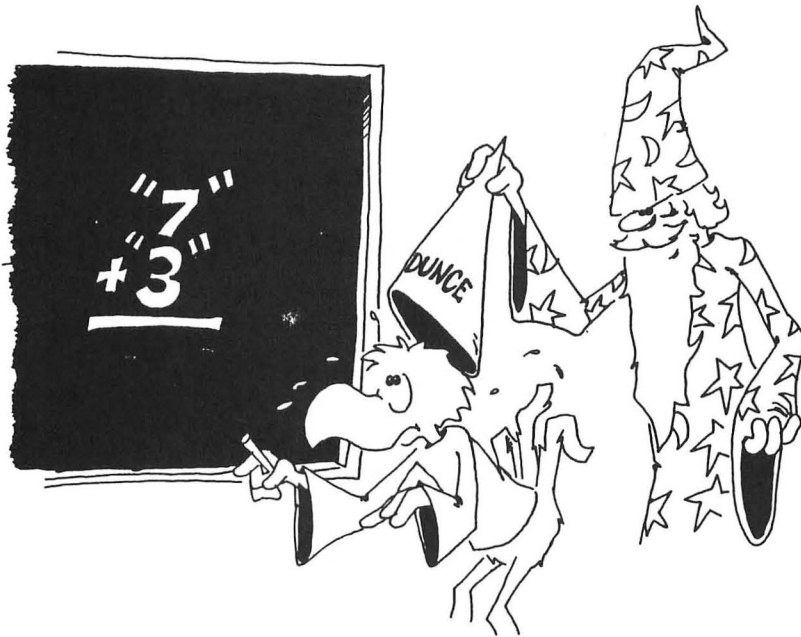
Wrong: `10 LET A$ = "3" + "7"`

Wrong: `10 LET A "3" + "7"`

If you enter any of these wrong lines, the computer will print:

ERROR -

The two types of variables are "string" and "numerical." You cannot mix them.



Enter:

```
10 DIM B$(15)
15 LET A=5
20 LET B$="10"
30 LET C=A+B$
```

Lines 10, 15, and 20 are OK, line 30 is wrong. What will the computer do when you enter line 30? Try it.

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 command. (Just remember that you cannot do arithmetic with the mixture.)

Correct:

```
PRINT A;"SEVEN";"7"
```

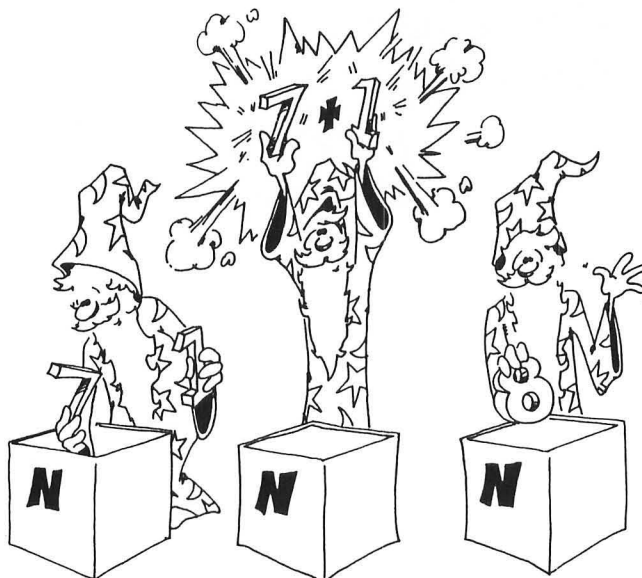
Correct:

```
PRINT A;B$
```

Run this line.

```
10 PRINT 5/2;" IS EQUAL TO 5/2"
```

What do you get? _____



A FUNNY THING ABOUT THE EQUAL SIGN

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

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

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

$$7 = 7 + 1$$

which is not correct.

But it is OK 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 into the box.

Another way to say the same thing is:

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

 means

LET N = N + 1

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

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.)

INSTRUCTOR NOTES 11 DELAY LOOPS, SOUND

This lesson introduces loops in a painless way.

Delay loops slow the program down so that its operation can be more easily observed. They also are used for portions of the program that must run at certain speeds, and should then be called "timing loops."

The delay loop is all on one line with a colon to separate off the NEXT command. The amount of delay is determined by the size of the loop variable. A value of 500 gives about a one second delay.

After the student sees that the loop simply counts until a particular value is reached before going on to the next instruction, it will be easier to handle loops in which things are going on inside.

The SOUND command is introduced. Of the 4 arguments, only the pitch is dealt with. SOUND will be treated more fully later.

QUESTIONS:

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

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

3. Write a two line program to make a tone. Use a delay loop for one second. Try different numbers for pitch. Which numbers give high notes? Which give low notes?

LESSON 11 DELAY LOOPS, SOUND

DELAY LOOPS

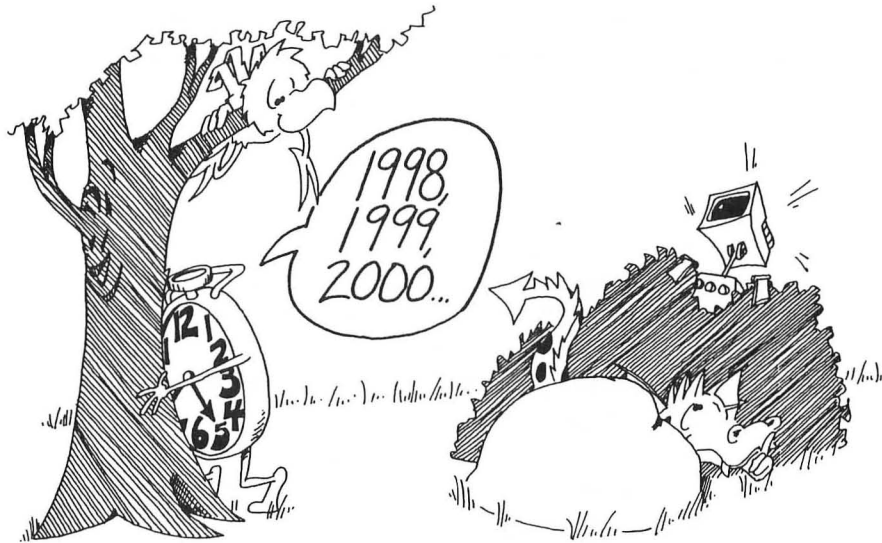
Here is a way to slow down parts of the program.

It is a "delay loop."

Run this program:

```
10 REM DELAY LOOP
20 PRINT "clear"
30 PRINT "WAIT"
40 FOR I=1 TO 2000:NEXT I
50 PRINT "DONE"
```

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



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

Each 500 in the delay loop is worth about 1 second of time. Try this:

```
10 REM ---- TICK TOCK ----
20 PRINT "clear"
30 INPUT "WAIT HOW LONG? "; S
36 T=S*500
40 FOR I=1 TO T:NEXT I
45 PRINT : PRINT "buzz"
50 PRINT S;" SECONDS ARE UP"
```

YOUR NAME IS FALLING!

```
10 PRINT "clear"
15 LET N=1
20 PRINT"YOUR FIRST NAME"
30 INPUT W$
40 PRINT W$
45 FOR T=1 TO 10*N:NEXT T
50 LET N=N+1
60 GO TO 40
```

Press BREAK to stop the run.

This program prints your name down the screen.

Assignment: 11A

1. Write an "insult" program. It asks your name. Then it clears the screen and writes your name. Then it waits for 2 seconds, prints an insult and buzzes.

SOUND

Run this:

```
10 REM TONES
15 REM -----FIRST TONE
20 PRINT "clear START MIDDLE C"
30 SOUND 1, 121, 10, 8
40 FOR T=1 TO 1000:NEXT T
45 PRINT "STOP"
50 SOUND 1, 150, 10, 0
55 REM -----PAUSE
60 FOR T=1 TO 500:NEXT T
61 REM -----SECOND TONE
65 PRINT "START AGAIN, LOWER"
70 SOUND 1, 162, 10, 8
80 FOR T=1 TO 1000:NEXT T
90 PRINT "END"
```

If you do not hear the tones, then turn up the volume on your TV monitor.

The SOUND command has 4 numbers after it.

The second number tells the pitch of the note (number 121 in line 30 and number 162 in line 70). Use numbers from 1 to 255.

The fourth number tells how loud. Use "8" for normal loudness.

Line 50 shuts off the sound by making the loudness equal to zero.

The sound also shuts off automatically when the program ends.



Assignment 11B:

1. Write a Sound Program that lets the user choose which tone to play and how long it lasts. End it with a GOTO so the user can try out many sounds.
2. Write a digital "clock" program that uses a delay to count seconds and print them out. When 60 seconds have gone by, add one to the minutes and put seconds back to zero. (See next lesson.) Same with hours. Run the clock a long time and adjust the timing loop so the clock keeps good time. Add chimes on the quarter hour.

INSTRUCTOR NOTES 12 THE IF COMMAND WITH NUMBERS

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

= > < <>

The use of nested IF's is demonstrated.

A "home made" loop is demonstrated in the GUESSING GAME, but not discussed. The loop starts in line 50 and goes to 80. The exit test is made in line 70. The logic of this loop is that of a DO UNTIL.

QUESTIONS:

1. What part of the IF command can be TRUE or FALSE?
2. What follows the THEN in an IF command?
3. After this little program runs, what will be in box D?

```
10 LET D=4
```

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

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



LESSON 12 THE IF COMMAND WITH NUMBERS

Try this:

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

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

```
10 IF          something is true  THEN  do command  C
```

"Something A" can have these arithmetic symbols:

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

Each "something A" is a phrase. It is written in "math language" but you should say it out loud in English. For example:

A <> B	is pronounced	"A is not equal to B"
5 < 7	is pronounced	"five is less than seven"

PRACTICE

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

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

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

AN IF INSIDE AN IF

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

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

To understand this, break it into two parts:

```
60 IF A>12 THEN      (command C)      where  
      (command 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 command is not reached, so nothing is printed.

Assignment 12A:

1. Draw the "fork in the road" diagram for line 60 above. There will be two forks on the diagram. (See page 54 .)

GUESSING GAME

```
10 REM --- GUESSING GAME ---  
15 PRINT "clear"  
20 PRINT "TWO PLAYER GAME"  
25 PRINT  
30 PRINT "FIRST PLAYER ENTER A NUMBER FROM 1 TO 100"  
35 PRINT "WHILE SECOND PLAYER ISN'T LOOKING"  
37 PRINT  
40 INPUT N  
45 PRINT "clear"  
50 PRINT "MAKE A GUESS ";  
55 INPUT G  
60 IF G<N THEN PRINT "TOO SMALL"  
65 IF G>N THEN PRINT "TOO BIG"  
70 IF G=N THEN GOTO 90  
80 GOTO 50  
90 REM THE GAME IS OVER  
92 PRINT  
95 PRINT "THAT'S IT!"
```

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

Usually line 80 sends you to line 50 so you can make more guesses. But if $G = N$ in line 70, then you skip to line 90 and print "THAT'S IT!".

Assignment 12B:

1. Tell what happens in lines 50 through 80:

If G is 31 and N is 88:

50 _____

55 _____

60 _____

65 _____

70 _____

75 _____

80 _____

If G is 88 and N is 88:

50 _____

55 _____

60 _____

65 _____

70 _____

80 _____



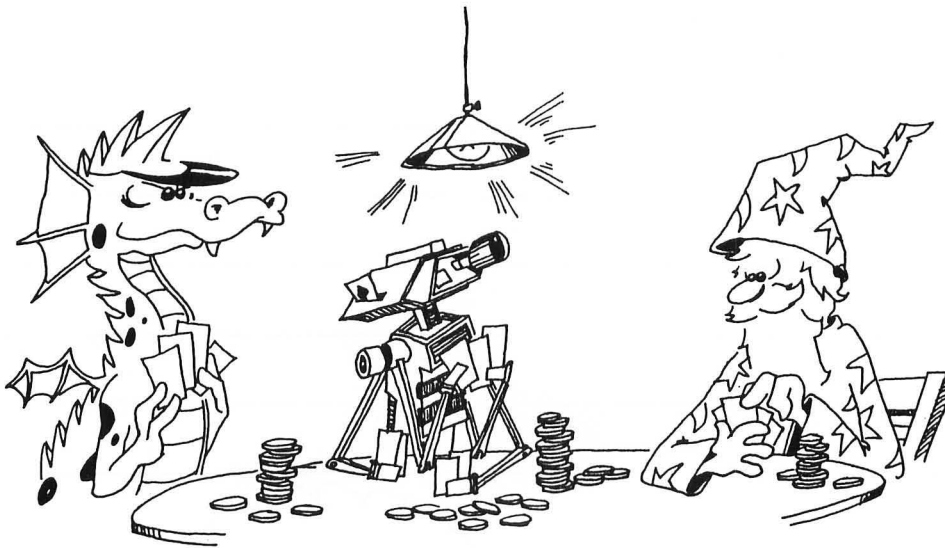
2. Here is another program. What will it print, and how many times?

```
10 LET N=1
20 IF N=13 THEN PRINT "UNLUCKY!"
30 LET N=N+2
40 IF N>30 THEN GOTO 99
50 GOTO 20
99 PRINT "DONE"
```

What will it print if line 10 is changed to:

```
10 LET N=2
```

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



INSTRUCTOR NOTES 13 RANDOM NUMBERS AND THE INT FUNCTION

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

The RND function produces psuedo-random decimal numbers between 0.0 and 1.0. Such numbers are directly usable as probabilities, but integers over some range such as 1 to 6 for a die, or 1 to 13 for a suit of cards are often more to the point.

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 larger range than 0 to 1, conversion to an integer is desired. The INT function does this by simply truncating the number, "throwing away the decimal part." (For negative numbers the situation is a little more complicated, and that rare case is not treated here.)

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

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

QUESTIONS:

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

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

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

2. Tell how the INT() function is different from "rounding off" numbers. Which is easier for you to do?
3. Tell how to change a number so that the INT() function will round it off.
4. What does the RND(9) function do?
5. How can you get random integers (whole numbers) from 0 through 10. (Hint: INT(RND(9)*10) is not quite right.)
6. How can you get random integers from 5 through 8?

LESSON 13 RANDOM NUMBERS AND THE INT FUNCTION

THE RND FUNCTION

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

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

The computer needs some way to let you "roll dice" and "deal cards" and do many other unpredictable things.

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

Run this program:

```
10 REM RANDOM NUMBERS
20 PRINT "clear"
25 LET N=RND(9)
30 PRINT N
40 IF N<.95 THEN GOTO 25
```

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

It doesn't matter what number you put in the parentheses just so long as it is positive. I chose "9" because it is near the "()" signs on the keyboard making it easy to type (9).



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

Change the program above to:

```
25 LET N=RND(9)*52
40 IF N<45 THEN GOTO 25
```

and run it again.

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

But:

We usually want whole numbers like 7 and 8 rather than decimal numbers like 7.03454323 and 8.89746582. Do this by using the INT function.

THE INT FUNCTION

The INT function takes the number in its parentheses and throws away the decimal part, leaving an integer.

Try the INT function in this little program:

```
10 LET I=INT (6.3)
20 PRINT I
```

And in this:

```
10 LET X=0.3
20 PRINT "X= ";X
30 PRINT "INT(X)= ";INT(X)
```

And this:

```
10 LET X=.3
20 LET Y=2.5
30 LET P=X+Y
40 LET Q=INT(X+Y)
50 PRINT P,Q
```

Look at the answers to see that the decimal part was thrown away.

Try this:

```
10 REM ----- INT -----
20 PRINT "clear"
30 PRINT "GIVE ME A DECIMAL NUMBER "
32 INPUT D
35 LET I=INT(D)
40 PRINT "DECIMAL ";D;" INTEGER ";I
50 IF I<>0 THEN GOTO 30
```

Enter 0 to end the program.



ROLLING THE BONES

Usually 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 //////////
15 DIM Y$(1)
20 PRINT "clear"
30 LET R=RND(9)
40 PRINT "RANDOM NUMBER ";R
50 LET S=R*6
55 PRINT "TIMES 6 ";S
60 LET I=INT(S)
65 PRINT "INTEGER PART ";I
70 LET D=I+1
75 PRINT "DIE SHOWS ";D
77 PRINT
80 PRINT "ANOTHER? <Y/N> ";
82 INPUT Y$
85 IF Y$="Y" THEN 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(9))`

Here is how to save a lot of room.

Instead of:

```
30 LET R=RND(9)
50 LET S=R*6
60 LET I=INT(S)
70 LET D=1+I
```

Use just: `70 LET D=1+INT(RND(9)*6)`

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 “paper, scissors, and rock” game, you against the computer. (Paper wraps rock, rock breaks scissors, scissors cut paper.) The computer chooses a number 1, 2 or 3 using the RND() function: 1 is paper, 2 is rock, 3 is scissors. You INPUT your choice as P, R, or S and the computer figures out who won and keeps score.



INSTRUCTOR NOTES 14 SAVING TO TAPE

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

The commands: `CSAVE` `CLOAD` are introduced.

The only other commands used are:

<code>NEW</code>	<code>REM</code>
<code>LIST</code>	<code>PRINT</code>

This lesson can be used any time 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 judgement should prevail, and you can insert this chapter at an earlier point in the flow of lessons so that your student can save some programs he/she is particularly proud of.

Ordinary audio tape is usually satisfactory for computer use. However, remember that a tiny imperfection can cause the tape to "drop a bit" and this makes the program wrong, or worse, unloadable!

You will not need long tapes. In fact, the special 10 minute data tapes are inexpensive and convenient.

The ATARI computers support use of named file and autobooting tapes but this is not discussed in this book. Please refer to the ATARI 400/800 manual.

QUESTIONS:

1. When you enter `CSAVE`, the computer buzzes twice. What does that mean?
2. What command tells the computer to get a program from tape?
3. About how long does it take to save a short program?
4. If a program is put onto tape, is it still in the computer's memory?

LESSON 14 SAVING TO TAPE

ENTERING A PROGRAM

If you have a program you want to save at this moment, skip to SAVING A PROGRAM.

If not, enter:

```
NEW
10 REM :::HI:::
20 PRINT "HI"
```

SAVING A PROGRAM

Put a brand new tape in the ATARI 410 program recorder and rewind it.

Press the little button beside the "speedometer" dial on the recorder. This resets it to zero. (More exactly to "000.")

Enter:

```
CSAVE
```

You will hear the buzzer give two "beeps." This is a signal to press two keys on the recorder.

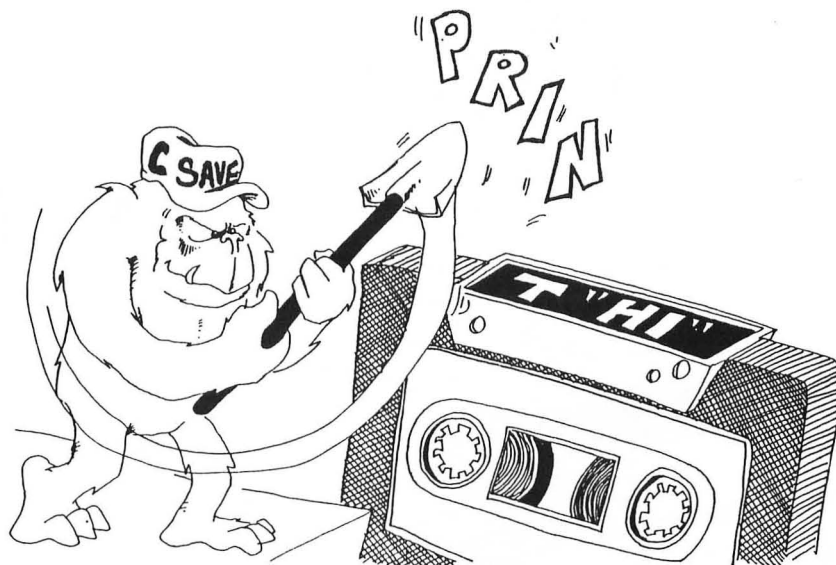
Press the

REC and the PLAY keys together.

Now you must press the

```
RETURN
```

key on the computer again.



The computer turns on the motor of the recorder and lets about 10 seconds go by so that a "leader" is left on the tape. Then it starts to record the program on tape.

After about 20 seconds (for the short program) the computer will finish recording and turn off the recorder motor. It will print:

READY

The dial on the recorder will read about 10.

MAKE A LITTLE LIST

You should write the name of the program on the front of the tape cassette, and put after it the number showing on the dial.

This is where the program ends. (It starts at zero or "000" on the dial.)

CAREFUL! If this is an important program, I suggest you put a second copy on the tape, right after the one you just did. (That is, don't rewind the tape. Just start where the directions above say **CSAVE** and continue on.)

If this is a short practice program, just go on to **LOADING THE PROGRAM** below.



LOADING THE PROGRAM INTO THE COMPUTER

Let's practice loading the program we just saved.

First, enter `NEW`

Then `LIST`

to erase the program from the computer. (Otherwise we won't know if it loaded from tape or just was left over from before.)

Rewind the tape.

Enter: `CLOAD`

Press: `RETURN`

The buzzer will beep once, telling you to press one key on the recorder.

Press the `PLAY` key on the recorder.

Then press `RETURN` on the keyboard again.

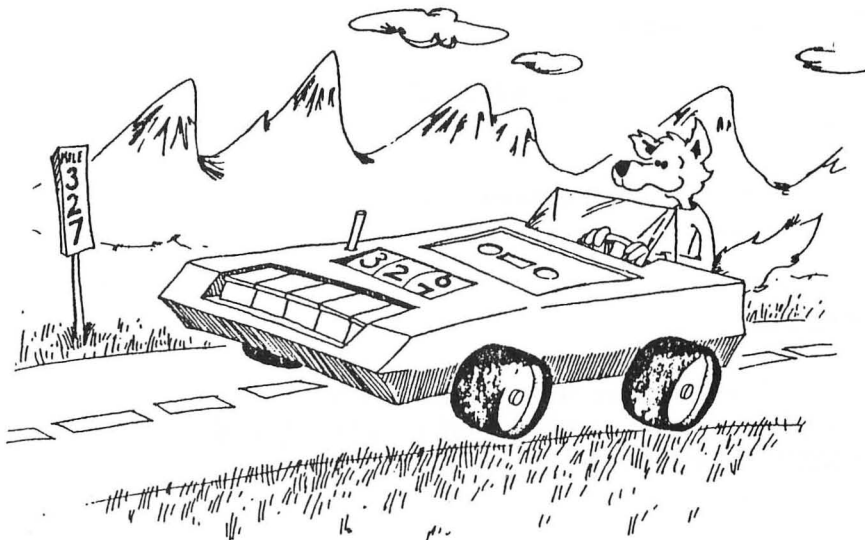
The computer will start the recorder motor and look for a program.

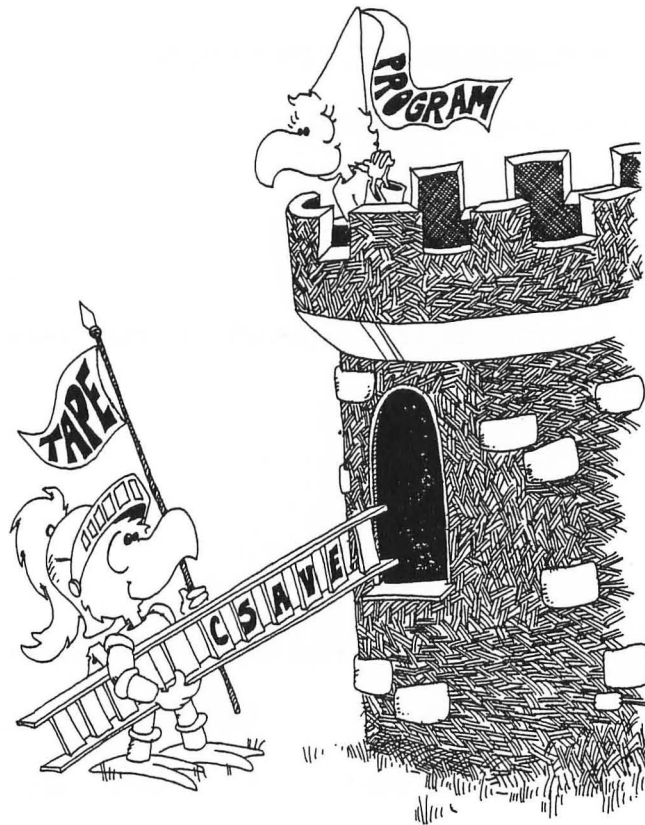
You will see the dial on the recorder start turning again.

After about 20 seconds, it will print:

Enter `READY`
`LIST`

to see if the program got into the computer memory OK.





HOW MANY PROGRAMS ON ONE TAPE?

If you use the dial on the recorder and keep a good list, you can put many programs on one tape. Just follow the directions starting at CSAVE. (Of course, you do not rewind between programs.)

But the more programs you put on tape , the harder it is to find them again, even with the help of the dial readings.

With many programs on a tape, it is more likely that you will make some mistake and ruin a lot of programs.

When you start writing long programs (over 25 lines), you will probably put only one program on each tape.

Assignment 14:

1. Write a short program (4 lines) and SAVE it on tape.
2. Do NEW, and write another short program. SAVE it.
3. Do NEW. Then load and run each program.

INSTRUCTOR NOTES 15 SOME SHORTCUTS

This lesson covers:

? used for PRINT
LET omission
: used between statements on a line

The sprint is over. We have reached RND and the saving of programs to tape. All the elements are in place for the student to write substantial programs. Programs will get longer and some shortcuts will help in writing them.

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

The colon can mess up a program too. Some statements are reached by GOTO's. If you move such a statement to the middle of another line, you will get an error message upon running the program.

A more subtle error that even experienced programmers occasionally make is to move a statement to the back of a line that has an IF in it. This changes the logic of the program, as now the statement will be executed only if the IF condition is true.

On the other hand, the colon in BASIC allows one to put a little "subroutine" consisting of several statements after an IF. This makes using a GOTO unnecessary for reaching the extended segment of the program: a shorter and much less cluttered program results. So the colon becomes a powerful and nontrivial means of improving the clarity of the program.

QUESTIONS:

1. What shortcut does the "?" give?
2. How can you tell that the word LET is missing from a LET command?
3. Why is it sometimes good to put two statements on the same line, separated by a colon?
4. What is wrong with each of these lines?

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


LESSON 15 SOME SHORTCUTS

A PRINT SHORTCUT

Instead of typing PRINT, just type a question mark.

Run: `10 ? "HI"`

The computer understands that the question mark means PRINT.

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.



A LIST SHORTCUT

There are 3 ways to use the LIST command:

LIST	lists whole program
LIST 10	lists line 10
LIST 10,34	lists all lines from 10 through 34

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 TO USE THE COLON SHORTCUT

Use the shortcut:

1. To make the program clearer.

Put similar statements on the same line. Example:

Instead of:

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

write:

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

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

Example:

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

THE COLON AFTER AN IF COMMAND

You can make neater IF statements using colons.

Without:

```
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 commands in the path "A=0 is TRUE" are on the line after THEN.

CAREFUL!

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

Example:

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

is not the same as:

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

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

SOME MORE MISTAKES WITH COLONS

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

Correct:

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

Wrong:

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

Because the computer jumps to the next line after reading REM.

Correct:

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

Wrong:

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

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

COMMANDS, STATEMENTS AND LINES

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

```
PRINT, NEW, RUN, LIST, REM, INPUT, LET,  
GOTO, IF, CSAVE, CLOAD,
```

Commands used in numbered lines may be called "statements." Used alone, they are always called "commands."

Enter: LIST

We say we have "entered a command."

But if we write this line in a program:

```
20 LIST
```

We say that line 20 has one “statement,”
the LIST command.

Some lines have several statements, separated by colons.

```
30 DIM E$(29):PRINT:LET Z=55
```

is a line with three statements.

Assignment 15:

1. Write a program that uses each of these shortcuts at least once.
2. Write a “silly 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 GRAPHICS CHARACTERS, POSITION

The POSITION command is used to move the output cursor to any point on the text screen.

These commands are used for flexible manipulation of text and/or for graphics.

To make effective use of this command, the screen needs to be thought of as a 40 character across by 24 line down grid.

ATARI computers have graphics characters that appear when the CTRL key is held down and letter and punctuation keys are pressed. These include various lines and corners as well as diagonals and even card suit symbols.

QUESTIONS:

1. If you want to print the next word on line 12, what command do you use?
2. If you want to print the next character on line 6, indented by 20 spaces, what command do you use?
3. Show how to print the two words "FAT" and "CAT" on the same line with "CAT" printed first, starting at space 25, and then "FAT" printed starting at 5.



LESSON 16 GRAPHICS CHARACTERS, POSITION

IN GRAPHICS 0 MODE

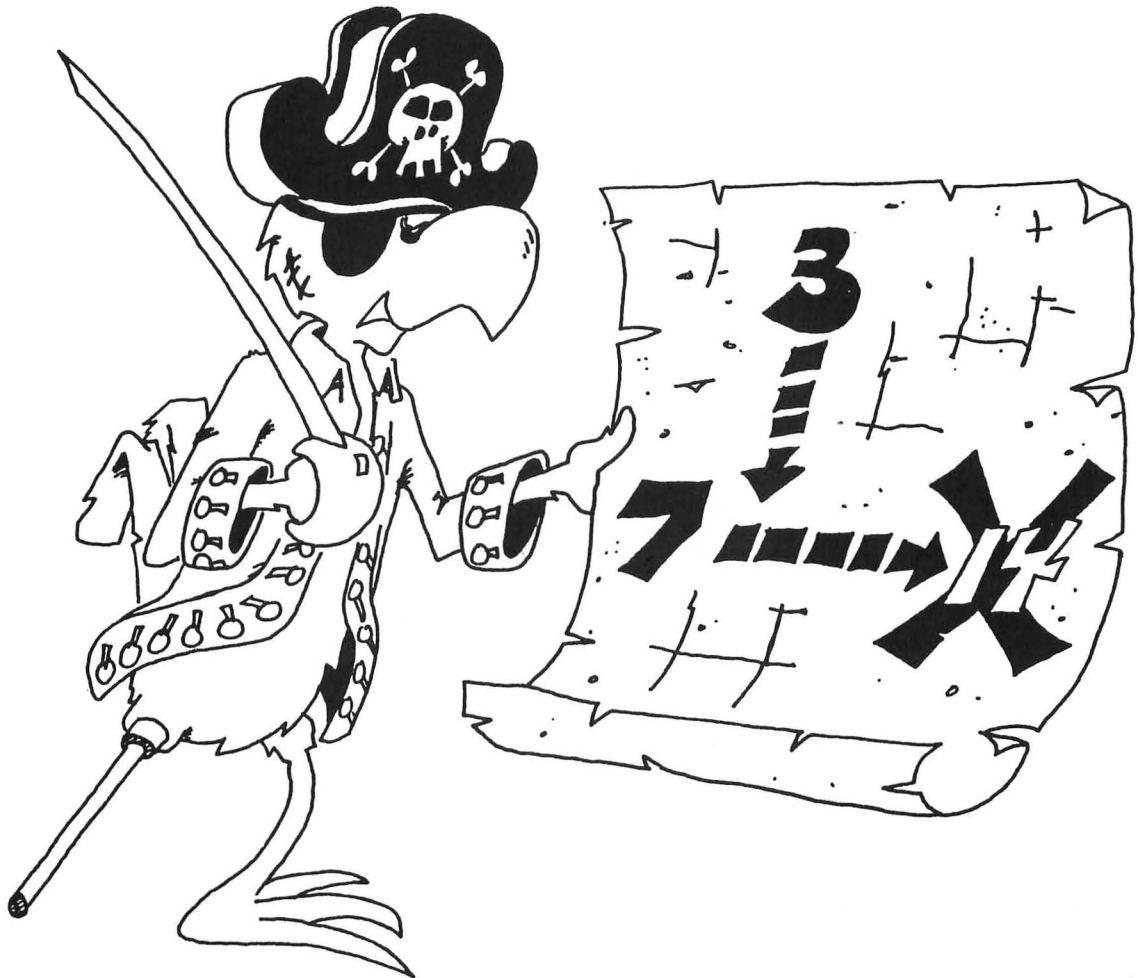
There is room for 24 lines of typing on the screen. The lines are numbered from 0 at the top to 23 at the bottom.

Each line can hold 40 characters. They are numbered from zero on the left to 39 on the right.

Run this:

```
10 REM POSITION DEMO
15 PRINT "clear"
20 POSITION 0,10:PRINT "LINE 10 FIRST"
25 FOR T=1 TO 500:NEXT T
30 POSITION 0, 1:PRINT "LINE 1 NEXT"
35 FOR T=1 TO 500:NEXT T
40 POSITION 0,17:PRINT "LINE 17 LAST"
```

The second number in POSITION tells which row the printing cursor will go.



JUMPING ANYWHERE ON THE SCREEN

Run:

```
10 REM COLUMN AND ROW
15 PRINT "clear"
20 PRINT "WHICH ROW";:INPUT R
30 PRINT "WHICH COLUMN";:INPUT C
40 POSITION C,R:PRINT "*";
45 FOR T=1 TO 500:NEXT T
50 GOTO 20
```

Press BREAK to stop the program.

ERASING WHAT YOU WRITE

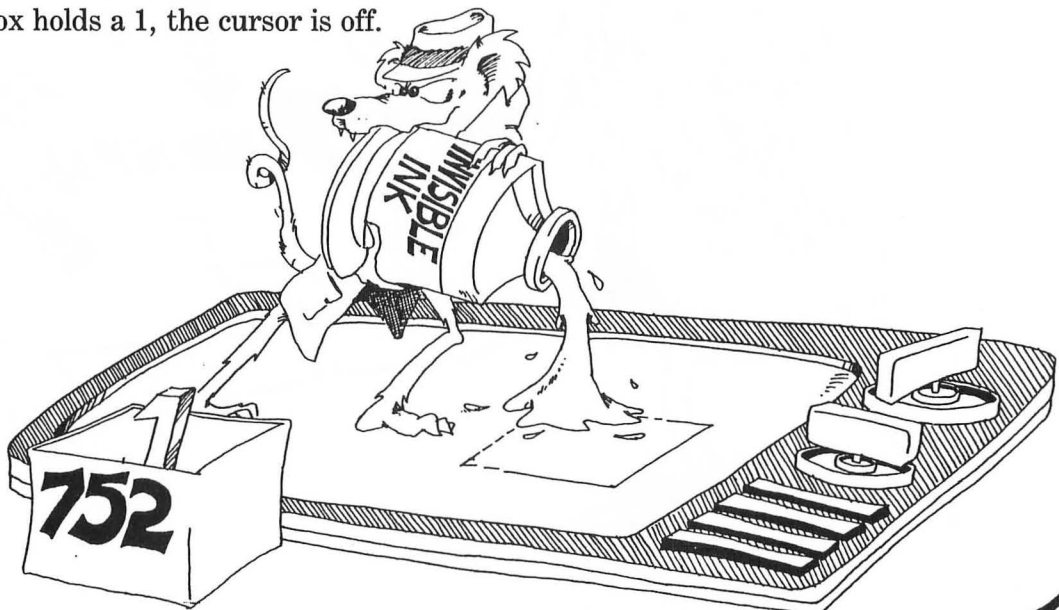
```
10 REM JUMPING HERE
12 POKE 752,1:REM TURN OFF CURSOR
15 PRINT "clear":POKE 752,1
20 X=INT(RND(9)*36)
25 Y=INT(RND(9)*24)
30 POSITION X,Y: PRINT "HERE"
50 FOR T=1 TO 200:NEXT T
60 POSITION X,Y: PRINT "  "
80 GOTO 20
```

MAKING THE CURSOR INVISIBLE

Look at line 15. POKE puts a number 1 in the memory box 752. This box tells the cursor to be "on" or "off."

When the box holds a zero, the cursor shows on the screen.

When the box holds a 1, the cursor is off.



GRAPHICS CHARACTERS FROM THE KEYBOARD

You are in the GRAPHICS 0 screen mode.

You can print graphics symbols directly from the keyboard.

Try this: Hold down the CTRL key and press any letter key, semicolon, comma, or period. You see graphics symbols.

Now press the "ATARI" key once, then again hold down the CTRL key and type letters. This makes the inverse of the characters.

All together, there are 58 characters you can use.

These can be put in PRINT commands to make the program print a picture.

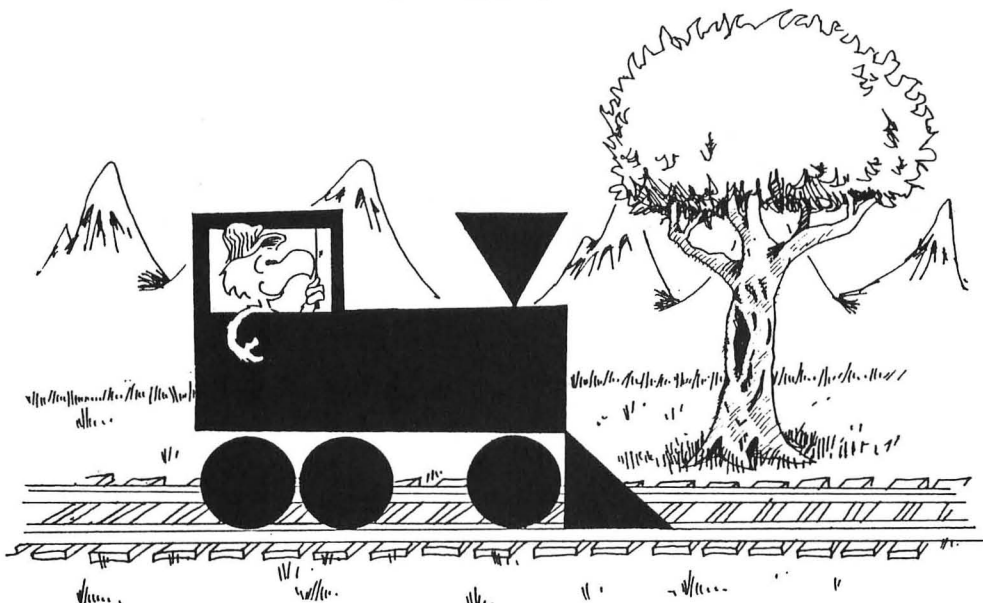
Use POSITION to put the parts of the picture in the right spots on the screen.

Run:

```
10 REM CHOO-CHOO
12 PRINT "clear"
20 POSITION 10,10
30 PRINT "■ ■ ▽"
32 POSITION 10,11
34 PRINT "■■■■■"
36 POSITION 10,12
38 PRINT "● ● ● ▴"
```

Assignment 16:

1. Use the RND() function to write your name at random places on the screen. Make it write your name many times all over the screen.
2. Use POSITION to write your name in a large "X" on the screen.
3. Draw the rest of the train by using graphics in PRINT commands.



INSTRUCTOR NOTES 17 FOR-NEXT LOOPS

FOR, NEXT and STEP commands which make loops are described in this lesson.

The loop is made of two statements, one starting with FOR and the other with NEXT. These commands may be separated by several lines and yet are strongly interdependent. This could be a bit confusing to your student. The delay loop in a previous lesson helps form the notion that the FOR... and the NEXT are coupled. It remains then to show the utility of repeating a set of commands in the middle of the loop.

Nested loops are introduced using a case where the inside loop is a delay loop.

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

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

From then on, all the looping action takes place at the NEXT command. 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 negative STEPs) NEXT passes control to the statement after itself. Otherwise, it sends control to the statement after the FOR command.

Because the loop variable is treated just like any other variable by BASIC, it can be used or changed in the body of the loop. Changing it should be done with care, as it will be further changed by the NEXT which also uses it to decide if the loop has ended.

QUESTIONS:

1. Write a loop that prints the numbers from 0 to 20.
2. Write a program loop that prints the numbers from 30 down to 20, by twos.
3. Write a pair of nested loops to print the numbers 100, 200, 300 and between them, the numbers 1, 2, 3, 4, 5 on separate lines.

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 "clear"
30 FOR I=5 TO 20
40 PRINT I
50 NEXT I
```

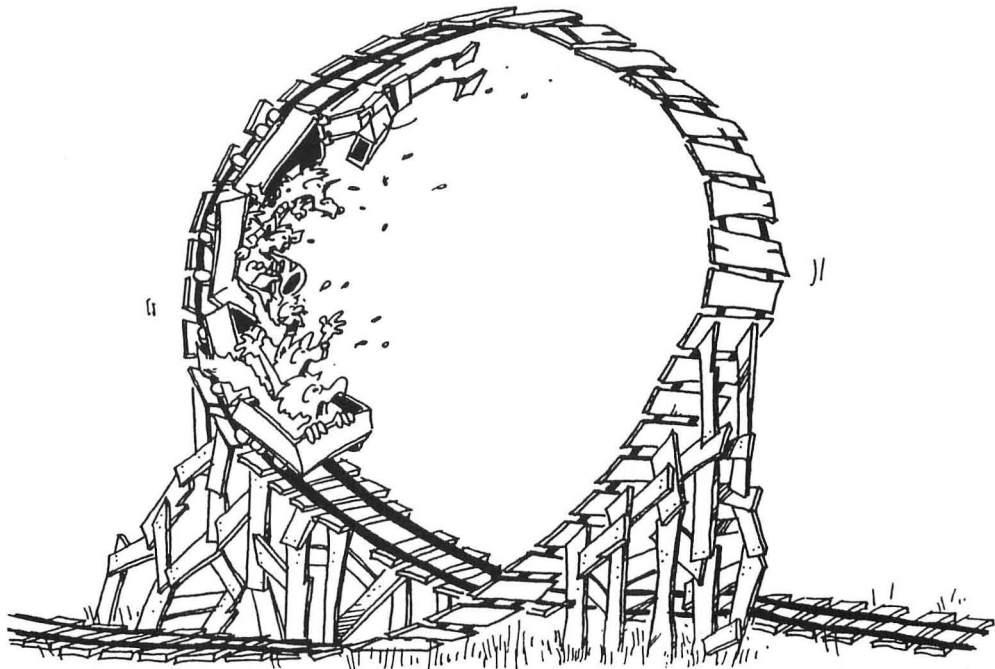

The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

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

MARK UP YOUR LISTINGS

Show where the loop is by a bracket:

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



THE STEP COMMAND

The computer was counting by one's in the above programs. To make it count by two's, change line 30 to this:

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

Assignment 17A:

1. Have the computer count by five's from zero to 100.

COUNT DOWN LOOPS

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

Try this:

```
10 REM *** APOLLO 11 ***
20 PRINT "clear"
30 PRINT "T MINUS 12 SECONDS AND COUNTING"
35 FOR T=1 TO 500:NEXT T
40 FOR I=11 TO 0 STEP -1
50 PRINT I:PRINT "buzz"
60 FOR J=1 TO 500:NEXT J:REM TIMING LOOP
70 NEXT I
80 PRINT "ALL ENGINES RUNNING. LIFT OFF,"
81 FOR T=1 TO 500:NEXT T
82 PRINT "WE HAVE A LIFT OFF,"
83 FOR T=1 TO 500:NEXT T
84 PRINT "32 MINUTES PAST THE HOUR,"
85 FOR T=1 TO 500:NEXT T
86 PRINT "LIFT OFF ON APOLLO 11."
```

Line 60 is the timing loop.

(If you got tired of all those FOR T=. . . relief is at hand. When we get to GOSUB. . . it will save all that repetition.)

NESTED LOOPS

In this program, we have one loop inside another.

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

The inside loop is in line 60.

These are 'nested'. It is like the baby's set of toy boxes which fit inside each other.



LOOP VARIABLES

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

```
60 FOR J=1 TO 500:NEXT J
```

J is the loop variable. And for the loop starting in line 40:

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

I is the loop variable.

BADLY NESTED LOOPS

The inside loop must be all the way inside:

Right:

```

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

Wrong:

```

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

Assignment 17B:

1. Write a program that prints your name 15 times.
2. Now make it indent each time by 2 spaces more. It will go diagonally down the screen.
3. Now make it write your name 24 times, starting at the bottom of the screen and going up.
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 5 times.
5. Make the locomotive picture in the last lesson move across the screen.
Remember: "Draw, erase, draw, erase . . ." Use the POSITION command to put each new locomotive in the correct spot. Then a delay loop. Then erase by writing blanks to the same spot. Then draw the new picture. Use a loop whose loop variable gives the column number where the picture starts.

INSTRUCTOR NOTES 18 EDIT AND RUN MODES, THE CALCULATOR

This lesson explains the EDIT MODE and the RUN MODE of the computer.

We placed this material rather late in the book, despite its fundamental nature, because it is abstract and because we did not wish to slow down the race to mastery of the core commands in BASIC.

However, you may want to take up this chapter at some earlier time in the course. The only commands used in this lesson are:

PRINT and RUN

Other names for these modes are:

Edit mode:	direct mode	immediate mode
	calculator mode	command mode
Run mode:	deferred mode	

The edit mode is the home base of the computer user. In the edit mode, you enter a line. The characters go into the input buffer.

When RETURN is pressed, the computer looks to see if the line starts with a number. If so, it stores the line in the program space, making room at the right location so that the lines are numbered in order.

If the line doesn't start with a number, the computer executes the line right out of the input buffer. Most commonly, the line consists of a single command, like LIST or RUN. But the immediate mode is a very powerful one in that fairly long one line programs can be executed. This feature is handy both in the program design phase, where arithmetic concerning the design can be done in between entering lines of the program, and during debugging.

QUESTIONS:

1. What does the computer do in the "RUN mode"?
2. How can you tell if the computer is in the "edit mode"?
3. What 3 kinds of things can you do in the edit mode?
4. If you enter a line that starts with a line number, what happens to the line?
5. If you enter a line that does not have a line number, what happens?

LESSON 18 EDIT AND RUN MODES, THE CALCULATOR

Enter: NEW

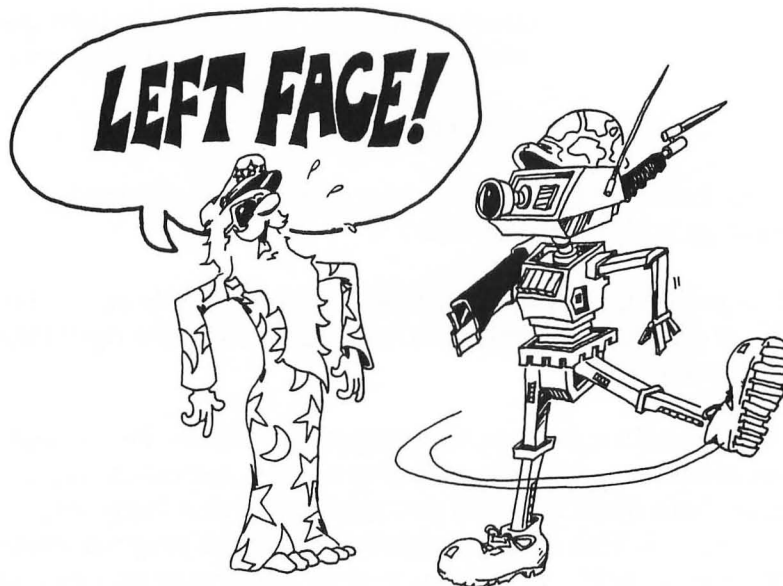
press SHIFT CLEAR

You are ready to begin the lesson.

EXECUTION AND RUNNING

We mean “execution” like the soldier executing the command “Left Face,” not “execution by firing squad.”

“Execute a program” means the same as “run a program.”



DEFERRED EXECUTION

Enter and run this program:

```
10 PRINT "HI"
```

This is the usual way to make and run programs, and is called “deferred execution.”

In “deferred execution” the computer waits until you enter the “RUN” command before executing the program.

Rule for Deferred Execution: If the line starts with a number, it is put into memory. The line becomes part of the program in the computer’s memory. The program is executed by the “RUN” command.

IMMEDIATE EXECUTION

Here is a short cut. Enter this (no line number in front):

```
PRINT "HI"
```

This time the computer printed "HI" right away, without waiting for you to enter RUN. This is called "immediate execution."

Rule for Immediate Execution: If the line does not start with a number, the computer executes the command right away (as soon as you press the RETURN key).

Try this longer example:

```
FOR I=1 TO 20:PRINT I:NEXT I:PRINT:PRINT"DONE"
```

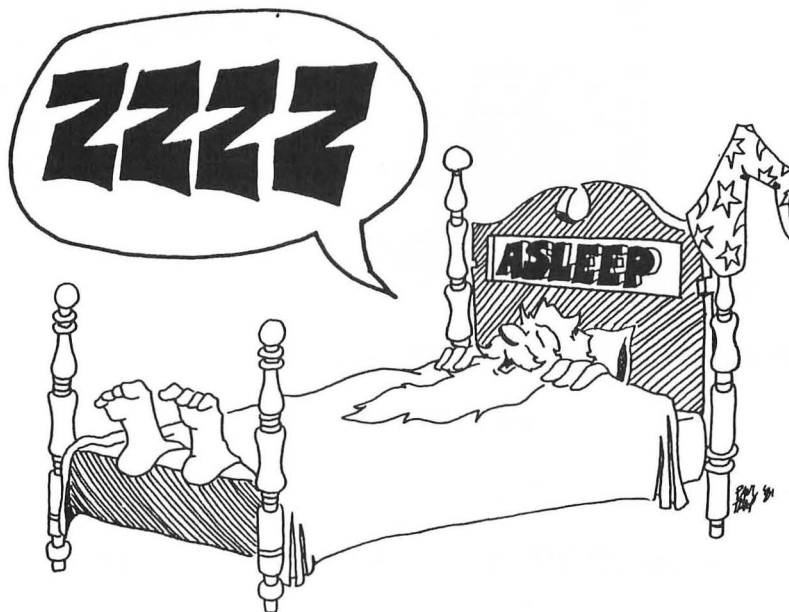
Rule: In immediate execution you can run a one line program that has several statements separated by colons.

ASLEEP OR AWAKE?

People act one way if they are awake and another way if they are asleep. They have two "operating modes."

You can tell if they are asleep because they snore. (Well, not all people snore, but to explain how computers are like people, let's pretend that all sleeping people snore.)

The computer has two operating modes too. They are called the "edit mode" and the "RUN mode."



THE EDIT MODE

Press the BREAK key.

You see READY and the cursor square. READY is called a “prompt” and says that the computer is in the “edit mode” of ATARI BASIC. The READY is the “snoring” of the computer when it is in the edit mode.

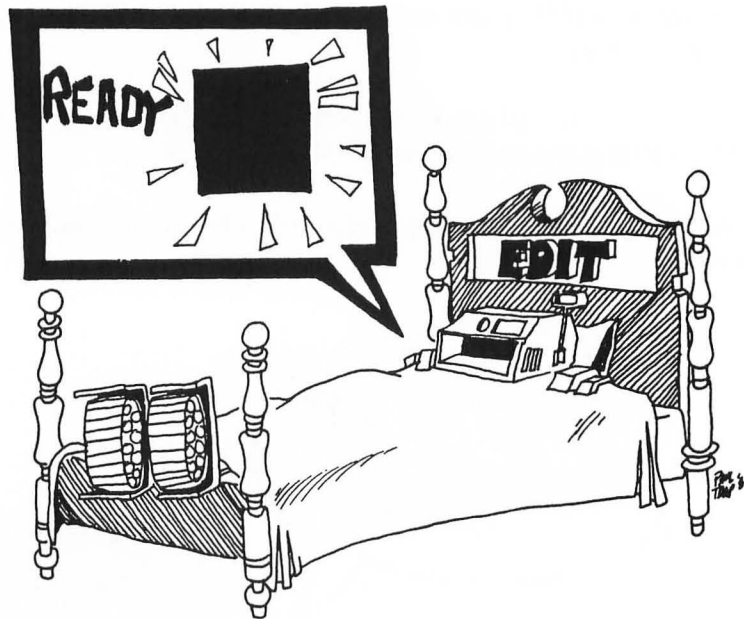
The square is called the “cursor.” It tells us that the computer is waiting for us to type something. The next letter we type will be on the screen at the cursor square.

While the computer is in the edit mode:

You can enter programs by typing lines that start with numbers.

You can use the computer like a pocket calculator. Big pocket!

You can correct errors in programs. This is called “editing” a program and gives the “edit mode” its name. Later in the book we will learn more about how to edit programs.



THE CALCULATOR

You can do arithmetic in the Edit Mode. Try this:

```
PRINT 3+7
```

The computer prints the answer “10”

Of course, you can use the PRINT shortcut of using “?” in place of “PRINT.”

```
? 3+7
```

THE RUN MODE

Enter RUN to leave the EDIT MODE and go to the RUN MODE.

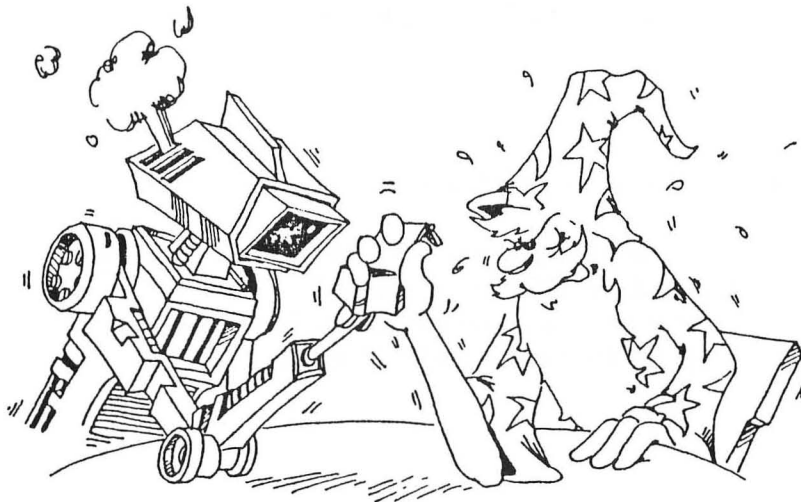
While the computer is in the run mode:

The program in memory runs.

When the program is finished, the computer automatically goes back to the edit mode.

Assignment 18:

1. Explain what "immediate execution" means. Use the computer as a calculator to do some arithmetic problems.
2. Explain what "deferred execution" means. Write a program that has several lines. In one line it prints "22 plus 67 is" and then in another line does the addition and prints the answer.
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 is done running?
6. How can you tell where the next letter you type will appear on the screen?



INSTRUCTOR NOTES 19 DATA, READ, RESTORE

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

The storing of data in DATA statements has a few confusing aspects when first confronted. You can never change any of the data in the 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 the stuff before it. (This procedure is not discussed in the lesson, and may be mentioned to the student when other ideas about DATA are well entrenched.)

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

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

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

One of the most common uses of DATA is to fill arrays with initial values.

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. Can you put numerical data and string data into the same DATA statement?
4. Can you change the items in a DATA statement while the program runs?

LESSON 19 DATA, READ, RESTORE

TWO KINDS OF DATA

There are two kinds of data in your programs:

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

```
10 REM FIRST KIND OF DATA
15 DIM P$(40)
20 PRINT"clear"
30 PRINT"YOUR PET PEEVE";:INPUT P$
40 PRINT"REALLY!"
50 PRINT"YOU DON'T LIKE"
60 PRINT P$
```

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

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

```
10 REM THE SECOND KIND OF DATA
20 PRINT"clear"
30 X=2
40 Y=3
50 PRINT X+Y
```

In this program X and Y are data stored in the program by the programmer when she wrote the program.

STORING LOTS OF DATA

It is OK 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
15 DIM D1$(10),D2$(10),D3$(10),D4$(10)
20 PRINT"clear"
30 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY
FRIDAY,SATURDAY
40 READ D1$
42 READ D2$
44 READ D3$
46 READ D4$
60 PRINT D1$,D2$
```

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

STRANGE RULES

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

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

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

Do this: Break the DATA statement into two:

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

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

IT IS POLITE TO "POINT"

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

You can't see the pointer. Just imagine it is there.

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 command, the pointer moves to the next item in the DATA list.

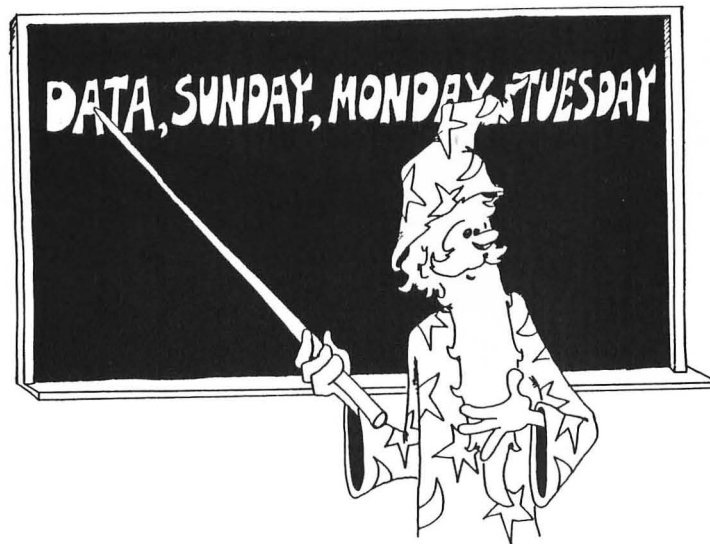
If the pointer gets to the end of one DATA statement, it automatically goes to the next DATA statement. (That is, to the DATA statement with the next higher line number.)

It doesn't matter if there are a lot of 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

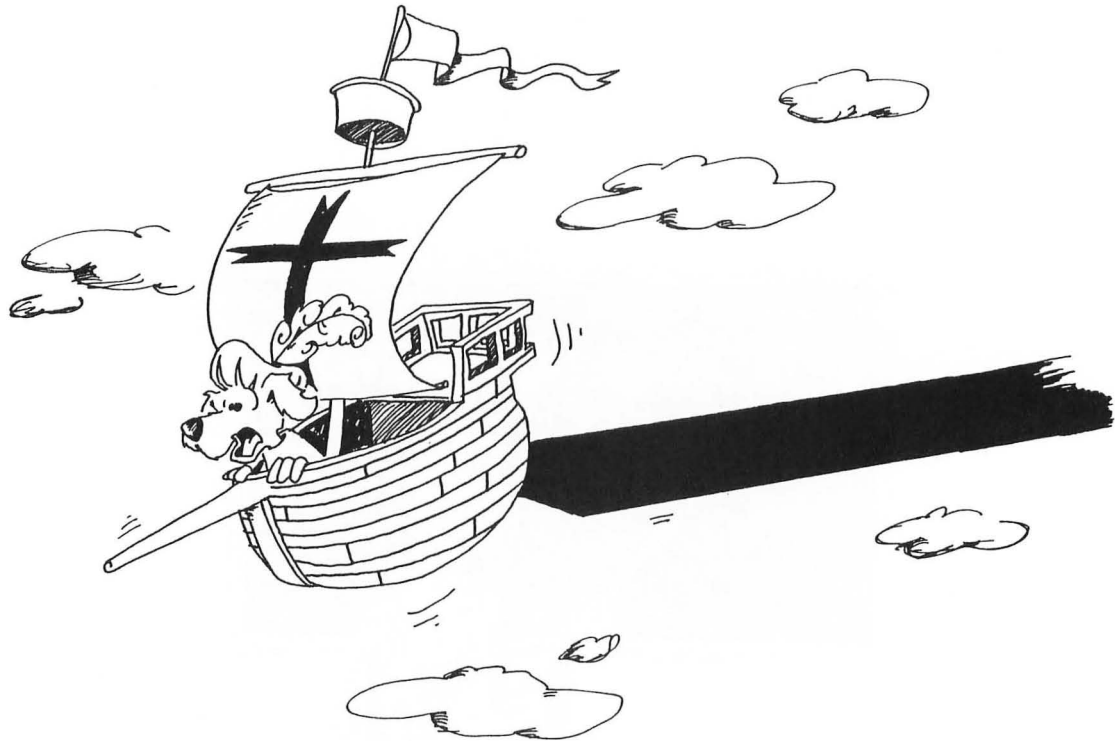
When the pointer reaches the last item in the last DATA statement in the program, there are no more items left to read. If you try to READ again, you will see an error message:

ERROR -

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 command 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 command to have the correct kind of variable to match the kind of data.

Correct:

```

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

```

Wrong:

```

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

```

OK, B\$ box holds "77"
TYPE MISMATCH ERROR

You can't put "FUZZ" into a number box.

Assignment 19:

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

INSTRUCTOR NOTES 20 SOUND

The SOUND command turns any of 4 voices on and off. You can set the pitch, loudness and “distortion” of each voice.

Musical notes use a “distortion” value of 10 (a square wave form). The other distortion numbers eliminate certain pulses from the square wave form, giving various amounts of roughness or hissing and popping.

The sound must be turned off after it is turned on.

When using sound in graphics situations, you get the most elaborate effects if you interweave the sound commands with the “move the graphics” commands.

The DATA command is useful for storing the notes in music.

QUESTIONS:

1. Which pitch numbers give deep sounds? Which give high notes?
2. How do you turn the sound off?
3. How do you make the most musical sounding note?
4. How do you make a hissing noise?
5. What number gives the loudest noise? The softest?
6. How do you make a “motor” sound?

LESSON 20 SOUND

The ATARI has 4 sound voices. All can "sing" at the same time.

They can be used in music and in other sound effects.

Examples: 30 SOUND V, P, D, L
30 SOUND 0,121,10, 8

variable	value
V for "voice"	0, 1, 2, or 3
P for "pitch"	from 1 to 255
D for "distortion"	even numbers from 0 to 14
L for "loudness"	from 0 to 15

VOICE

The voices are all alike. It does not matter which one you pick to use.

There are 4 voices so you can use 2, 3 or 4 together if you want.



PITCH

Pitch tells whether you have a "high note" or "low note." The bigger the number, the deeper the pitch. Here is a tempered scale of musical notes:

note	number
C (below middle C)	243
C #	229
D	216
D #	204
E	193
F	182
F #	172
G	162
G #	153
A	144
A #	136
B	129
C (middle C)	121
C #	115
D	108
D #	102
E	96
F	91
F #	86
G	81
G #	77
A	72
A #	68
B	64
C (above middle C)	60

These notes may be a little out of tune.

This is the reason:

only integers are allowed in the SOUND command

but: decimal numbers are needed for notes in tune

You get very high notes for pitch numbers near 10. You may not be able to hear anything for pitch number 2, 1, or 0.

DISTORTION

Change the “timber” or sound quality by changing the distortion number.

The purest musical notes come from distortion 10.

Sound effects may want distortion numbers different from 10.

Try other even numbers in the program below.

Can you make thunder?, snake hissing?, motor boat?, laser gun?

LOUDNESS

Zero turns the voice off.

Normal loudness is 8.

The loudest is 15.

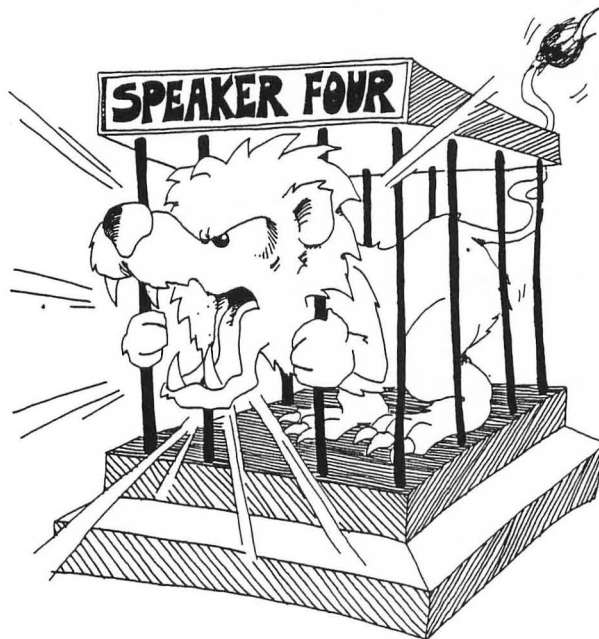
Of course, you can turn up the TV to get a louder sound.

CAREFUL! If more than one voice is used at the same time, the sum of the loudness numbers should not be bigger than 32.

Example:

```
30 SOUND 0,200,10,9
32 SOUND 1, 35,8 ,11
34 SOUND 2,155,12,9
```

This is OK because $9 + 11 + 9$ is only 29.



ONE VOICE

```
10 REM ONE VOICE
20 PRINT "VOICE ZERO"
25 PRINT "WHAT PITCH <1 TO 255>"
26 INPUT P
30 PRINT "WHAT DISTORTION <0 TO 14, EVEN>"
31 INPUT D
35 PRINT "HOW LOUD <0 TO 15>"
36 INPUT L
40 SOUND V,P,D,L
45 FOR T=1 TO 1000:NEXT T
50 GOTO 20
```

DUET

```
10 REM DUET
20 PRINT "VOICES 0 AND 1"
25 PRINT "PITCH 0":INPUT P0
30 PRINT "PITCH 1":INPUT P1
40 SOUND 0,P0,10,8
42 SOUND 1,P1,10,8
45 FOR T=1 TO 1000:NEXT T
50 GOTO 20
```

Assignment 20:

1. Make a list of sounds that each distortion number gives. For each distortion number 0, 2, 4, 6, 8, 10, 12, 14 try pitch 244, 122, 66, and 33. Write a little note to tell what the sound is like. Think of a game or program that the sound would be good for.
2. Write a program to play a short tune, like "Row, Row, Row Your Boat" or "Mary Had A Little Lamb." Use a DATA statement to store the pitch numbers.

INSTRUCTOR NOTES 21 COLOR GRAPHICS

This lesson introduces the commands GRAPHICS 3, SETCOLOR, COLOR, PLOT, and DRAWTO.

If you have a black and white TV or monitor then you can still use graphics. In fact, the drawings will be somewhat crisper.

If you have a color monitor, then the student should set up its controls for pleasing color. The pink and gold colors are sensitive to correct setting of the TV color controls.

The GRAPHICS 3 mode uses spots (pixels or PICTURE ELEMENTS) that are rather large. They are rectangles and each is the size of a letter in text mode of display.

GRAPHICS 5 and GRAPHICS 7 use smaller pixels. It will be easy for the student to use all other graphics modes after mastering GRAPHICS 3.

Drawing pictures dot by dot is quite tedious. It is a little less work when using the DRAWTO command.

In any case, use of graph paper to block out the picture first is often helpful. I recommend using a variable to designate a corner (or the center) of the drawing, with offsets from the corner for the other points and lines in the drawing. Then it is easy to move the whole figure if necessary for animation or just for correction of the composition.

QUESTIONS:

1. If you give the command GRAPHICS 3, what happens?
2. How do you "pour paint into the pots" if you want faint pink in pot 2?
3. What brush must you use for pot 2? What command "picks up the brush"?
4. How many colors are there to choose from?
5. How many colors can you use at once?
6. What range of numbers are allowed for X and Y in the command PLOT X,Y?
7. Where will this line be on the screen?

```
20 PLOT 1,5 : DRAWTO 1,20
```

LESSON 21 COLOR GRAPHICS

Before you use color graphics, you have to tell the computer “which kind” of graphics you will use. In this lesson you will use only GRAPHICS 3.

In lesson 29 you will learn how to use the rest of the color graphics modes.

PAINT POTS

The ATARI has 5 “paint pots” numbered 0 to 4.

Before using color, you must use the SETCOLOR command to “pour paint” into the pots.

SETCOLOR P,C,B

P is the pot number (0 to 4)

C is the color put into the pot (0 to 15)

B is the brightness of the color (even numbers 0 to 14)

You have 15 colors to choose from, but you can never use more than 5 colors at the same time (because you have only 5 paint pots.)

In GRAPHICS 3 you can only use pots 0, 1, 2, and 4.

0 GREY

1 GOLD

2 ORANGE

3 RED-ORANGE

4 PINK

5 PURPLE

6 PURPLE-BLUE

7 BLUE

8 BLUE

9 LIGHT-BLUE

10 TURQUOISE

11 GREEN-BLUE

12 GREEN

13 YELLOW-GREEN

14 ORANGE-GREEN

15 LIGHT ORANGE



PAINT BRUSHES

You have 5 “paint brushes” to dip into the paint pots. Each brush belongs in a certain pot. In GRAPHICS 3 these are the brushes you use.

Brush 1 in pot 0
Brush 2 in pot 1
Brush 3 in pot 2
Brush 0 in pot 4

(Sorry, the numbers don't match up!)

Pot 4 has a special use. The paint you put into pot 4 is poured directly on the screen as background color. You can also dip brush COLOR 0 into it and erase any dot you painted earlier.

paint pot

SETCOLOR 0,C,B
SETCOLOR 1,C,B
SETCOLOR 2,C,B
SETCOLOR 4,C,B

paint brush

COLOR 1
COLOR 2
COLOR 3
COLOR 0 and background

You “pick up” a brush to paint with by using the COLOR command.



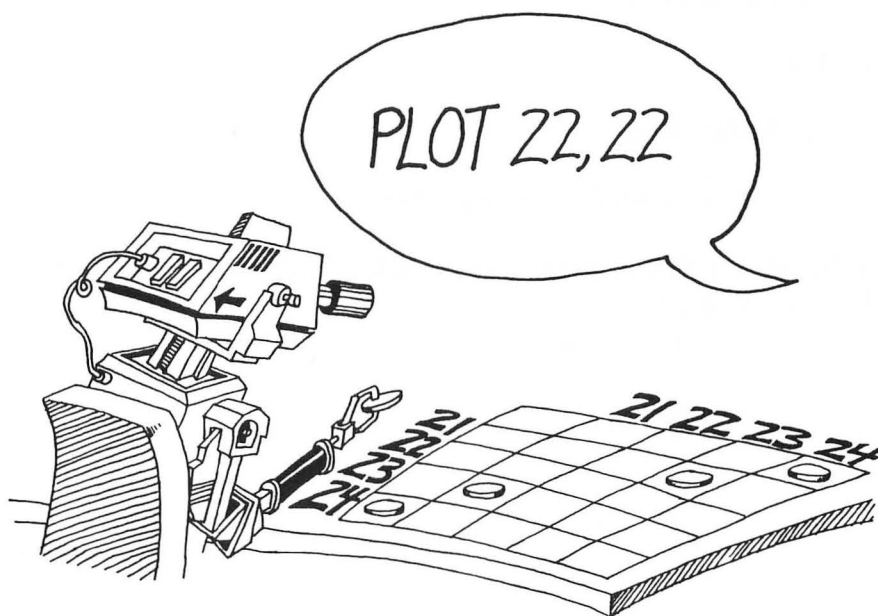
A TWO COLOR PROGRAM

Run:

```
10 REM ((( SMILE )))
15 DIM A$(1)
20 GRAPHICS 0
25 PRINT "BACKGROUND COLOR <0-15>":INPUT BC
26 PRINT "PICTURE COLOR <0-15>";INPUT C
31 GRAPHICS 3+16
32 SETCOLOR 4,BC,8
35 SETCOLOR 0,C,8
40 COLOR 1
42 PLOT 20,10
44 PLOT 21,11
46 PLOT 22,12
48 PLOT 23,12
50 PLOT 24,12
52 PLOT 25,11
54 PLOT 26,10
60 FOR T=1 TO 1000:NEXT T
90 INPUT "AGAIN? <Y/N> ";A$
95 IF A$="Y" THEN GOTO 20
99 GRAPHICS 0
```

save to tape

First, make the background pink and the smile gold and adjust your TV or monitor color controls to give the right colors.



What happens?

Line 20 Sets the screen mode to the text mode we have been using

Line 25 Gets the color for the background

Line 26 Gets the color for the smile.

Line 31 Changes the screen to "graphics 3." The "+ 16" means that the whole screen is graphics. Otherwise 4 lines of text appear at the bottom.

Line 32 Sets the background color

Line 35 Sets paint pot 0 to the chosen color

Line 40 Use brush 1. (It dips into pot zero.)

Line 42 Puts a dab of paint at 20 across and 10 down

THE PLOT COMMAND

<code>PLOT 4,13</code>	Puts a spot 4 lines down and 13 "squares" across.
------------------------	---------------------------------------------------

Rule: The command `PLOT X,Y` means put a spot on the screen at point X across and Y down. For a GRAPHICS 3 screen, X is a number or variable in the range 0 to 39. Y is in the range 0 to 23.

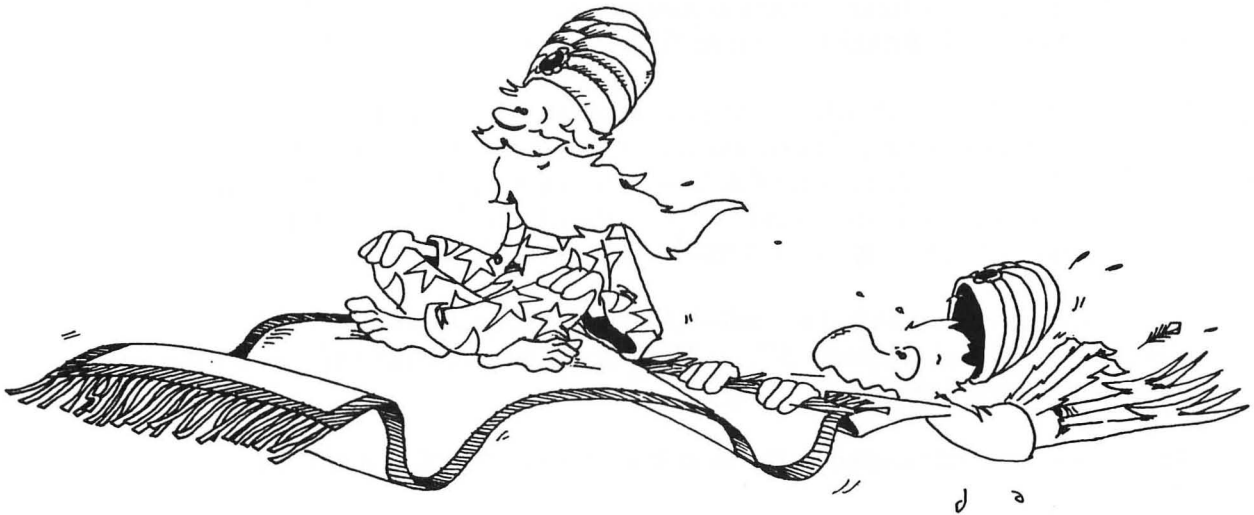
THE DRAWTO COMMAND

Change line 44 to read:

```
44 DRAWTO 3,9
```

It ruins the picture but shows how to draw lines.

Rule: The `DRAWTO X,Y` command draws a line from where the cursor is now to the point X across and Y down.



Assignment 21:

1. Add lines 27 and 28 to the SMILE program so the user can choose the brightness to go in lines 32 and 35.
2. Put eyes and a nose on the face in the "smile" program.
3. Use the PLOT and the DRAWTO commands to draw a picture in 3 colors on the screen. (Also one background color.)
4. Add to the number guessing game in lesson 13 so that a colored star shows when the correct answer is guessed. Use a timing loop so that the star shows for a few seconds before the game starts again.
5. Write a program to draw "Sinbad's Magic Rug." Let the user choose which colors to put in the rug. Then draw a pattern on the screen.

INSTRUCTOR NOTES 22 ASCII CODE

This lesson treats the ASCII code for characters, and the functions ASC() and CHR\$() that change characters to numbers and vice versa.

The ASCII code consists of numbers from 0 through 127. Each upper and lower case letter, digit and punctuation mark is assigned an ASCII number. Also included are numbers for certain functions such as "line feed" and "bell" that teletypes use.

The ASCII code is primarily intended to standardize signals between hardware pieces such as computers with printers, terminals, other computers, etc. But within programs the ASCII numbers also are useful. The letters are numbered in increasing order and so the ASCII numbers are useful in alphabetizing routines. The numerical digits are also in order, and the punctuation marks also have ASCII numbers.

The ATARI uses a modified set called ATASCII. It excludes the "teletype" codes but includes graphics characters. ATASCII numbers between 128 and 255 are the inverses of those between 0 and 127.

The GET command treated in the next lesson uses ATASCII numbers.

QUESTIONS:

1. Does ASC(S\$) return a string or a number for its value?
2. Does ASC(S\$) have a string or a number for its argument?
3. Same two questions for CHR\$(N).
4. Which letter has the larger ASCII code number, B or W?
5. Do you know the ASCII code for the character "1"? Is it the number 1?

LESSON 22 ASCII CODE

NUMBERING THE LETTERS IN THE ALPHABET

“That is easy,” you say. “A is 1, B is 2, C is 3 ...”

Well, for some strange reason it goes like this: A is 65, B is 66, C is 67

These numbers are called the ASCII code of the characters. ASCII is pronounced “ask-key.”

The punctuation marks and number digits have ASCII code numbers too.



ASCII AND ATASCII CHARACTERS

The ASCII numbers are used by all computer brands.

The ATARI uses a special kind of ASCII called ATASCII.

ATASCII is the same as ASCII for all the letters, numbers and punctuation. These are the numbers 32 to 127 (except 96).

ATASCII is different from ASCII in numbers 0 to 31 which are graphics characters.

The ATASCII numbers 128 to 255 are usually the inverses of the characters 0 to 127.

CHR\$() MAKES NUMBERS INTO CHARACTERS

Use CHR\$() to change ATASCII code numbers into a string holding one character.

Run:

```
10 REM --- DISPLAY ALL THE ATASCII NUMBERS --
11 REM
20 PRINT "clear"
30 FOR I=0 TO 255
40 PRINT I, CHR$(I)
50 FOR T=1 TO 500:NEXT T
60 NEXT I
```

As this program runs, you see the number and its ATASCII character. Something funny happens for numbers 27 to 31, 125 to 127, 155 to 159 and 253 to 255.

These numbers are control characters. Look at the list of ATASCII numbers in your ATARI 400/800 Basic Reference Manual to see what they do.

Number 253 makes the buzzer sound.

ASC() CHANGES CHARACTERS INTO NUMBERS

Use the ASC() function to change characters into ASCII numbers.

Run:

```
10 REM --- WHAT NUMBER IS THIS KEY? ---
15 DIM C$(1)
17 OPEN #1,4,0,"K: "
20 PRINT "clear"
25 PRINT "PRESS KEYS TO SEE ASCII NUMBER"
30 GET #1,C
35 C$=CHR$(C)
40 PRINT C$;"      ";ASC(C$)
50 GO TO 30
```

Try out some letters, digits, and punctuation. Try also the RETURN key and other keys.

Try this: Hold down CTRL and press keys. Hold down SHIFT and press keys.

Press BREAK to end the program.

GET and OPEN are explained in the next lesson.



ALPHABETICAL LIST

What good are the ASCII numbers? Well, they can help in making alphabetical lists.

```
Run:      10 REM ALPHABETIZE
          15 DIM A$(1), B$(1)
          20 PRINT
          30 PRINT"GIVE ME A LETTER: ";INPUT A$
          35 PRINT
          40 PRINT"GIVE ME ANOTHER: "; INPUT B$
          45 A=ASC(A$):B=ASC(B$)
          47 REM PUT IN ALPHABETICAL ORDER BY
          48 REM SEEING WHICH HAS THE LOWER ASCII NUMBER
          50 IF A>B THEN X=A:A=B:B=X:REM SWAP THEM
          55 PRINT
          60 PRINT"HERE THEY ARE IN ALPHABETICAL ORDER"
          65 PRINT:PRINT CHR$(A);"    ";CHR$(B)
```

Look at these two functions: `ASC()` and `CHR$()`.

`ASC()` gives you the ASCII number for the FIRST character in the string.

`CHR$()` does the reverse. It gives you the character belonging to each ASCII number.

Assignment 22:

1. Write a program which asks for a word. Then it rearranges all the letters in alphabetical order.
2. Write a program that speaks "double dutch." It asks for a sentence then removes all the vowels and prints it out.

INSTRUCTOR NOTES 23 SECRET WRITING AND THE GET COMMAND

This lesson concerns the GET command.

GET is a method of requesting a single character from the keyboard. The computer waits until the keystroke is made.

There is no screen display at all. No prompt or cursor is displayed while waiting and the keystroke, when made, is not echoed to the screen.

The utility of the GET command lies just in this fact. For example, a requested word may be received with a series of GET's without displaying it to bystanders.

Another advantage over INPUT is that no RETURN keypressing is required. This makes GET useful in "user friendly" programming.

To use GET you must first OPEN the keyboard as an input device. Then the GET command obtains a keystroke as an ATASCII number. Use CHR\$() to convert the number to a character.

QUESTIONS:

1. Compare INPUT and GET. One gets one letter at a time, the other gets whole words and sentences. One has a cursor, the other not. One prints on the screen, the other not. One needs the RETURN key, the other not. Which does which?

LESSON 23 SECRET WRITING AND THE GET COMMAND

THE INPUT COMMAND

Examples:

```
10 INPUT A$  
10 INPUT N  
10 INPUT NAME$,AGE,DAY,MONTH$,YEAR
```

The “?” and cursor square show that you should INPUT a letter, word, sentence or number.

THE GET COMMAND AND SECRET WRITING

The GET command is different from INPUT. It gets a single character from the keyboard.

Nothing shows on the screen:

- no question mark will show on the screen
- no cursor will show
- what you type will not show.

You do not need to press the RETURN key afterward. The computer waits for a key to be pressed then immediately goes on with the program.



Use GET in guessing games for entering the word or number to be guessed without the other player being able to see it.

Run this program:

```
10 REM -----GET-----
15 DIM K$(1)
17 OPEN #1,4,0,"K:"
20 PRINT "clear"
30 PRINT "PRESS ANY KEY"
40 GET #1,K
45 K$=CHR$(K)
47 FOR T=1 TO 500:NEXT T
50 PRINT "THE KEY YOU PRESSED WAS ";K$
55 FOR T=1 TO 500:NEXT T
99 GOTO 20
```

Line 17 is the OPEN command. OPEN tells the computer to listen to the keyboard under a different name: "I/O device #1."

Line 40 is the GET command. GET #1,K says: "listen to I/O device number 1 and put what he says into variable box K."

The number the keyboard sends to box K is the ATASCII number of the key that was pressed.

MAKING WORDS OUT OF LETTERS

The GET command gets one letter at a time. To make words, glue the strings.

```
10 REM GET A WORD
15 DIM W$(40),L$(1)
17 OPEN #1,4,0,"K:"
20 PRINT "clear"
30 PRINT"TYPE A WORD.  END IT WITH A PERIOD."
35 W$="": REM WORD STRING IS EMPTY
39 I=1
40 GET #1,Q:L$=CHR$(Q):REM GET A LETTER
50 IF L$="." THEN GOTO 80:REM TO TEST FOR END
60 W$(I)=L$:I=I+1:REM ADD LETTER TO END OF WORD
65 GOTO 40: REM TO GET ANOTHER LETTER
80 REM WORD IS FINISHED
85 PRINT W$
```

How does the computer know when the word is all typed in? It sees a period at the end of the word. Line 50 tests for the period and ends the word when it finds the period. Line 60 glues the letters together to make a longer string. Lesson 26 tells more about gluing strings.

Assignment 23:

1. Write a program that has a "menu" for the user to choose from. The user makes her choice by typing a single letter. Use GET to get the letter.

Example:

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

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."). Then clear the screen and print the sentence. Use GET so no player can see the words of the others. 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 command is also treated here because the program will usually have its subroutines at high line numbers and so must END in the middle line numbers.

Subroutines are useful not only in long programs but in short ones where “chunking” the task into sections leads to clarity.

One of the hardest habits to form in some students (and even some professionals) is to impose structure on the program. Structuring has gone by many names such as “structured programming” and “top down programming” and uses various techniques to discipline the programmer.

Call the student’s attention to ways that structuring can be done and the advantages in clarity of thought and ease of programming that result. In this book, writing good REM statements and using modular construction in the program are the main techniques offered.

GOSUB was put in BASIC for making modules. This lesson shows modular construction by example in the outline to the hangman program.

QUESTIONS:

1. What happens when the command END is executed?
2. How is GOSUB different from GOTO?
3. What happens when RETURN is executed?
4. If RETURN is executed before a GOSUB, what happens?
5. What does “call the subroutine” mean?
6. How many END commands are you allowed to put in one program?
7. Why do you want to have subroutines in your programs?

LESSON 24 PRETTY PROGRAMS, GOSUB, RETURN, END

Run this program then save it to disk:

```
100 REM ----- MAIN PROGRAM
101 REM
110 PRINT "READY TO GO TO THE SUBROUTINE"
120 GOSUB 200
130 PRINT "BACK FROM THE SUBROUTINE"
133 PRINT
135 PRINT "GO TO THE SUBROUTINE AGAIN"
140 GOSUB 200
150 PRINT "BACK AGAIN"
190 END
199 REM
200 REM ----- SUBROUTINE
201 REM
210 PRINT "IN THE SUBROUTINE"
215 FOR T=1 TO 1000:NEXT T
217 PRINT "buzz"
290 RETURN
```

This is the skeleton of a long program. The main program starts at line 100 and ends at line 190.

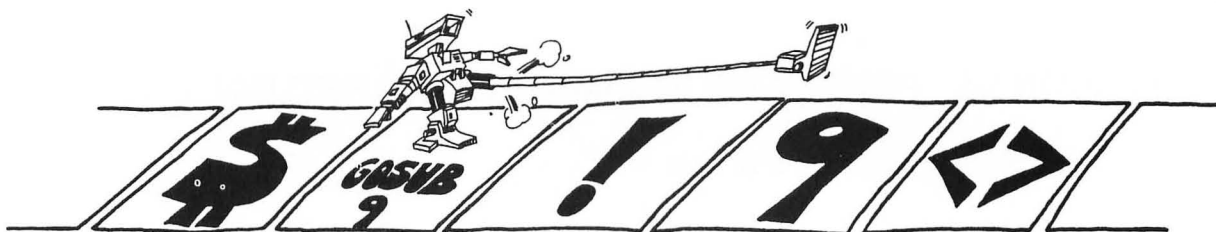
Where there are PRINT commands, you may put many more program lines.

The END command in line 190 tells the computer that the program is over. The computer goes back to the edit mode.

Line 120 and 140 "call the subroutine." This means the computer goes to the lines in the subroutine, does them, and then comes back.

The GOSUB 200 command is like a GOTO 200 command except that the computer remembers where it came from so that it can go back there again.

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



WHAT GOOD IS A SUBROUTINE?

In a short program, not much.

In a long program, it does two things:

1. It saves you work and saves space in memory. You do not have to repeat the same program lines in different parts of the program.
2. It makes the program easier to understand and faster to write and debug.

THE END COMMAND

The program may have zero, one, or many END commands.

Rule: The END command tells the computer to stop running and go back to the Edit Mode.

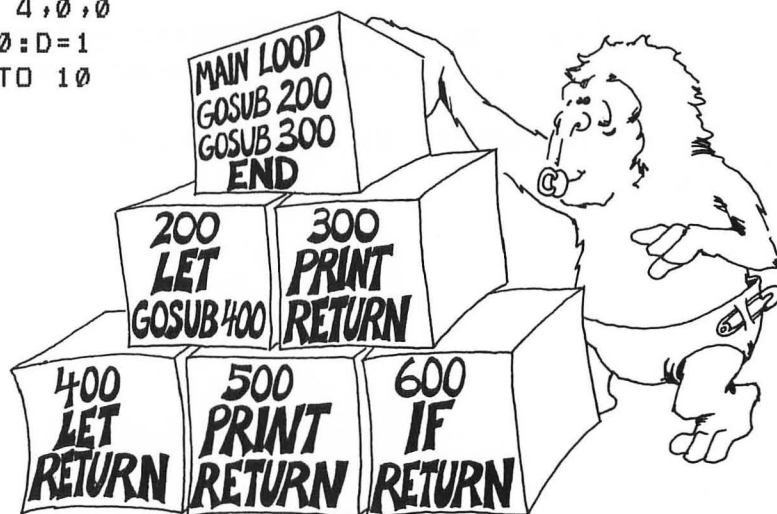
That is really all it does. You can put an END command anywhere in the program—for example, after THEN in an IF statement.

MOVING PICTURES

```

10 REM ??? JUMPING J ???
12 PRINT "clear"
13 GRAPHICS 3+16
14 SETCOLOR 0,6,8
18 SETCOLOR 4,0,0
20 X=15:Y=10:D=1
25 FOR J=1 TO 10

```



```

26 FOR I=1 TO 10
30 COLOR 1:GOSUB 100:REM DRAW
35 COLOR 0:GOSUB 100:REM ERASE
45 Y=Y-D
50 NEXT I
55 D=-D
60 NEXT J
99 END
100 REM
101 REM  DRAW THE J
102 REM
110 PLOT X,Y:DRAWTO X+6,Y
115 PLOT X+3,Y+1:DRAWTO X+3,Y+7
120 PLOT X,Y+7:DRAWTO X+2,Y+7
125 PLOT X,Y+6
190 RETURN

```

The picture is the letter "J." The subroutine starting in line 100 draws the "J." Before you GOSUB 100 you pick what color you want the "J" to be, using a COLOR command. Look at line 30 and at line 35. If you pick color number two, then the subroutine erases a "J" from that spot.

The subroutine draws the "J" with its upper left corner at the spot X,Y on the screen. When you change X or Y (or both) the "J" will be drawn in a different spot.

The letter "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 10 pictures have been drawn.

Line 45 says that each picture will be drawn at the spot where Y is larger than the last Y by the amount D.

Assignment 24A:

1. Enter the JUMPING J program and run it. Then make these changes:

Change the subroutine so it prints your own initial.

Change the color of your initial to blue.

Change the "jumping" to "sliding" (so the J moves horizontally instead of vertically).

Change the starting point to the lower right hand corner instead of the middle of the screen.

Change the distance the slide goes to 12 steps instead of 10.

Change the size of each step from 1 to 2.

Change the "sliding" so it slides uphill. Use

$X=X+D$: $Y=Y-D$

Change the program so the initial changes color from grey (color 0) through all the colors to orange (color 15) as it jumps.

HOW TO WRITE A LONG PROGRAM

Let's write a hangman game. This is a word guessing game where you draw another part of the hanging person each time you make a wrong guess for a letter.

First make an outline. You can do this on paper or right on the screen. If you have trouble deciding what to do, then just play through a game on paper and keep track of what happens. Then the program has to do the same things.

The outline of could be:

```
10 REM --- HANGMAN GAME ---
200 REM INSTRUCTIONS
300 REM GET THE WORD TO GUESS
400 REM MAKE A GUESS
500 REM TEST IF RIGHT
600 REM ADD TO THE DRAWING
700 REM TEST IF GAME IS OVER
800 REM END GAME MESSAGE
```

After making this outline, I filled in more details.

```
10 REM --- HANGMAN GAME ---
99 REM
100 REM MAIN LOOP
101 REM
115 DIM Y$(1)
120 PRINT" NEED INSTRUCTIONS? <Y/N> "
121 INPUT Y$
122 IF Y$="Y" THEN GOSUB 200
130 GOSUB 300:REM GET WORD
132 STOP
135 GOSUB 400:REM MAKE GUESS
140 GOSUB 500:REM TEST GUESS
145 GOSUB 700:REM TEST IF GAME IS OVER
190 GOTO 135:REM MAKE ANOTHER GUESS
199 REM 200 REM INSTRUCTIONS
--- write the instructions last
290 RETURN
299 REM
300 REM GET THE WORD TO GUESS
--- use INPUT to get a word from player 1
--- draw dashes for the letters to be guessed
390 RETURN
399 REM
400 REM MAKE A GUESS
--- player 2 guesses a letter
490 RETURN
499 REM
```

```

500 REM TEST IF GUESS IS RIGHT
---  if wrong, GOSUB 600:REM draw hangman part
---  if right, GOSUB 700:REM see if game is over
590 RETURN
599 REM
600 REM ADD TO THE DRAWING
---  add to the hangman drawing
---  test if drawing is done
---  if so, then GOSUB 800
690 RETURN
699 REM
700 REM TEST IF GAME IS OVER
---  see if all letters have been guessed
---  if yes, GOSUB 900
790 RETURN
799 REM
800 REM END GAME MESSAGE
---  message for when guesser loses
890 RETURN
899 REM
900 REM END OF GAME MESSAGE
---  message for when guesser wins
990 RETURN

```

Things are getting a little mixed up in my mind on how to end the game. So I will leave that to later, and start writing and testing the first part of the program. I put a STOP in line 132 so that only the first subroutine will be run. I will start by writing the subroutine at 300, GET A WORD.

Assignment 24B:

1. Write a short program that uses subroutines. It doesn't have to do anything useful just print some silly things. In it put three subroutines:
 Call one of them twice from the main program.
 Call one of them from another of the subroutines.
 Call one of them from an IF statement.
2. Write a program that writes your 3 initials on the screen, each one a different color. Then make them jump up and down one at a time!
3. Finish the hangman game . This is a long project, and you may want to do part of it now and SAVE it to tape and finish the game later.

ADVANCED PROGRAMMING

INSTRUCTOR NOTES 25 KEYBOARD, ON ... GOTO

The byte at 764 (decimal) holds a “raw” number from the keyboard. Its usefulness is explored in this lesson.

The ON ... GOTO command is explained.

The keyboard puts a number into byte 764 when a key is pressed. The number is latched in, and further keypresses will not change the number. You can clear the box for the next keypress by POKEing 255 into it. The box records the combination SHIFT + KEY by adding 64 to the number from KEY alone.

Likewise, CTRL + KEY yields 128 plus the number from KEY alone. The advantage to this method of reading the keyboard is that the program does not “hang” until the key is pressed, as it would with GET or INPUT. The keyboard is read on the fly.

One disadvantage is that the number assigned to each key is not related to any more universal code, such as ASCII.

The ON ... GOTO command is a primitive example of the CASE OF construction. ON ... GOSUB is also supported in ATARI BASIC.

QUESTIONS:

1. What number is on the front of the keyboard's box?
2. Why do you have to empty the keyboard's box?
3. How do you find out what number belongs to each key?
4. What happens in ON V GOTO 200,300,400 if:

V = 0 _____
V = 1 _____
V = 3 _____
V = 8 _____

LESSON 25 KEYBOARD, ON ... GOTO ...

GAMES AND THE KEYBOARD

The GET command makes the computer wait for you to press a key. The program stops running until you press a key.

There is another way to get a keystroke from the keyboard that does not make the computer wait. It is used in action games.

```
2 GOTO 1000:REM SNAKE
100 REM MAIN LOOP
105 DR=PEEK(AR)
106 POKE AR, 255
110 IF DR=R THEN D=D-1:IF D=0 THEN D=4
111 IF DR=LL THEN D=D+1:IF D=5 THEN D=1
113 FOR T=1 TO 50:NEXT T
115 ON D GOTO 120,122,124,126
120 Y=Y-1:GOTO 130
122 X=X-1:GOTO 130
124 Y=Y+1:GOTO 130
126 X=X+1
130 COLOR 3:PLOT X,Y
140 A=B:B=C:C=E:E=F:F=G:G=X
142 L=M:M=N:N=O:O=P:P=Q:Q=Y
145 COLOR 2:PLOT A,L
199 GOTO 100
999 END
1000 REM
1001 REM ----- SNAKE -----
1002 REM
1010 REM BY E. H. CARLSON
1011 REM
1020 GOSUB 3000
2000 GRAPHICS 3+16
2010 SETCOLOR 0,8,8 :REM BORDER
2020 SETCOLOR 4,0,4 :REM BACKGROUND
2025 SETCOLOR 1,0,4 :REM ERASE
2027 SETCOLOR 2,12,8:REM SNAKE
2030 COLOR 1
2040 PLOT 0,0
2042 DRAWTO 0,23
2044 DRAWTO 39,23
2046 DRAWTO 39,0
2048 DRAWTO 0,0
2100 LL=6 :REM LEFT ARROW
2102 R =7 :REM RIGHT ARROW
2105 AR=764 :REM KEYBOARD'S BOX
```

```

2108 D=1          :REM START DIRECTION
2110 X=20:Y=12    :REM START POSITION
2115 A=X:B=X:C=X:E=X:F=X:G=X
2116 L=Y:M=Y:N=Y:O=Y:P=Y:Q=Y
2999 GOTO 100
3000 REM INSTRUCTIONS
3020 PRINT "TURN LEFT: LEFT CURSOR KEY"
3030 PRINT "TURN RIGHT: RIGHT CURSOR KEY"
3999 RETURN

```

THE KEYBOARD'S BOX

When a key is pressed, the computer puts a number in a special box with the name "764" on the front.

To look into the box, use the PEEK command. The SNAKE program used line:

```
105 DR=PEEK(AR)
```

where AR=764 (look at line 2105).

```
105 DR PEEK 764
```

would have been easier, but

the program runs a little faster when variables are used in PEEK, instead of numerical constants.

Line 105 looks into the keyboard's box to see if a key has been pressed since the last time the box was emptied.

If so, the number of the key is taken from the keyboard's box and put into box DR.

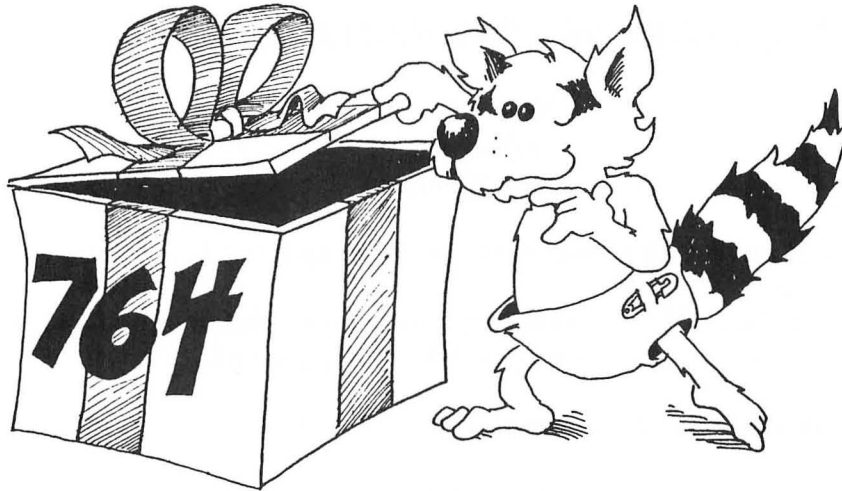
These keyboard numbers are not ATASCII numbers. They are special numbers that are used by the ATARI computers.

Line 110 asks if the key was the right arrow (number 7, see line 2102).

What does line 111 do? _____

Line 106 tells the computer to empty the keyboard's box. ("Empty" is the number 255.) You have to empty the box after each use so that it is ready to "catch" another keypress.

```
106 POKE AR,255
```



LOOKING INTO THE BOX

Try this:

```
10 V=PEEK(764)
20 PRINT V
30 GOTO 10
```

Press each key. See what number you get.

Hold down SHIFT or CTRL while you press a key. SHIFT adds 64 to the number. CTRL adds 128. THE ON...GOTO COMMAND

```
115 ON D GOTO 120,122,124,126
```

This means that

if D is	1	GOTO	120
	2		122
	3		124
	4		126
if D is something else		GOTO	the next line

After the GOTO, you can put one, two, or as many numbers as you want. Each number is the same as the number of a line somewhere in the program.

Assignment 25:

1. Make a table of what numbers show up in the keyboard's box when you press a key. You want this table when you make game programs.
2. Write a program that uses GET to get a letter A to C to use in a menu. Change the letter to a number 1 to 3. Then use the ON...GOTO command to pick which menu item to do.

INSTRUCTOR NOTES 26 SNIPPING AND GLUING STRINGS

In this lesson:

The LEN function
substrings
concatenation

With these, one can cut up strings and glue them back together in any order.

Remember that the DIM statement only gives the maximum size of a string. If you try to stuff a bigger string in, it is chopped off at the right end.

Substrings can have one or two "arguments" inside the ().

If there is one number, it means the substring from the numbered character to the end.

If there are two numbers, it means the substring is all the characters from the first number to the last, inclusive.

Examples:

10 DIM G\$(7)	
12 G\$="123456789"	only 1234567 gets in the box
13 ? G\$	prints 1234567
15 ? G\$(5)	prints 567
20 ? G\$(2,4)	prints 234
25 ? G\$(1)	prints 1234567, the same as G\$

Concatenation: There are a large number of legal cases. The clearest case is when you open up a hole in the G\$ that is the same size as the slice of H\$ you wish to put in.

```
20 G$(3,5)=H$(8,10)
```

Other cases are treated in the lesson. Use LEN to keep track of how long the strings are.

QUESTIONS:

1. How do you save "STAR" from B\$="STARS AND STRIPES"?
2. How do you save "AND"?
3. If you want to count the number of characters in the string PQ\$, what function do you use? What argument?
4. How would you change "2" to "5" in the string:

```
D$="TAKE 2 MINUTES"?
```

5. Write a short program that takes the string

```
C$="COMPUTER" and snips and reglues it into
```

```
K$='PUTERCOM'
```

LESSON 26 SNIPPING AND GLUING STRINGS

HOW LONG IS THE STRING?

Run:

```
10 REM --- LONG ROPE ---
15 DIM N$(8)
20 PRINT "clear"
30 PRINT "GIVE ME A STRING: "; INPUT N$
40 L=LEN(N$)
50 PRINT "THE STRING: '";N$;"' "
55 PRINT:PRINT "IS ";L;" CHARACTERS LONG"
```

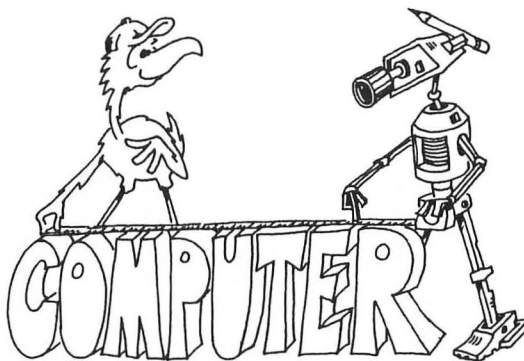
First time answer: "HUDSON"

Second time answer: "MISSISSIPPI"

The DIM command made room for 8 letters.

When you INPUT "HUDSON," you put 6 letters in.

When you INPUT "MISSISSIPPI" the string filled up at 10 letters and threw the "PPI" away.



SNIPPING STRINGS

There are two ways to snip a piece off a string. The piece is called a "substring."

1. Cut it off the right end of the string.
2. Cut it out of the middle of the string.
3. (If you want to snip it off the left end, use the rule for cutting it from the middle.)

Run:

```
10 REM >>> SCISSORS >>>
15 DIM N$(40)
20 PRINT "clear"
30 N$="123456789"
40 PRINT N$(3)
50 PRINT N$(3,5)
```

Line 40 counts 3 letters in from the left then snips and keeps the right end, including the third letter.

Rule: If there is one number inside the ()

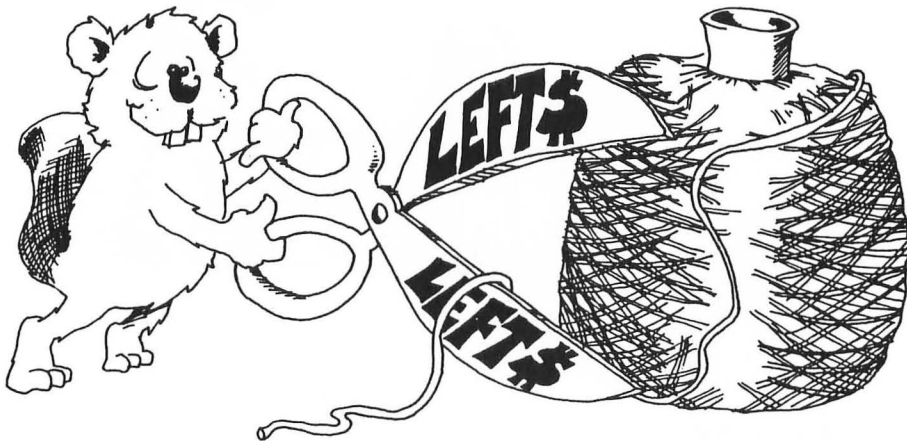
1. It tells you to count that number of letters from the left.
2. Take the letter you land on and all the rest to the right end of the string.

Notice that N\$(1) means exactly the same as N\$.

Line 50 cuts a piece out of the middle of W\$, starting from the third character and going to the fifth.

Rule: The two numbers inside the () of a string tell you:

1. The number of the first letter you want.
2. The number of the last letter you want.
3. Take the first, the last, and all in between.



You can cut just one letter out if you want.

Run:

```
10 REM --- SNIPPING ---
12 PRINT "clear"
15 DIM W$(40)
20 PRINT "GIVE ME A WORD":INPUT W$
25 L=LEN(W$)
30 PRINT "WHICH LETTER DO YOU WANT?"
31 PRINT "(GIVE ME A NUMBER)"
35 INPUT N
40 IF L<N THEN PRINT "TOO BIG":GOTO 30
45 PRINT W$(N,N)
```

Add these and run:

```
30 PRINT "HOW MANY LETTERS DO YOU WANT?"
32 INPUT N
33 PRINT "STARTING WHERE?"
34 INPUT ST
35 ND=ST+N-1
40 IF ND>L THEN PRINT "TOO MUCH":GOTO 30
45 PRINT W$(ST,ND)
```

GLUING STRINGS

Here is how to glue the substring snips back together.

The real name for “gluing” is “concatenation.”

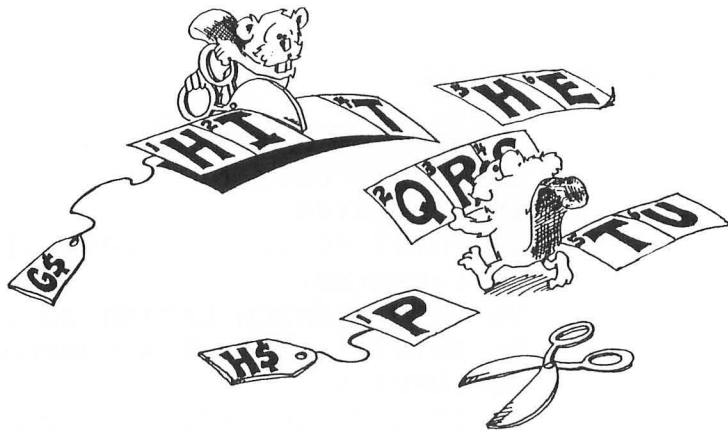
Concatenation means “make a chain.” Maybe we should call them “chains” instead of “strings.”

```
10 REM >>> SCISSORS AND GLUE >>>
15 DIM G$(10), H$(5)
20 G$="123456789"
30 H$="ABCDE"
40 G$(3,5)=H$(2,4)
50 PRINT G$
```

Line 40 says: “Open a hole in G\$ from character 3 to character 5.” These just happen to be the numbers 3,4,5 put in G\$ in line 20.

Then snip a piece out of H\$. It will be the letters BCD.

Then fill the hole in G\$ with the substring snipped out of H\$.



Experiment by changing line 40. Try these cases:

- | | |
|----------------------|-----------------------------------------------------------------------------------|
| 40 G\$(3,4)=H\$(2,4) | The hole in G\$ is too small, and part of the H\$ snip is thrown away. |
| 40 G\$(3,6)=H\$(2,4) | The hole in G\$ is too big, and H\$ only covers part of the hole. |
| 40 G\$ =H\$(2,4) | The old G\$ string is thrown away. The new G\$ is the snip of H\$. |
| 40 G\$(1) =H\$(2,4) | Same as above. |
| 40 G\$(3) =H\$(2,4) | The new G\$ string is half and half. Front is a snip of G\$, back is snip of H\$. |

LOOK MA, NO SPACES

```
Enter: 10 REM >>> NO SPACES >>>
11 REM
15 DIM S$(40),L$(1),T$(40)
20 PRINT "clear":PRINT
30 PRINT"GIVE ME A LONG SENTENCE":PRINT
35 INPUT S$
40 L=LEN(S$)
45 T$="":N=1
50 FOR I=1 TO L:REM LOOK AT EACH LETTER
60 L$=S$(I,I)
70 IF L$<>" " THEN T$(N)=L$:N=N+1
90 NEXT I
92 PRINT:PRINT T$
95 PRINT:PRINT:PRINT
```

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

Line 70 glues the character into T\$ if it is not a space.



Assignment 26:

1. Write a secret cipher making program. You give it a sentence and it finds how long it is. Then it switches the first letter with the second, third with the fourth, etc. Example:

THIS IS AN ATARI. becomes:

HTSII SNAA ATIR .

2. Write a question answering program. You give it a question starting with a verb and it reverses verb and noun to answer the question. Example:

ARE YOU A TURKEY?

YOU ARE A TURKEY.

3. Write a PIG LATIN program. It asks for a word. Then it takes all of 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 only adds LAY. Examples:

SLAM becomes AMSLAY

ATARI becomes ATARILAY

INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS

This lesson treats two functions, STR\$ and VAL and reviews functions.

STR\$ takes a number and makes a string that represents it.

VAL does just the opposite, taking a string and making a numerical value from it.

If VAL is given a string that cannot be made into a number, it issues an ERROR - 18.

This interconvertability of the two variable types adds great flexibility to the treatment of numbers in programs.

Functions and their arguments are summarized in the lesson. The notion that a function "returns a value" is treated.

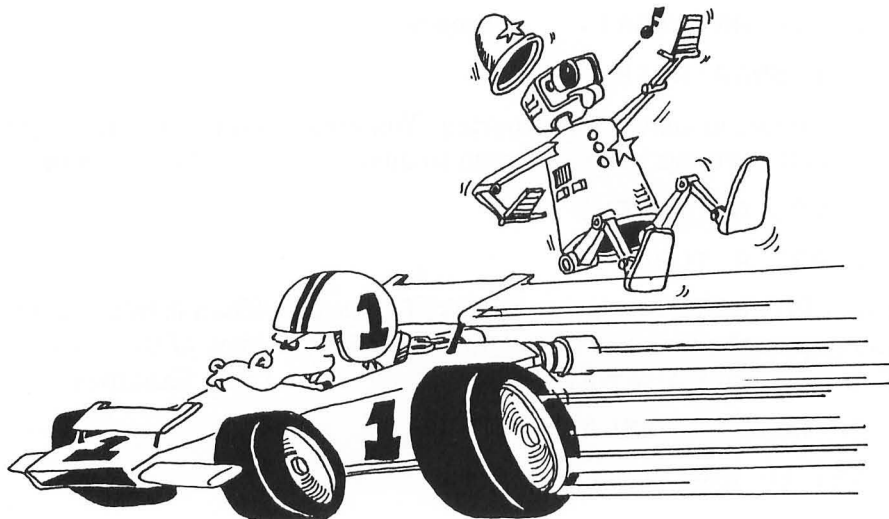
QUESTIONS:

1. If your number "marches" too quickly in the program of assignment 27, how do you slow it down?
2. Your program has the line:

```
20 S$= "GEORGE WASHINGTON WAS BORN IN 1732."
```

Write a few lines to answer the question "How long ago was Washington born? (You need to get the birthdate out of 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. What is an "argument" of a function?
5. Where in the line do commands always go? Can you put a function at the start of a line?



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 MAKING STRINGS INTO NUMBERS
               15 DIM L$(3), M$(3)
               20 PRINT "clear"
               30 L$="123"
               40 M$="789"
               50 L=VAL(L$)
               60 M=VAL(M$)
               70 PRINT L
               72 PRINT M
               74 PRINT "---"
               76 PRINT L+M
```

VAL stands for "value." It changes what is in the string to a number, if it can.

MAKING NUMBERS INTO STRINGS

```
Run:          10 REM MAKING NUMBERS INTO STRINGS
               11 REM
               15 DIM N$(10)
               20 PRINT "clear"
               25 PRINT"GIVE ME A NUMBER ":INPUT NB
               30 N$=STR$(NB)
               37 PRINT
               60 PRINT"HERE IS ITS SECOND DIGIT"
               65 PRINT:PRINT N$(2,2)
```

STR\$() makes a number into a string.

FUNCTIONS HAVE ARGUMENTS BUT DON'T FIGHT

In this book we use these functions:

RND(), INT(), LEN(), VAL(), STR\$(), ASC(), CHR\$()

Rules about functions:

1. Functions always have () with an "argument" in them. Example:

ASC(D\$)	ASC is the function
D	is the argument

The argument may be a number or a string.

The argument may be a constant, variable, expression or another function.

LEN("GOAT")	argument is a string constant
LEN(M\$)	argument is a string variable
STR\$(1982)	argument is a numerical constant
STR\$(DATE)	argument is a numerical variable
STR\$(3+2*X)	argument is a numerical expression
LEN(STR\$(DATE))	argument of LEN is a function

2. A "function" is not a "command." It cannot begin a statement.

right: 10 D=LEN\$(CS\$)

wrong: 10 LEN(CS\$)=5

3. A function acts just like a number or a string. We say the function "returns a value." The value can be put into a box or printed just like any other number or string.

```
10 PRINT INT(3.14)
15 W$ = STR$(2+Q)
```

The argument helps the function decide which value to return.

(Remember, string values go into string variable boxes, numerical values go into numerical boxes.)

Assignment 27:

1. For each function in the list below:

Tell the name of the function.

Tell the name of its argument.

Tell whether the argument is string or numerical.

Tell whether the argument is constant, variable, expression, or function.

Tell whether the value of the function is string or numerical.

```
RND(Q)
INT(Q)
VAL(ER$)
STR$(INT(RND(8)))
LEN("FUSS")
INT(22.4/V)
```

2. Each line below has errors. Explain what is wrong.

```
10 INT(Q)=65 _____
10 D$=CHR(1) _____
10 PW$=VAL(F$) _____
10 PRINT CHR$ _____
```

3. 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 like an addition problem (with "+" sign and a line under the numbers).
4. Make a number "march" slowly across the screen. That is, write it on the screen, then take its left digit and move it to the right. Keep repeating. Don't forget to erase each digit when you move it.

INSTRUCTOR NOTES 28 JOY STICK FOR ACTION GAMES

This lesson introduces the functions STICK and STRIG.

Joy sticks are commonly used in animated graphics games. In this lesson, the joy stick is used to move a dot around on the screen.

In the next lesson, the dot will shoot a missile upwards and hit stars.

The student will need to understand the X,Y addressing of the squares on the 40 by 24 GRAPHICS 3 screen.

When drawing moving objects, you need to erase each old image before the next image is drawn. The erasing is best done just before the new dot is drawn, to minimize flicker on the screen.

Graphics games may grow to be rather long. BASIC is a little slow for such games. Maximum speed can be obtained if the "working" part of the program is first, and the "initialization" part is at the end, reached by a call from early in the program. This idea is further developed in the lesson on user friendly programs.

Perhaps the most important rule for gaining speed in BASIC is to avoid repeatedly converting numbers to floating point. Rather than:

```
60 POSITION 33,20
```

it is better to write:

```
60 POSITION A,B
```

where variables $A = 33$, $B = 20$ are defined in the initialization section.

Of course, if line 60 is executed just once, it is faster to use the numbers directly in the POSITION command because of the overhead time in initializing A and B. Time savings come only when POSITION A,B is used often in a loop. The innermost loop of a nested set contains, of course, the program lines most in need of careful optimization for speed.

QUESTIONS:

1. Which numbers does the STICK function return?
2. What does the "0" in STICK(0) mean?
3. How do you tell if the button on the joy stick is being held down?

LESSON 28 JOY STICK FOR ACTION GAMES

Plug the joy stick into the left socket on the front.

JOY STICKS AND THEIR BUTTONS

```
Run:  10 REM === JOY STICK ===
      15 DIM C$(4)
      16 POKE 752,1:REM TURN OFF CURSOR
      20 PRINT"clear"
      30 PRINT "PUSH THE JOY STICK AROUND"
      35 PRINT:PRINT "AND PUSH THE BUTTON"
      38 REM CHECK THE STICK
      40 S0=STICK(0)
      42 POSITION 10,5:PRINT "STICK READS:  "
      45 POSITION 10,5:PRINT "STICK READS: ";S0;
      55 REM CHECK THE BUTTON
      60 B0=STRIG(0)
      70 C$="      ":IF B0=0 THEN C$="BANG"
      75 POSITION 10,7:PRINT C$;
      80 FOR T=1 TO 20:NEXT T
      99 GOTO 40
```

Use the BREAK key to end the program. Save to tape.

Line 42 erases the old number before the new number is put in.

THE JOY STICKS

There are 4 joy stick sockets on the front of the ATARI 800, numbered 0, 1, 2, and 3 from left to right.

(Your computer may have zero, 1, 2, 3, or 4 sticks.)

The STICK() function tells which way the stick has been pushed.

THE PUSH BUTTON

To see if the button on stick 0 is being pushed, use the STRIG(0) function. STRIG means "stick trigger."

If the number is 0, then the button is being pushed. If the number is 1, it is not being pushed.

MOVING A SPOT ON THE SCREEN

```
Run:  10 REM MOVE A SPOT
      12 PRINT "clear":GRAPHICS 3+16
      25 SETCOLOR 0,12,8
      27 SETCOLOR 4,0,0
      50 X=20:Y=12
      60 GOSUB 900
      64 COLOR 0:PLOT X,Y:REM ERASE OLD SPOT
      67 X=X+DX:Y=Y+DY
      70 IF X<0 THEN X=0 REM AT LEFT EDGE?
      71 IF Y<0 THEN Y=0
      72 IF X>39 THEN X=39
      73 IF Y>23 THEN Y=23
      80 COLOR 1:PLOT X,Y:REM PUT SPOT ON SCREEN
      99 GOTO 60
     900 REM ASK JOY STICK
     910 S=STICK(0)
     915 IF S=15 THEN DX=0:DY=0:RETURN
     920 IF S< 8 THEN DX=1:GOTO 950
     925 IF S>12 THEN DX=0:GOTO 965
     930 DX=-1
     935 IF S=10 THEN DY=-1:RETURN
     940 IF S=11 THEN DY= 0:RETURN
     945 IF S= 9 THEN DY= 1:RETURN
     950 IF S= 5 THEN DY= 1:RETURN
     955 IF S= 7 THEN DY= 0:RETURN
     960 IF S= 6 THEN DY=-1:RETURN
     965 IF S=14 THEN DY=-1:RETURN
     970 IF S=13 THEN DY= 1:RETURN
```

Use the BREAK key to stop the program. Save to tape.



ERASE AND PUT

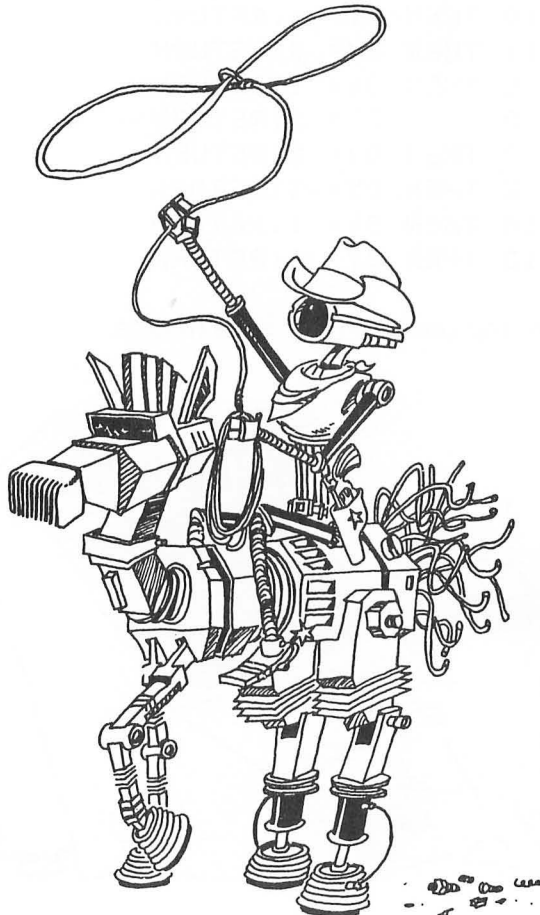
"Erase and put, erase and put" Every time you put a dot, you have to erase it again before putting it somewhere else. Otherwise, you will get more and more dots. (To see this happen, remove line 64.)

Line 64 does the erasing.

Line 80 makes the dot have color 2, then puts it on the screen.

Assignment 28:

1. Make a drawing showing which numbers the STICK command returns when pushed in each of the 8 directions. You will find this diagram useful when making joy stick games.
2. Add a border to the MOVE A DOT program. What changes must you make in lines 70 to 73 so that the border is not erased?
3. Make the dot change color when it moves within 3 squares of any border.



INSTRUCTOR NOTES 29 SHOOTING STARS

The methods of using STRIG to shoot a missile and LOCATE to detect a target are shown. The graphics modes 3 through 8 are explained.

The graphics modes differ from each other in the size of the pixel (the little square of color that PLOT puts down on the screen) and in the number of colors that can be used at once.

Most ATARI computers are used with color TVs. The ATARI 800 has a monitor jack to which a color monitor can be connected. This will usually give a somewhat clearer graphics picture. You would also need to hook up an amplifier and speaker for the sound.

In any event, for most dot-background color combinations, a single pixel or a vertical line does not show up well. Horizontal lines and filled in areas look better. The reason is due to the specifications of the standard TV signal. Not enough bandwidth is allocated to the color signal to give very high resolution.

So like many other art forms, graphics on the ATARI requires skirting around the limitations of the medium. With practice you can pick color combinations and luminance levels that go well together.

The color "black" is really grey (color number zero) with luminance zero.

"White" is grey with luminance 14.

QUESTIONS:

1. In the command LOCATE 3,9,Z what point on the screen is looked at? What goes into the variable box Z?
2. How does the program decide that a star was hit?

LESSON 29 SHOOTING STARS

SHOOT A LASER

Load program "MOVE A SPOT" and add these lines:

```
10 REM SHOOTING
28 SETCOLOR 2,4,8
62 IF STRIG(0)=0 THEN GOSUB 200
200 REM SHOOT
205 IF Y=0 THEN RETURN
210 FOR I=Y-1 TO 0 STEP -1
215 COLOR 3:PLOT X,I
220 FOR T=1 TO 5:NEXT T
225 COLOR 0:PLOT X,I
250 NEXT I
299 RETURN
```

Run. Use the BREAK key to end the program. Save to tape.

Line 62 asks if the button on the stick is being pressed. If yes, then the subroutine at line 200 shoots the laser.

SHOOTING STARS, THE LOCATE FUNCTION

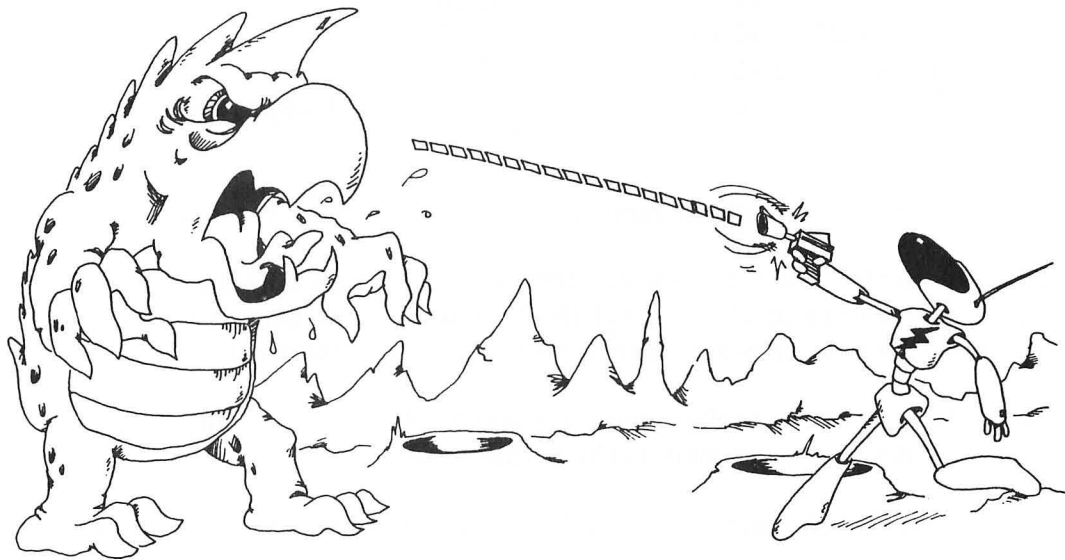
Load the program SHOOTING and add these lines:

```
10 REM *** SHOOTING STARS ***
40 GOSUB 300
212 LOCATE X,I,L
213 IF L=3 THEN GOSUB 400
300 REM INITIALIZE STARS
310 COLOR = 3
311 FOR I=1 TO 10
320 X=RND(9)*39
330 Y=RND(9)*23
350 PLOT X,Y
360 NEXT I
390 RETURN
400 REM HIT STAR
410 SOUND 0,200,8,8
420 FOR T=1 TO 30:NEXT T
430 SOUND 0,0,0,0
490 RETURN
```

Run the program. Use the RESET key to stop the program. Save it to tape.

DRAWING THE STARS

The program calls the subroutine at 300 just once. It draws 10 stars at random on the screen.



HAS THE LASER HIT A STAR?

While the program is drawing the laser line, it looks to see if the next square is a star (color 3). If so, it jumps to the subroutine at line 400 and makes a “hitting” sound.

Line 212 looks at the square by using the LOCATE X,Y,L function. LOCATE looks at point X,Y on the screen and puts the color of the spot there in the variable L.

Then line 213 says if the spot is color 3, it must be a star so go to the “hit” subroutine.

OTHER GRAPHICS MODES

FOUR COLOR GRAPHICS

These modes have 3 colors plus the background color.

Mode 3 is “coarse” (the square dots it makes are large.)

Mode 5 is finer, and mode 7 makes still smaller dots.

MODE	ACROSS	DOWN
GRAPHICS 3+16	Ø TO 39	Ø TO 23
GRAPHICS 5+16	Ø TO 79	Ø TO 47
GRAPHICS 7+16	Ø TO 159	Ø TO 95

Paint pots and brushes used by these modes:

POT	BRUSH IN THE POT
SETCOLOR 0,C,B	COLOR 1
SETCOLOR 1,C,B	COLOR 2
SETCOLOR 2,C,B	COLOR 3
SETCOLOR 4,C,B	COLOR 0, background and border

Where you see "C", you put the number (0 to 15) of the color you want.

Where you see "B", you put the even number (0 to 14) that tells the brightness. (Actually, it is not the brightness, but the "luminance", the amount of white mixed into the color. But the dot does look brighter for larger "B" numbers).

SETCOLOR 4 gives the background color and brightness. You cannot see a border in these modes; it is the same color as the background.

You can use the brush named COLOR 0 to dip into the background paint pot to erase a dot from the screen.

Run:

```
10 REM FOUR COLOR GRAPHICS
12 GRAPHICS 5+16
20 SETCOLOR 0,2,4:REM POT 0
21 SETCOLOR 1,9,8:REM POT 1
22 SETCOLOR 2,6,6:REM POT 2
24 SETCOLOR 4,6,4:REM BACKGROUND
31 COLOR 1:PLOT 0,5:DRAWTO 20,5:REMBRUSH 1
32 COLOR 2:PLOT 0,7:DRAWTO 20,7:REMBRUSH 2
33 COLOR 3:PLOT 0,9:DRAWTO 20,9:REMBRUSH 3
40 FOR T=1 TO 1000:NEXT T
50 COLOR 0:PLOT 0,5:DRAWTO 10,5:REMERASE
99 GOTO 99:REM HOLD PICTURE
```

TWO COLOR GRAPHICS

These modes have a background of one color and brightness and dots of another color and brightness.

Their only advantage over modes 3, 5, and 7 is that they take up less room in memory.

MODE	ACROSS	DOWN
GRAPHICS 4+16	0 TO 79	0 TO 47
GRAPHICS 6+16	0 TO 159	0 TO 95

Here are your choices:

POT

```
SETCOLOR 0,C,B  
SETCOLOR 4,C,B
```

BRUSH IN THE POT

```
COLOR 1  
COLOR 0, background and border
```

```
10 REM TWO COLOR GRAPHICS  
12 GRAPHICS 6+16  
15 SETCOLOR 4,12,8 :REM GREEN BACKGROUND AND BORDER  
17 SETCOLOR 0,8,10 :REM BLUE DOTS  
30 COLOR 1:PLOT 5,5:DRAWTO 20,5 :REM LINE  
40 FOR T=1 TO 500:NEXT T  
50 COLOR 0:PLOT 9,5:DRAWTO 17,5 :REM ERASE  
99 GOTO 99 :REM HOLD PICTURE
```

HIGH RESOLUTION

A background of one color and brightness, a border of another, and dots that have the background color but different brightness.

```
GRAPHICS 8+16 0 TO 319 0 TO 191
```

You choose a color and brightness for the background.

Then the dots must have the same color as the background. But you can choose the brightness of the dots.

POT

```
SETCOLOR 1,0,B  
SETCOLOR 2,C,B  
SETCOLOR 4,C,B
```

BRUSH IN THE POT

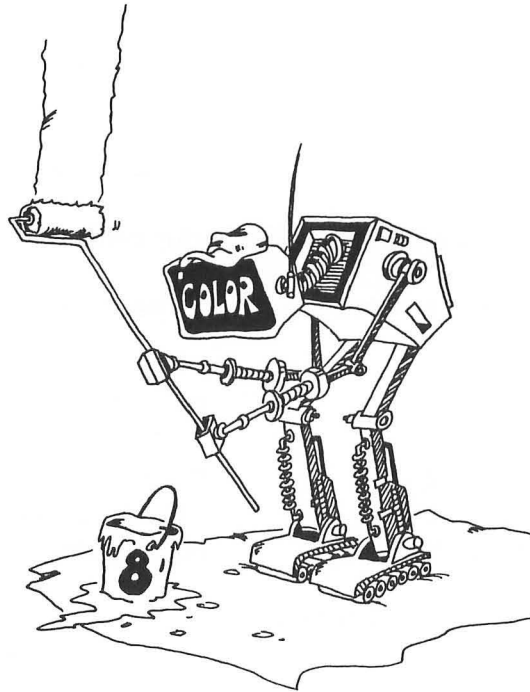
```
COLOR 1  
COLOR 0 background  
border color
```

```
Run: 10 REM HI-RES GRAPHICS  
15 SETCOLOR 4,12,8 :REM GREEN BORDER  
16 SETCOLOR 2,8,4 :REM BLUE BACKGROUND  
17 SETCOLOR 1,0,10 :REM BRIGHT DOTS  
30 COLOR 1:PLOT 5,5:DRAWTO 20,5:REM LINE  
40 FOR T=1 TO 500:NEXT T  
50 COLOR 0:PLOT 9,5:DRAWTO 17,5:REM ERASE  
99 GOTO 99 :REM HOLD PICTURE
```

Line 17 sets the brightness of the dots. The color "0" or "grey" in SETCOLOR 1,0,10 is not used.

Assignment 29:

1. Add to the "hit a star" subroutine so that a big explosion is put on the screen when a star is hit.
2. Change the subroutine at 300 so the laser beam stops after it hits a star.
3. Make the stars of two colors, red and blue, (in the subroutine at line 300) and then in subroutine at 200, let the laser "pop" only the red ones.
4. Change the program so it runs in graphics mode 5.



INSTRUCTOR NOTES 30 ARRAYS

This lesson introduces arrays. The DIM and CLR statements are described.

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, with the index value being the "first name" of the member.

ATARI calls arrays having two indices "matrices." They are conveniently pictured as numbers arranged in a rectangle like the days on a calendar month.

ATARI BASIC does not allow higher dimensional arrays.

Array indices start with "0" which means that an array always has one more member than would appear from the DIM statement. (The array defined by DIM A(7) has 8 members.)

By now it is obvious that ATARI BASIC treats string variables as a special kind of array. Nevertheless, it is better to teach string variables separately from arrays as the differences in use of each outweigh the similarities in form.

QUESTIONS:

1. What does the DIM AD(5) command do?
2. Where do you put the DIM command in the program?
3. What does the command CLR do?
4. What is the "index" or "subscript" of an array?
5. How many boxes does the command DIM SR(5,9) save?

LESSON 30 ARRAYS

MEET THE ARRAY FAMILY

```
22 F(0) = 3
24 F(1) = 44
26 F(2) = 12
```

Each member of the family is a numerical variable.

The family has a “last name” like F or B.

Each member has a “first name” which is a number in (). 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.”

CAREFUL! Only numerical variables come in families like this.

As you know, string variables are already written with an index.

`A$(31)` means the 32nd letter in the string `A$`

THE DIM() COMMAND SAVES BOXES

When the array family goes to a movie, they always reserve seats first. They use a DIM command to do this.



The DIM... command tells the computer to reserve a row of boxes for the array. Each box is big enough to hold one number.

DIM stands for "dimension" which means "size."

For example, the statement

```
18 DIM A(3)
```

saves four memory boxes, one each for the variables A(0), A(1), A(2) and A(3). These boxes contain the number "0" to start with.

Another example:

```
30 DIM A(3),B$(4)
```

This time, DIM reserves 4 numerical boxes for the A() array and one box for the string B\$(). The B\$ box is large enough to hold 5 letters and is empty to start with.

Rule: The program must execute the DIM command before it executes any command using the array or the string.

Rule: The program may execute only one DIM command for a given array or string.

A COMMON ERROR!

Looping the program through the DIM command more than once is a "NO NO" that everyone does once in a while. Run this:

```
10 DIM Q(15)
20 Q(3)= 3/7
30 GOTO 10
```

You get: ERROR- 9 AT LINE 10

Change line 30 to: 30 GOTO 20 to fix it.

THE CLR COMMAND

If you are done using the strings and arrays and want to give them new dimensions and use them again, use the CLR command.

CLR means "clear" and it clears the memory of all string variables and numerical arrays without erasing the numerical variables or the program.

Add this: 22 PRINT Q(3)
25 CLR and run it.

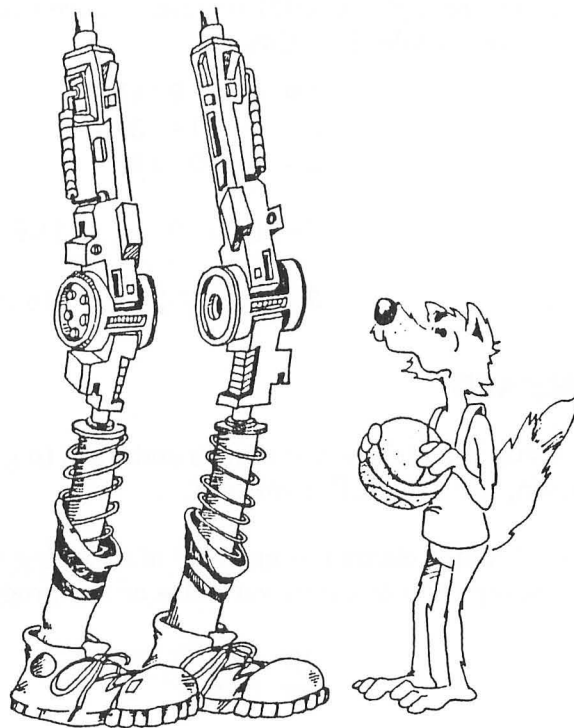
You see ERROR- 9 AT LINE 20
because the CLR erased the box A() that the DIM made.

MAKING A LIST

Run:

```
10 REM +++ IN A ROW +++
20 PRINT "clear"
30 DIM A(5)
35 PRINT"ENTER A NUMBER "
40 FOR N=0 TO 5
45 IF N>0 THEN PRINT"ANOTHER"
50 INPUT Q
55 A(N) = Q
60 NEXT N
70 PRINT
100 REM PUT IN A ROW
105 PRINT"HERE THEY ARE IN A ROW"
106 PRINT
110 FOR I=0 TO 5
120 PRINT A(I);" ";
130 NEXT I
```

Arrays are usually used in a loop where the index keeps changing. Lines 50 and 120 are used in loops.



ONE DIMENSION, TWO DIMENSION, ...

The arrays that have one index are called "one dimensional arrays."

But arrays can have 2 indices. A two dimensional array is often called a "matrix." (The plural is "matrices.")

Two dimensional arrays have their "family members" put in a rectangle like the days in a month on a calander.

```
10 REM +++ TWO-DIM ARRAY +++
18 PRINT "clear"
19 REM ----- MAKE THE ARRAY
20 DIM T(4,5)
30 FOR X=0 TO 4
40 FOR Y=0 TO 5
50 T(X,Y)= X+Y
60 NEXT Y
61 NEXT X
70 REM ----- PRINT OUT THE ARRAY
80 FOR J=0 TO 4
82 POSITION 0,2+3*J
85 FOR I=0 TO 5
86 REM ----- J="WHICH ROW"
87 PRINT " ";T(J,I);" ";
90 NEXT I
91 NEXT J
```

Assignment 30:

1. Write a program that uses an array to store the number of days each month has. That is $D(1)=31$, $D(2)=28$, etc.
2. Use a two dimensional array to make a "school calendar" program. It could use an array made by `DIM CA$(5,6)` so that each day of the week could have an entry for each school hour.



INSTRUCTOR NOTES 31 LOGIC: AND, OR, NOT

This lesson treats the AND, OR and NOT relations and the numerical values for TRUE and FALSE. These are important for some types of IF statements.

The TEENAGER program in lesson 13 used a nested IF to print out "You are a teenager." A more concise logic uses the OR relation.

There are several abstract ideas in this lesson that are difficult to grasp. The fact that TRUE and FALSE have numerical values of 1 and 0 is bad enough. But in addition, the computer often treats any number that is not zero as being TRUE.

QUESTIONS:

1. For each IF statement, tell if it will print anything:

```
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 23
10 PRINT NOT 0
10 PRINT 3 AND 7
10 PRINT 3 AND 0
```

LESSON 31 LOGIC: AND, OR, NOT

ANOTHER TEENAGER PROGRAM

```
Enter: 10 REM <<< AND, OR, NOT >>>
        20 PRINT "clear"
        25 DIM N$(40)
        30 PRINT"YOUR FIRST NAME ":INPUT N$
        35 PRINT
        40 PRINT"YOUR AGE ":INPUT A
        45 PRINT
        50 IF (A>12) AND (A<20) THEN PRINT N$;" IS A TEENAGER,"
        55 NFLAG = (A<13) OR (A>19)
        60 IF NFLAG THEN PRINT N$;" IS NOT A TEENAGER."
        65 PRINT
        70 IF (NOT NFLAG) AND (A=16) THEN
PRINT "AND " ;N$;" IS SWEET SIXTEEN."
```

Run and save to tape.

WHAT DOES "AND" MEAN?

Two things are true about teenagers: they are over 12 years old and they are less than 20 years old. Look at line 50.

IF (you are over 12) AND (you are less than 20) THEN (you are a teenager).

WHAT DOES 'OR' MEAN?

In line 55 the OR is used. Two things are said: "age is under 13" and "age is over 19."

Only one of them needs to be true for you to be "not a teenager."

IF (you are under 13) OR (you are over 19)
THEN (you are not a teenager).

TRUE AND FALSE ARE NUMBERS

How does the computer do it? It says true and false are numbers.

Rule: TRUE is the number 1

FALSE is the number 0

(It is easy to remember that 0 is FALSE because zero 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 numerical variable names on the front. Run this:

```
10 N= (3=22)  
20 PRINT N
```

The number 0 is stored in the box N because $3=22$ is FALSE.

And this:

```
10 N= "B"="B"  
20 PRINT N
```

The number 1 is stored in the box N because the two letters in the quotes are the same so the statement "B"="B" is TRUE.



THE IF COMMAND TELLS LITTLE WHITE LIES

The IF command looks like this:

```
1 0 IF (something A) THEN (command 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"
```

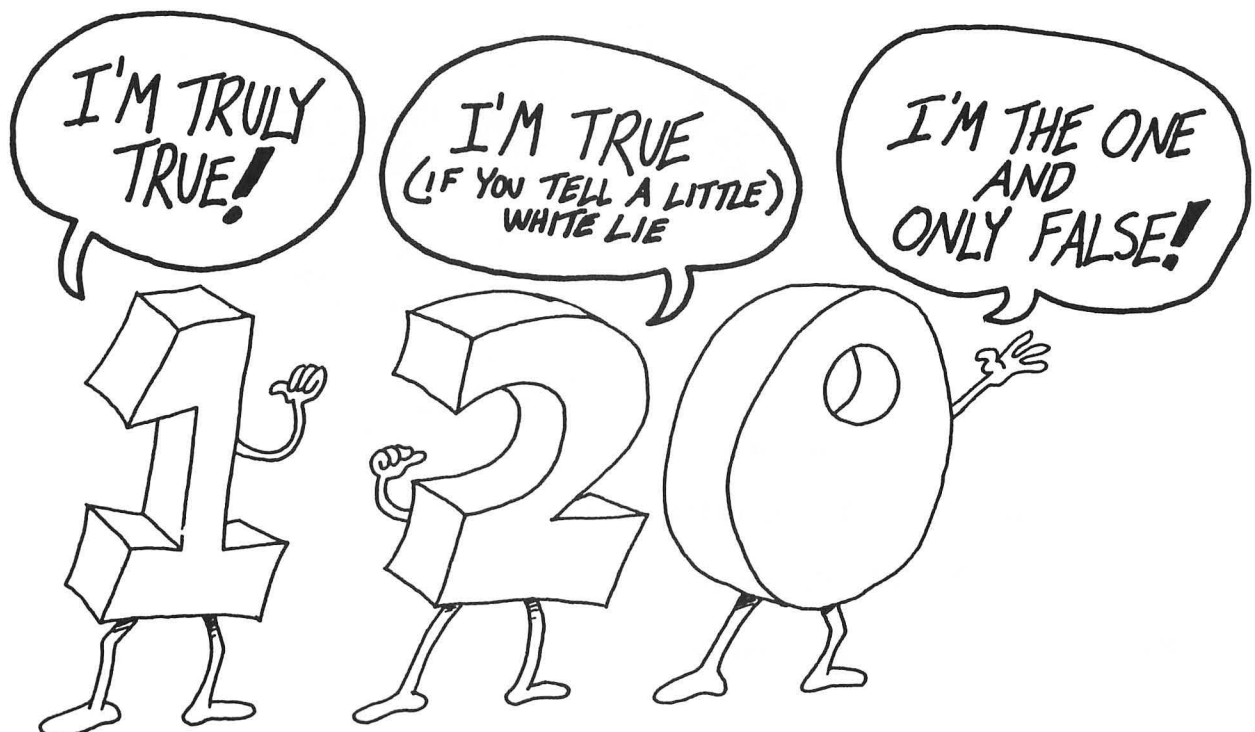
What did it print? _____

Rule: In an IF, the computer looks at "something A."

If it is zero, the computer says "something A is FALSE," and skips what is after THEN.

If it is not zero, the computer says "something A is TRUE," and obeys the commands after THEN.

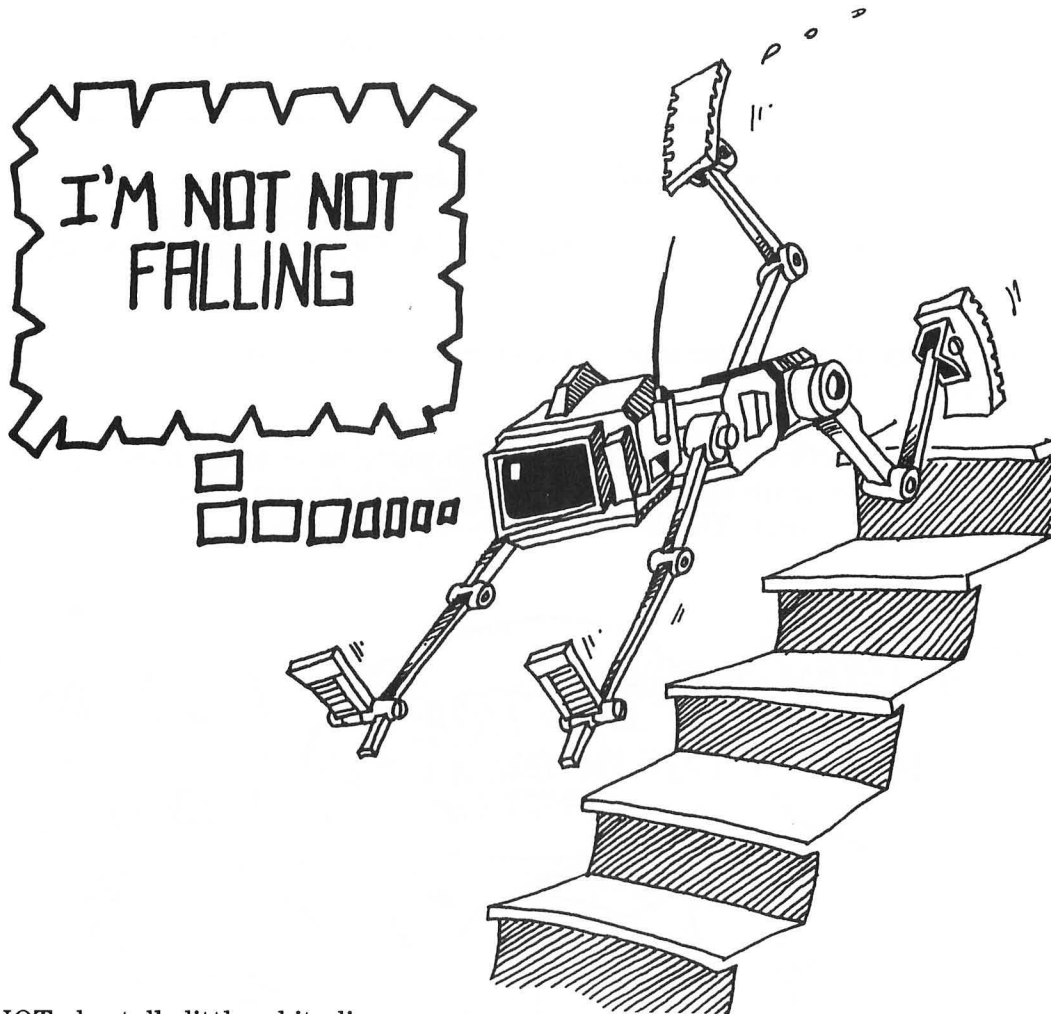
The IF command tells little white lies. TRUE is supposed to be the number "1," but the IF stretches the truth to say "TRUE is anything that is not FALSE." That is, any number that is not zero is TRUE.



WHAT DOES "NOT" MEAN?

NOT changes FALSE to TRUE and TRUE to FALSE. Try this:

```
10 REM ??? DOUBLE NEGATIVE ???  
20 N=3  
30 PRINT "N "; TAB(20); N  
40 PRINT "NOT N"; TAB(20); NOT N  
50 PRINT "NOT NOT N "; TAB(20); NOT(NOT N)  
60 REM The computer knows that "I don't have no..."  
61 REM means "I do have ...."
```



The NOT also tells little white lies:

N was 3, which is called TRUE, (a little white lie.)

Then NOT 3 is FALSE, or the number 0.

Finally, NOT (NOT 3) is the same as NOT (0) or NOT (FALSE) or TRUE or the number 1.

THE LOGICAL SIGNS

You can use these 6 symbols in the "something A" phrase:

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

You have to press two keys to make the <> sign and the <= and >= signs.

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 "something A" phrases mean the same:

2<=Q (2<Q) OR (2=Q)

Assignment 31:

1. Tell what will be found in the box N if:

```
N=4=4
N="G"<>"S"
N=5>7
N=3>2 AND 3<2
N=4=3 OR 4=4
N=NOT 0
N=5>=4
```

2. Tell if the word "JELLYBEAN" will be printed:

```
IF 0 THEN PRINT "JELLYBEAN"
IF 1 THEN PRINT "JELLYBEAN"
IF 9 THEN PRINT "JELLYBEAN"
IF 3<>0 THEN PRINT "JELLYBEAN"
IF 2 AND 4 THEN PRINT "JELLYBEAN"
IF 0 OR 1 THEN PRINT "JELLYBEAN"
IF NOT 3 THEN PRINT "JELLYBEAN"
IF "A"="Z" THEN PRINT "JELLYBEAN"
IF NOT(3) AND 2 THEN PRINT "JELLYBEAN"
IF NOT(0) OR 0 THEN PRINT "JELLYBEAN"
IF 4<=5
```

3. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing and count them. If there are 2 such words there is a double negative. Test the program on the sentence "COMPUTERS AIN'T GOT NO BRAINS".

INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS

This lesson illustrates clear programs which interact with the user in a “friendly” way.

The “spaghetti” program should be discouraged. A format for writing programs is presented in this lesson. While methods of imposing order on the task are largely a matter of taste, the methods used in this lesson can serve to introduce the ideas.

“User friendly” means that the screen displays are easy to read, keyboard input is “RETURN key free” as much as possible, and errors are “trapped.” Ask if entries are OK. If not, give an opportunity to fix things. Instructions and “HELP” should be available. Prompts need to be given. Beginners need complete prompts, but experienced users would rather have curt prompts.

It is hard to teach the writing of “user friendly” programs. Success depends mostly on the attitude of the programmer. The best advice is to “turn up your annoyance detectors to high” as you write and debug the program.

Most young students will not progress very far toward fully “friendly” programming. To be acquainted with the desirability of “friendly” programming and to use some simple techniques toward accomplishing it 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? (Avoid using the RETURN key.)
5. What is an “error trap”? How would you trap errors if you asked your user to enter a number from 1 to 5?
6. In what part of the program are most of the GOSUB commands found?
7. Why put the “STARTING STUFF” section of the program at the end of the program (at high line numbers)?

LESSON 32 USER FRIENDLY PROGRAMS

There are two kinds of users:

1. Most want to run the program. They need:

- instructions
- prompts
- clear writing on the screen
- no clutter on the screen
- erasing old stuff from the screen
- not too much key pressing
- protection from their own stupid errors

2. Some want to change the program. They need:

- a program made in parts
- each part with a title in a REM
- explanations in the program

(Don't forget you are a user of your own programs, too! Be kind to yourself!)

PROGRAMS HAVE THREE PARTS

"STARTING STUFF": at the beginning of the program run.

- give instructions to the user
- draw a screen display
- set variables to their starting values
- ask the user for starting information

MAIN LOOP:

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

SUBROUTINES:

- do parts of the program



PROGRAM OUTLINE

```

1 GOTO 1000:REM *** program name ***
---
100 REM MAIN LOOP
---
--- calls subroutines
---
199 END
1000:
1001 REM *** program name ***
1002:
--- REM 's that give a description of the
--- program, variable names, etc.
---
1999:
2000 REM STARTING STUFF
---
--- ask for starting information
--- set variable values
--- give instructions
---
2999 GOTO 100

```

PUT THE MAIN LOOP AT THE BEGINNING OF THE PROGRAM

Put the MAIN LOOP near the front because it will run faster there.

PUT STARTING STUFF AT THE END OF THE PROGRAM

Put the STARTING STUFF near the back because it may be the biggest part of the program, and you may keep adding to it as you write to make the program more “user friendly.” It does not need to run fast.

PUT SUBROUTINES IN THREE PLACES

between line 2 and line 99 for subroutines that must run fast
after line 2999 for starting stuff subroutines
between lines 200 and 999 for the rest of the subroutines

INFORMATION PLEASE

```
280 PRINT "DO YOU WANT INSTRUCTIONS  
<Y/N> "
```

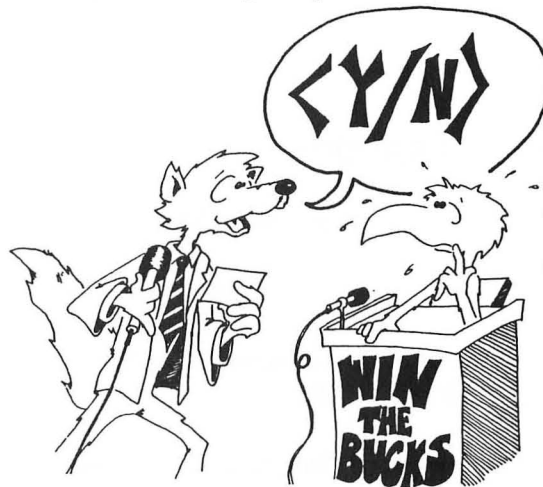
This lets a beginner see instructions, and lets others say "no".

TIE A STRING AROUND THE USER'S FINGER

Use a "prompt" to remind users what choices they have.

Example: Y/N where the choice is Y for "yes" or N for "no"

Beginners need long prompts. Other users like short prompts.



DON'T GIVE THE USER A HEADACHE

SCROLLING gives headaches!

BASIC usually scrolls. It writes new lines at the bottom of the screen and pushes old lines up.

It is like the scrolls the Romans used for writing. They unwound from the bottom and wound up at the top.

Avoid scrolling. Use POSITION to print just where you want. Erase by printing a string of blanks to the same spot.

Use delay loops so the writing stays on the screen while the user reads it.



OUCH! MY FINGERS HURT

Use the GET command to enter single letters. This saves having to press RETURN.

```
100 OPEN #1,4,0,"K:"
102 DIM A$(1)
380 PRINT "DO YOU NEED INSTRUCTIONS? <Y/N> "
382 GET #1,A:A$=ASC$(A)
383 IF A$="A" THEN 3400
```

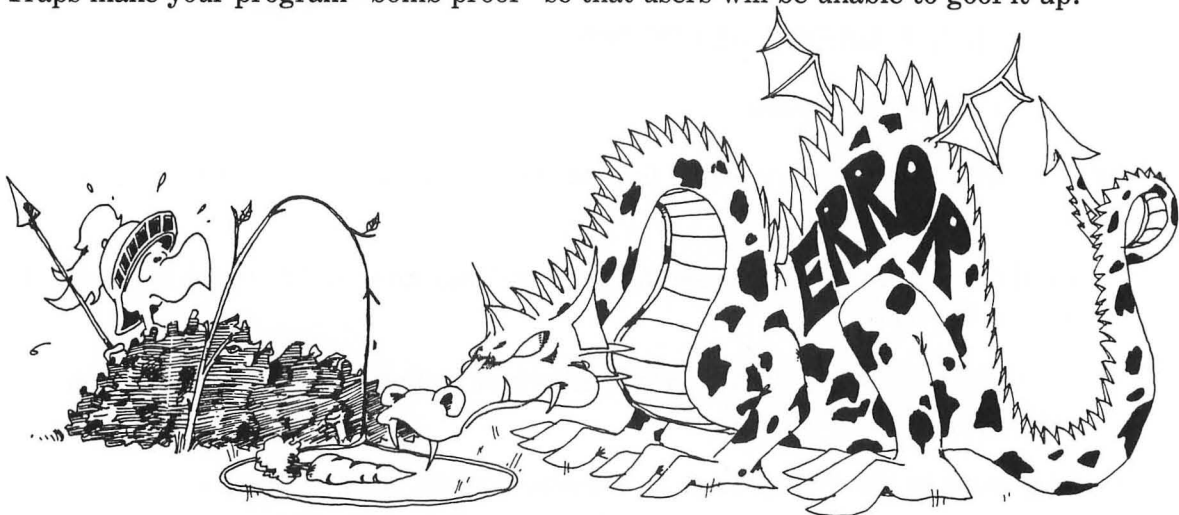
SET TRAPS FOR ERRORS

Example: Add this line to the above lines:

```
384 IF A$<>"N" THEN GOTO 380
```

Line 380 asked for only two choices, Y or N. If the user presses some other key, line 384 sends him back to line 380.

Traps make your program "bomb proof" so that users will be unable to goof it up!



Assignment 32:

1. Look at the COLOR EATER program. Add REM's to explain the lines in the program. Fix up the program to be user friendly.
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 normal order

Match letters against a normal alphabet to code or decode.

DRAGONETBCFHIJKLMPQSUUVWXYZ

ABCDEFGHIJKLMNPOQRSTUVWXYZ

The user chooses to code or decode from a menu.

```
1 GOTO 1000:REM *** COLOR EATER ***
100 NC=0
101 FOR I=X-1 TO X+1
102 FOR J=Y-1 TO Y+1
104 IF I<0 THEN I=0
105 IF I>39 THEN GOTO 120
106 IF J<0 THEN J=0
107 IF J>23 THEN GOTO 120
109 LOCATE I,J,CC
110 IF CC=C THEN X=I:Y=J:COLOR 0:PLOT X,Y:GOTO 100
112 IF CC<>0 THEN NC=1
115 NEXT J:NEXT I
120 C=C+1:IF C>3 THEN C=1
121 SOUND 1,0,0,0
122 SOUND 0,100+C*40,10,8
125 IF NC=0 THEN GOSUB 300
199 GOTO 100
300 REM COLOR EATER HAS NO FOOD
310 X=X+1:IF X>39 THEN X=1
320 SOUND 1,30,10,8
330 SOUND 0,0,0,0
399 RETURN
1000 REM STARTING STUFF
2012 GRAPHICS 3+16
2013 SETCOLOR 0,5,8
2014 SETCOLOR 1,8,4
2015 SETCOLOR 2,12,4
2016 SETCOLOR 4,0,10
2020 FOR I=0 TO 39
2030 FOR J=0 TO 23
2040 COLOR INT(RND(9)*3)+1
2050 PLOT I,J
2060 NEXT J:NEXT I
2100 X=20:Y=12
2999 GOTO 100
```

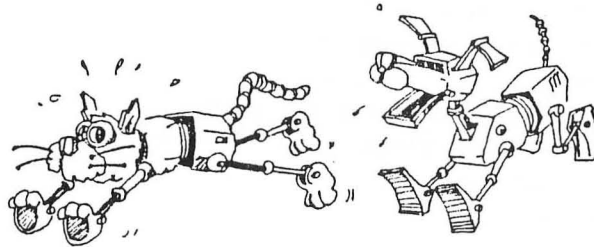

INSTRUCTOR NOTES 33 DEBUGGING, STOP, CONT

It is difficult to drill systematically on debugging, unless you are NASA with NASA's budget and time scale.

We present a series of small techniques and a description of how to put them together in a debugging scheme. Only practice will serve to make debugging a chore that the student approaches with some confidence.

QUESTIONS:

1. What two ways can you make the computer print
STOPPED AT LINE 55
while the program is running?
2. How are the STOP and the END commands different?
3. How is the STOP command different from the BREAK key?
4. What does the CONT command do?
5. Why would you put STOP commands in your program?
6. How do delay loops help you debug a program?
7. How do extra PRINT commands help you debug a program?
8. Why do you take the STOP and extra PRINT commands out of the program after you have fixed the errors?
9. Can you pick in what line the BREAK key will stop the program? Can you pick using the STOP command?



LESSON 33 DEBUGGING, STOP, CONT

THE STOP COMMAND

Enter and run:

```
10 REM SECRET STOP
20 PRINT "clear"
25 J = INT(RND(9)*200)
30 FOR I=0 TO 200
40 IF I=J THEN STOP
50 NEXT I
```

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

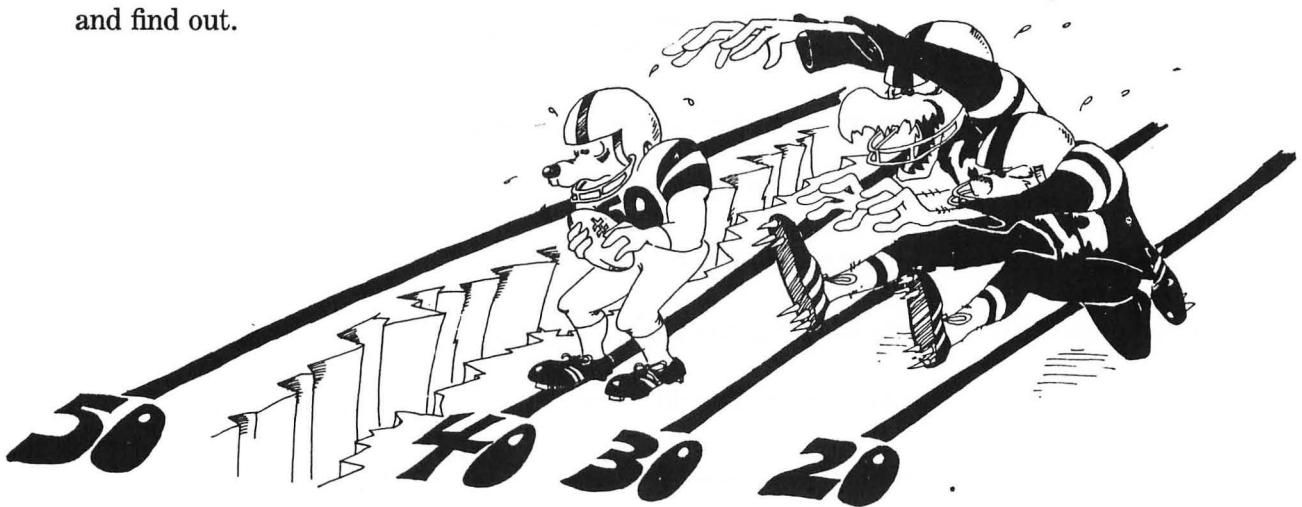
STOPPED AT LINE 40

What do you suppose the secret value of I was?

Enter:

PRINT I (No line number)

and find out.



HOW TO START IT AGAIN

Enter the command CONT. Try it!

“STOP” IS LIKE “END”

STOP makes the computer stop and enter the edit mode.

It is like END except it prints the number of the line that the STOP is in.

You can have as many STOP commands in your program as you like.

STOP is used for debugging your program.

ANOTHER WAY TO STOP RUNNING THE PROGRAM

You can stop running the program by pressing the BREAK key.

Try it:

```
10 REM GO FOREVER
15 PRINT "clear"
20 PRINT "MUD "
21 GOSUB 40
22 PRINT "  TURTLES "
23 GOSUB 40
24 PRINT "      OF "
25 GOSUB 40
26 PRINT "          THE "
27 GOSUB 40
28 PRINT "              WORLD ":PRINT
29 GOSUB 40
30 PRINT "UNITE!":PRINT
31 GOSUB 40
39 GOTO 10
40 FOR T=1 TO 500:NEXT T
49 RETURN
```

The BREAK key stops the program wherever it is. It prints:

STOPPED AT LINE XX and enters the edit mode.

(where XX is the line number where it stops.)

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

This program almost always stops in line 40. Why? _____

WHAT DO YOU DO AFTER YOU STOP?

You put STOP in whatever part of your program is not working right.

Then you run the program. After it stops, you look to see what happened.

(Or you use the BREAK 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 command to look at variables. Do they have the values you expected?

Use the LET command to change the values of variables

Use the computer as a calculator to check what the program is doing

If you find the trouble:

You may add a line

Change a line

Delete a line.

STARTING THE PROGRAM AGAIN

There are three ways to start a program. They are:

CONT	picks up at the next line
GOTO XX	where XX is a line number
RUN	your old friend

What is the difference between these three ways?

CONT, GOTO XX	These two way keeps the variable boxes left over from the last time you ran.
RUN	This way throws away all the variable boxes made the last time, then executes the program from the beginning.

The CONT command starts the program at the next numbered line. It leaves the values in the variable boxes the same.

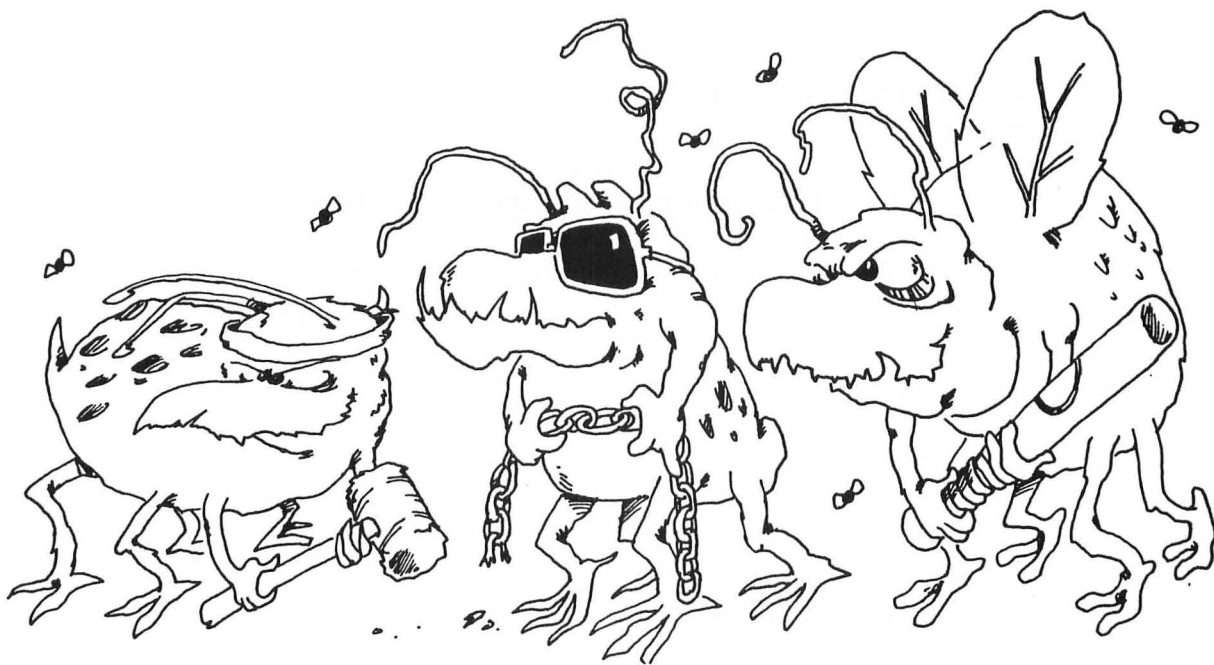
CAREFUL! The program does not start at the next statement if there are several statements in the same line. It goes on to the next numbered line. This may mean that the program does not run correctly after CONT.

So: Do a LIST before you do a CONT, to see if the line has more than one statement. If it does, you may not want to use CONT.

You may start running the program at a different spot by entering (without a line number in front) the command:

GOTO XX

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



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 put some **PRINT** lines in that will tell what is happening.
3. Or you can put **STOP** commands 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 have fixed the program, take the **PRINT** lines, the **STOP**'s and the delay loops out of the program.

Assignment 33:

1. Go back to the **SNAKE** program and fix up some of the bugs. For example, the program “crashes” when the snake hits a wall. Add “food” for the snake. Add score keeping. Let the game end if the snake touches a wall. (**SNAKE** page 133)
2. Go back and fix up some other program that you have written.



RESERVED WORDS IN ATARI BASIC

ABS	ADR	AND	ASC	ATN
BYE				
CLOAD	CHR\$ COLOR CSAVE	CLOG COM	CLOSE CONT	CLR COS
DATA	DEG	DIM	DOS	DRAWTO
END	ENTER	EXP		
FOR	FRE			
GET	GOSUB	GOTO	GRAPHICS	
IF	INPUT	INT		
LEN	LET LOCATE	LIST LOG	LOAD LPRINT	
NEW	NEXT	NOT	NOTE	
ON	OPEN	OR		
PADDLE	PEEK POP PUT	PLOT POSITION	POINT PRINT	POKE PTRIG
RAD	READ RND	REM RUN	RESTORE	RETURN
SAVE	SETCOLOR SQR STRIG	SGN STATUS STOP	SIN STEP STR\$	SOUND STICK
THEN	TO	TRAP		
USR				
VAL				
XIO				

Variable names in ATARI BASIC can be any length. You should not use names that have reserved words in them, beginning, middle, or end.

ANSWERS TO ASSIGNMENTS

A1-3

```
10 REM GREETING
20 PRINT "HI THERE"
30 PRINT "ATARI COMPUTER"
```

A2-1

```
10 REM NAMES
20 PRINT "MINDA"
30 PRINT "ANNE"
40 PRINT "inverse CARLSON"
50 REM REMEMBER "inverse" MEANS PRESS "ATARI" KEY
```

A2-3

```
10 REM NAMES
15 PRINT "clear"
16 REM "clear" MEANS "ESC SHIFT CLEAR"
20 PRINT "buzz MINDA"
30 PRINT "buzz ANNE"
40 PRINT "buzz inverse CARLSON"
50 REM HOW DO YOU ENTER "buzz"?
```

A3-5

```
10 REM BIRDS
15 PRINT "clear"
20 PRINT
25 PRINT "buzz ---0---"
30 PRINT
40 PRINT
50 PRINT "          buzz ---0---"
60 PRINT
70 PRINT
80 PRINT "          buzz --0--"
```

A4-4

```
10 REM SMILE
12 PRINT "clear"
20 PRINT
30 PRINT
```



```

40 PRINT
50 PRINT "    00    00    "
60 PRINT
61 PRINT
62 PRINT
63 PRINT " *          * "
64 PRINT "  *        *  "
65 PRINT "    *      *    "
66 PRINT "      *****      "

```

A5-2

```

10 REM THE STRING BOX
12 PRINT "clear"
15 DIM N$(40)
20 LET N$="MINDA"
30 PRINT N$
40 LET N$="BETH"
50 PRINT N$

```

A6-1

```

10 REM COLORED NAME
12 DIM N$(40)
15 PRINT "clear"
20 PRINT "YOUR NAME?"
25 INPUT N$
30 SETCOLOR 2, 4,8
40 SETCOLOR 4,12,6
50 PRINT N$
60 PRINT "LIKE THE COLORS?"

```

A6-2

```

10 REM WHAT'S IN THE BOX?
12 DIM F$(40),A$(40)
15 PRINT "clear"
20 PRINT "YOUR FAVORITE FOOD?"
22 INPUT F$
25 PRINT
30 PRINT "YOUR FAVORITE ANIMAL?"
32 INPUT A$
40 F$=A$
50 PRINT "NOW THE BOX HAS "
52 PRINT

```

```

55 PRINT " '";F$;"' "
60 PRINT
65 PRINT "IN IT."

```

A7-3

```

10 REM MUSIC
12 PRINT "clear"
15 DIM G$(40), T$(40)
20 PRINT "WHAT IS YOUR FAVORITE MUSICAL GROUP?"
25 INPUT G$
30 PRINT "clear WHAT TUNE DO THEY PLAY?"
35 INPUT T$
40 PRINT "clear"
50 PRINT
52 GRAPHICS 2+16
55 PRINT G$;" PLAYS ";T$
99 GOTO 99

```

A8-3

```

10 REM FRIENDS
15 PRINT "clear"
20 PRINT "MINDA"
25 PRINT
30 PRINT "NELL"
99 GOTO 20

```

A9B-1

```

2 REM PIZZA
3 REM BY CHRIS CLARK, JR. AGE 14 GOING ON (YOU FIGURE IT
OUT)
4 PRINT "clear"
5 PRINT "HALLO, AY AM MARIO, YOUR PIZZA MAN."
6 PRINT
7 PRINT "JUST TELL ME ZE GORY DETAILS AND I'LL DO ZE REST"
9 PRINT
10 PRINT "WHAT SIZE SHOULD ZIS PIZZA BE?" (S/M/L)"
15 DIM S$(1),CH$(1)
20 INPUT S$
21 PRINT
30 IF S$="S" THEN PRINT "ON A DIET? HO HO!"
33 IF S$="M" THEN PRINT "GOOD CHOICE-NOT TOO BIG, BUT
FILLING!"

```

```

38 IF S$="L" THEN PRINT "YOU MUST HAVE A BIG BUNCH AT
HOME!"
39 PRINT
40 PRINT "NOW, YOU WANT DOUBLE GHEES ON ZIS (Y/N)?"
42 INPUT CH$
45 REM ETC.
50 REM MUSHROOMS, ETC.
60 REM ANCHOVIES, ETC.
80 REM PEPPERS, ETC.
90 REM MEAT, ETC.
150 PRINT "inverse"
151 PRINT "HOKAY, HERE IS YOUR PIZZA!"
152 PRINT "normal"
154 IF S$="S" THEN PRINT "WAN SMALL PIZZA WITH ";
155 REM ETC.
160 IF BASE$="P" THEN PRINT "PEPPERONI"
165 REM ETC., ETC.
238 PRINT
240 FOR J=1 TO 2000
242 NEXT J

```

A9A-2

```

10 REM BOYS AND GIRLS
12 DIM A$(10)
15 PRINT "clear"
20 PRINT
25 PRINT "ARE YOU A BOY OR A GIRL?"
26 PRINT "ANSWER 'BOY' OR 'GIRL'"
30 INPUT A$
32 PRINT
35 IF A$="BOY" THEN PRINT "SNIPS AND SNAILS"
40 IF A$="GIRL" THEN PRINT "SUGAR AND SPICE"

```

A9B-2

```

10 REM === COLOR GUESSING GAME ==
15 DIM C$(40), G$(40)
20 PRINT "clear"
23 PRINT
24 PRINT
25 PRINT "PLAYER 2 TURN YOUR BACK"
30 PRINT "PLAYER 1 ENTER A COLOR"
35 INPUT C$
40 PRINT "clear"
42 PRINT
43 PRINT

```

```

50 PRINT "PLAYER 2 TURN AROUND AND GUESS"
52 PRINT
53 PRINT
54 PRINT
55 INPUT G$
60 IF G$<>C$ THEN PRINT "WRONG!"
65 IF G$ =C$ THEN PRINT "RIGHT!"
67 PRINT
70 GOTO 55

```

A10-1

```

10 REM BIRTH YEAR
12 DIM Y$(1)
15 PRINT "clear"
30 PRINT "HOW OLD ARE YOU?"
32 PRINT
34 INPUT A
36 PRINT
40 PRINT "AND WHAT YEAR IS IT NOW?"
45 INPUT Y
50 B=Y-A
52 PRINT
55 PRINT "HAS YOUR BIRTHDAY COME YET THIS YEAR?"
56 PRINT "<Y/N>"
60 INPUT Y$
65 IF Y$="N" THEN B=B-1
70 PRINT "YOU WERE BORN IN ";B;","

```

A10-2

```

10 REM MULTIPLICATION
15 PRINT "clear"
20 PRINT
22 PRINT
24 PRINT
30 PRINT "GIVE ME A NUMBER"
32 PRINT
35 INPUT A
37 PRINT
38 PRINT
40 PRINT "GIVE ME ANOTHER "
42 PRINT
45 INPUT B
48 C=A*B
50 PRINT
52 PRINT
60 PRINT "THEIR PRODUCT IS ";C

```

A11A-1

```

10 REM &%$%! INSULTS !%$%&
12 DIM N$(40)
15 PRINT "clear"
16 PRINT
17 PRINT
20 PRINT "HEY YOU!! WHAT IS YOUR NAME?"
21 PRINT
22 PRINT
25 INPUT N$
30 PRINT "clear"
31 PRINT
32 PRINT
35 PRINT N$
36 PRINT
37 PRINT
38 FOR T=1 TO 1000:NEXT T
40 PRINT "BAH!!"
41 PRINT
42 PRINT "YOUR FATHER EATS LEEKS!!!"
50 PRINT "buzz"

```

A11B-1

```

10 REM SOUNDS
12 DIM Y$(1)
15 PRINT "clear"
20 PRINT
21 PRINT
22 PRINT "PITCH <1 TO 255>"
25 PRINT
26 INPUT P
27 PRINT
30 PRINT "LENGTH IN SECONDS"
31 PRINT
32 INPUT S
33 PRINT
35 T=S*500
40 SOUND 1,P,10,8
45 FOR I = 1 TO T:NEXT I
50 SOUND 1,P,10,0
60 PRINT "ANOTHER?"
61 INPUT Y$
65 IF Y$="N" THEN END
99 GOTO 15

```

A11B-2

```

10 REM CLOCK
15 PRINT "clear"
20 PRINT "TIME? <H, M, S>"
25 INPUT H, M, S
30 FOR T=1 TO 400:NEXT T
31 S=S+1
32 PRINT H;" ":"M;" ":"S
50 IF S<60 THEN GOTO 60
55 S=0
56 M=M+1
60 GOTO 30
70 REM DO SAME FOR HOURS

```

A12B-3

```

10 REM I GOT YOUR NUMBER!
20 PRINT "clear"
25 PRINT
26 PRINT
27 PRINT
30 PRINT "GIVE ME A NUMBER BETWEEN ZERO AND TEN:"
35 PRINT
36 PRINT
37 PRINT
40 INPUT N
45 PRINT
46 PRINT
50 IF N=0 THEN PRINT "I GOT PLENTY OF NOTHING!"
51 IF N=1 THEN PRINT "I'M NUMBER ONE!"
52 IF N=2 THEN PRINT "TWO IS COMPANY"
53 REM ETC.
70 IF N>10 THEN GOTO 99
90 GOTO 25
99 PRINT "THAT'S ALL, FOLKS"

```

A13-1

```

10 REM ** A PAIR OF DICE **
12 DIM Y$(1)
15 PRINT "clear"
20 LET D1=1+INT(RND(9)*6)
22 LET D2=1+INT(RND(9)*6)
25 D=D1+D2
30 PRINT "THE ROLL GAVE:"

```

```

32 PRINT
33 PRINT " THE FIRST DIE ";D1
34 PRINT " THE SECOND ";D2
35 PRINT " THE DICE "; D
47 PRINT
48 PRINT
50 PRINT "AGAIN?"
55 INPUT Y$
57 PRINT
58 PRINT
60 IF Y$="Y" THEN GOTO 15

```

A13-2

```

10 REM PAPER, SCISSORS, ROCK
12 PRINT "clear"
13 PRINT
14 PRINT
15 DIM C$(1), Y$(1)
16 PRINT "PLAY THE "
17 PRINT
18 PRINT
19 PRINT " P A P E R "
20 PRINT " S C I S S O R S "
21 PRINT " R O C K "
22 PRINT
23 PRINT "GAME AGAINST THE COMPUTER"
24 PRINT
25 PRINT "PRESS 'BREAK' KEY TO END GAME "
26 PRINT "ENTER YOUR CHOICE <P,S,R>"
29 REM COMPUTER CHOOSES ITS MOVE
30 C=INT(RND(9)*3)+1
31 IF C=1 THEN C$="P"
33 IF C=2 THEN C$="S"
34 IF C=3 THEN C$="R"
35 REM C$ IS THE COMPUTER'S CHOICE
37 INPUT Y$
38 REM Y$ IS YOUR CHOICE
39 REM
40 REM IS THERE A TIE?
41 REM
50 IF C$<>Y$ THEN GOTO 60
52 REM THERE IS A TIE
55 PRINT " TIE"
57 GOTO 30
59 REM
60 REM NO TIE, WHO WINS?

```

```

61 REM
62 IF C$="P" THEN IF Y$="S" THEN 70
63 IF C$="S" THEN IF Y$="R" THEN 70
64 IF C$="R" THEN IF Y$="P" THEN 70
65 REM COMPUTER WINS
66 PRINT "                COMPUTER WINS"
69 GOTO 30
70 REM
71 REM YOU WIN
72 REM
75 PRINT "    YOU WIN"
79 GOTO 30

```

A15-1

```

10 REM !!! VACATION !!!
11 DIM M$(60), P$(60), Q$(60), Z$(60)
13 PRINT "clear"
14 PRINT
15 PRINT
16 PRINT
20 REM HEADING
21 PRINT "VACATION CHOOSING PROGRAM "
22 PRINT
23 PRINT "PICKS YOUR VACATION BY THE"
24 PRINT "AMOUNT YOU WANT TO SPEND"
25 PRINT
30 REM INSTRUCTIONS
31 PRINT "ENTER THE AMOUNT IN DOLLARS THAT "
32 PRINT "YOU CAN SPEND"
33 PRINT
35 REM GET DOLLAR AMOUNT
37 INPUT D
38 PRINT
40 M$="FLIP PENNIES WITH YOUR KID BROTHER"
41 P$="SPEND THE AFTERNOON IN BEAUTIFUL HOG    WALLOW,
MICH."
42 Q$="ENTER A PICKLE EATING CONTEST IN    SCRATCHY
BACK,TENN."
47 REM ETC.
58 Z$="BUY A COSY YACHT AND CRUISE THE CARIBBEAN SEA"
70 IF D<0.5 THEN PRINT M$:GOTO 90
71 IF D<1    THEN PRINT P$:GOTO 90
72 IF D<5    THEN PRINT Q$:GOTO 90
73 REM ETC.
86 IF D<1000000 THEN PRINT Z$:GOTO 90

```



```

88 PRINT "TREAT YOUR WHOLE SCHOOL TO A 'ROUND THE WORLD
TRIP!"
90 REM ENDING OF PROGRAM

```

A16-1

```

10 REM RANDOM NAME
12 PRINT "clear"
15 DIM N$(40)
30 PRINT "YOUR NAME ";
32 INPUT N$
35 PRINT "clear"
36 POKE 752,1:REM CURSOR OFF
40 X=INT(RND(9)*30)
41 Y=INT(RND(9)*23)
50 POSITION X,Y
60 PRINT N$
70 FOR T=1 TO 50:NEXT T
90 GOTO 40
99 REM MINDA SAYS "MODEST AREN'T WE!"

```

A16-2

```

10 REM X NAME
12 PRINT "clear"
15 DIM N$(40)
22 POSITION 14,6:PRINT "M"
23 POSITION 26,6:PRINT "M"
32 POSITION 17,9:PRINT "I"
33 POSITION 23,9:PRINT "I"
42 POSITION 20,12:PRINT "N"
43 REM MIDDLE LETTER, DO FIRST
52 POSITION 23,15:PRINT "D"
53 POSITION 17,15:PRINT "D"
62 POSITION 14,18:PRINT "A"
63 POSITION 26,18:PRINT "A"

```

A17A-1

```

10 REM COUNTING BY FIVES
12 PRINT "clear"
20 FOR I=5 TO 100 STEP 5
30 PRINT I
35 FOR T=1 TO 200:NEXT T
40 NEXT I

```

A17B-2

```

10 REM SLIPPING NAME
12 PRINT "clear"
15 DIM N$(40)
20 PRINT "YOUR NAME":INPUT N$
25 ? "clear"
28 FOR I=0 TO 22
30 POSITION I,I
32 PRINT N$
35 FOR T=1 TO 100:NEXT T
40 NEXT I

```

A17B-3

```

10 REM CLIMBING NAME
12 PRINT "clear"
13 DIM N$(40)
15 N$="STANISLAUS MAZURSKI"
20 FOR I=23 TO 1 STEP -1
25 POSITION 10,I
26 PRINT N$;
27 FOR T=1 TO 50:NEXT T
30 PRINT "clear"
40 NEXT I

```

A17B-3

```

10 REM CROSSING FRIENDS
12 PRINT "clear"
15 DIM F$(20), Y$(20), B$(20)
20 PRINT "YOUR NAME":INPUT Y$
22 PRINT
24 PRINT "YOUR FRIEND'S NAME":INPUT F$
26 B$="      "
28 PRINT "clear"
30 REM
31 REM MAIN LOOP
32 REM
40 FOR I=1 TO 22
45 IF I>16 THEN GOTO 60
50 POSITION 2*I,12
53 PRINT B$
55 POSITION 2*I+2,12
57 PRINT Y$
60 POSITION 12,I

```

```

65 PRINT B$
70 POSITION 12,I+1
75 PRINT F$;
90 NEXT I
91 REM
92 REM DELAY AT END
93 REM
95 FOR T=1 TO 1000:NEXT T

```

A19-1

```

10 REM RELATIVES
12 PRINT "clear"
15 DIM R$(40), N$(40), W$(40)
20 PRINT "clear RELATION?"
21 PRINT
22 INPUT W$
23 PRINT
24 FLAG=0
29 RESTORE
30 READ R$:READ N$
31 IF R$="END" THEN GOTO 300
32 IF R$=W$ THEN GOSUB 200
39 GOTO 30
90 DATA FATHER, WILLIAM
91 DATA MOTHER, ANNE
92 DATA SISTER, JOAN
93 DATA SISTER, SUZAN
94 DATA GRANDFATHER, JOHN
95 DATA GRANDMOTHER, ADA
96 DATA GRANDMOTHER, VIVIAN
97 DATA UNCLE, FRED
98 DATA UNCLE, GEORGE
99 DATA AUNT, MARY
100 DATA COUSIN, ROGER
110 DATA END, END
200 REM
201 REM PRINT IT
202 REM
210 PRINT R$;" " ;N$
220 FLAG=1
299 RETURN
300 REM
301 REM NO RELATION
302 REM
310 IF FLAG=0 THEN PRINT "YOU DO NOT HAVE A " ;W$
320 FOR T=1 TO 1000:NEXT T
399 GOTO 20

```

A20-1

```

10 REM SONG
15 PRINT "clear"
20 FOR I=1 TO 10
22 READ P,D
24 SOUND 1,P,10,8
26 FOR T=1 TO D*100:NEXT T
28 SOUND 1,0,0,0
40 NEXT I
100 DATA
121,3,121,3,121,2,108,1,96,3,96,2,108,1,96,2,91,1,81,6

```

A21-5

```

2 GOTO 1000:REM SINBAD'S MAGIC CARPET
12 PRINT "clear"
198 REM
199 REM MAIN LOOP
200 REM
201 GRAPHICS 3+16
202 SETCOLOR 4,CB,BB
203 SETCOLOR 0,C1,B1
204 SETCOLOR 1,C2,B2
203 SETCOLOR 2,C3,B3
206 A=RND(9):B=RND(9)
207 C=RND(9)
210 FOR I=0 TO 19:FOR J=0 TO 19
211 L=INT(I*24/40)
212 K=I+J:Q=INT(K*24/40)
216 X=X+A*(I+3*B)/(J+3)+C*(20-I-J)/53
218 IF X>3 THEN X=X-3
219 IF X<1 THEN X=1
225 COLOR X
230 PLOT I,Q:PLOT K,L
232 PLOT 39-I,Q:PLOT K,23-Q
234 PLOT 39-K,L:PLOT I,23-Q
236 PLOT 39-I,23-Q:PLOT 39-K,23-L
290 NEXT J:NEXT I
800 FOR T=1 TO 2000:NEXT T
999 END
1000 REM
1001 REM SINBAD'S MAGIC CARPET
1002 REM
2000 PRINT "clear"
2010 PRINT "FIRST COLOR, BRIGHTNESS":INPUT C1,B1
2011 PRINT "SECOND COLOR, BRIGHTNESS":INPUT C2,B2
2112 PRINT "THIRD COLOR, BRIGHTNESS":INPUT C3,B3
2013 PRINT "BACKGROUND COLOR, BRIGHTNESS":INPUT CB,BB
2999 GOTO 200

```

A22-1

```

10 REM ALPHABETICAL
12 PRINT "clear"
15 DIM W$(40), H$(40)
17 POSITION 0,3
20 PRINT "THIS PROGRAM ARRANGES THE LETTERS"
21 PRINT "OF A WORD IN ALPHABETICAL ORDER,"
25 POSITION 0,6
30 PRINT "GIVE ME A WORD"
31 PRINT:INPUT W$
32 PRINT
35 L=LEN(W$)
39 K=1
40 FOR I=65 TO 65+26
41 REM TEST LETTERS IN ALPHABET
42 REM TO SEE IF IN WORD
45 FOR J=1 TO L
50 G=ASC(W$(J,J))
55 IF G=I THEN H$(K)=CHR$(G):K=K+1
60 NEXT J:NEXT I
70 PRNT "HERE IT IS IN ALPHABETICAL ORDER:"
75 PRINT
80 PRINT " ";H$

```

A22-2

```

10 REM $%! DOUBLE DUTCH !%$
12 PRINT "clear"
15 DIM S$(99), L$(1), SS$(99)
25 ?"GIVE ME A SENTENCE":?:INPUT S$
27 ?
30 L=LEN(S$)
40 K=1
50 FOR I=1 TO L
51 L$=S$(I,I)
52 IF L$="A" THEN GOTO 72
53 IF L$="E" THEN GOTO 72
54 IF L$="I" THEN GOTO 72
55 IF L$="O" THEN GOTO 72
56 IF L$="U" THEN GOTO 72
69 SS$(K)=L$
70 K=K+1
72 NEXT I
76 ?"HERE IT IS IN DOUBLE DUTCH"
78 ?
80 ? SS$

```

A23-1

```

10 REM MENU MAKER
12 PRINT "clear"
15 DIM C$(1)
17 OPEN #1,4,0,"K:"
19 POSITION 0,3
20 PRINT "WHICH LOLOR DO YOU LIKE?"
21 PRINT
22 PRINT "    <P>  PINK  "
23 PRINT "    <O>  ORANGE"
24 PRINT "    <G>  GREEN  "
25 PRINT "    <B>  BLUE  "
26 PRINT
30 GET #1,C:C$=CHR$(C)
35 IF C$="P" THEN C=4
36 IF C$="O" THEN C=2
37 IF C$="G" THEN C=12
38 IF C$="B" THEN C=8
40 SETCOLOR 4,C,8

```

A23-2

```

10 REM SILLY SENTENCES
12 PRINT "clear"
13 OPEN 6,4,0,"K:"
15 DIM Y$(1), L$(1), S$(99)
16 ?"SILLY SENTENCES":?
17 ?"WANT INSTRUCTIONS <Y/N>":?:GET #6,Y
18 Y$=CHR$(Y)
19 IF Y$="Y" THEN GOSUB 100
20 K=1
21 ?"THE SUBJECT: (END WITH A PERIOD)":?
22 GET #6,Y:L$=CHR$(Y)
24 IF L$="." THEN GOTO 30
28 S$(K)=L$:K=K+1
29 GOTO 22
30 S$(K)=" ":K=K+1
32 ?"THE VERB: (END WITH A PERIOD)":?
34 GET #6,Y:L$=CHR$(Y)
36 IF L$="." THEN GOTO 42
38 S$(K)=L$:K=K+1:GOTO 34
42 S$(K)=" ":K=K+1
50 ?"THE OBJECT: (END WITH A PERIOD)":?
52 GET #6,Y:L$=CHR$(Y)
54 IF L$="." THEN GOTO 70
56 S$(K)=L$:K=K+1:GOTO 52

```

```

70 S$(K)="."
85 ? S$
99 END
100 ?"clear"
110 ?"THREE PLAYERS ENTER PARTS OF A SENTENCE":?
115 ?"NO PLAYER CAN SEE WHAT THE OTHERS ENTER":?
120 ?"THE FIRST ENTERS THE SUBJECT"
121 ?"      (THE PERSON DOING SOMETHING)":?
125 ?"THE SECOND ENTERS THE VERB"
126 ?"      (THE ACTION WORD)":?
130 ?"THE THIRD ENTERS THE OBJECT"
131 ?"      (THE PERSON OR THING TO WHOM"
132 ?"      THE ACTION IS DONE)":?
150 FOR T=1 TO 4000:NEXT T
199 RETURN

```

A24B-1

```

10 REM GOSUB AND RETURN
12 PRINT "clear"
20 GOSUB 200:REM FIRST ONE
30 GOSUB 300:REM NEXT ONE
40 GOSUB 400:REM LAST ONE
50 GOSUB 200:REM AGAIN
99 END
200 REM
201 REM SUBROUTINE 1
202 REM
210 PRINT "LOOK OUT!"
215 PRINT
250 GOSUB 900
299 RETURN
300 REM
301 REM SUBROUTINE 2
302 REM
350 PRINT "RED SMOKE"
355 PRINT
360 GOSUB 900
399 RETURN
400 REM
401 REM LAST ONE
402 REM
450 PRINT "IS POURING FROM YOUR ATARI!"
455 PRINT
460 GOSUB 900
499 RETURN

```

```

900 REM
901 REM TIMER
902 REM
950 FOR T=1 TO 400:NEXT T
999 RETURN

```

A25-2

```

10 REM "ON ... GOTO" SAMPLE
12 OPEN #6,4,0,"K:"
20 REM MAKE A MENU
22 PRINT "clear"
25 POSITION 0,3
30 PRINT "MAKE YOUR CHOICE:"
31 PRINT:PRINT "  <A>  TAKE A NAP  "
32 PRINT:PRINT "  <B>  EAT AN APPLE  "
33 PRINT:PRINT "  <C>  CALL A FRIEND  "
40 PRINT:GET #6,X:PRINT
41 X=X-64
50 ON X GOTO 60,70,80
52 GOTO 22
60 PRINT "YOUR BED IS NOT MADE!"
61 END
60 PRINT "YOUR SISTER ATE THE LAST ONE!"
61 END
70 PRINT "YOUR FATHER IS ON THE PHONE!"
81 END

```

A26-1

```

10 REM CIPHER MAKER
12 PRINT "clear"
15 DIM S$(99), P$(2), Q$(2), L$(99)
20 PRINT "CODE MAKING PROGRAM "
21 PRINT
25 PRINT "ENTER A SENTENCE FOR CODING:"
26 PRINT
30 INPUT S$
35 L=LEN(S$)
36 S$(L+1)=" "
40 FOR I=1 TO L STEP 2
45 P$=S$(I,I+1)
50 Q$=P$(2)
51 Q$(2)=P$(1,1)
55 L$(I)=Q$
60 NEXT I

```



```

64 PRINT
65 PRINT "HERE IS THE CODED SENTENCE:"
66 PRINT
70 PRINT " ";L$

```

A26-2

```

10 REM QUESTION ANSWERER
12 PRINT"clear"
15 DIM Q$(99), C$(1), S$(40), V$(40), P$(99)
17 POSITION 0,3
20 PRINT "ENTER A QUESTION"
22 PRINT
25 INPUT Q$
27 L=LEN(Q$)
28 PRINT
30 REM TAKE OFF THE QUESTION MARK
32 Q$(L)=","
36 REM LOOK FOR THE END OF THE FIRST WORD
40 FOR I=1 TO L
41 C#=Q$(I,I)
45 IF C#=" " THEN S1=I:I=L
46 NEXT I
48 REM LOOK FOR THE END OF THE SECOND WORD
50 FOR I=S1 + 1 TO L
52 C#=Q$(I,I)
54 IF C#=" " THEN S2=I:I=L
56 NEXT I
58 REM TURN THE WORDS AROUND
60 S#=Q$(S1+1,S2)
62 V#=Q$(1,S1)
65 PRINT S#;V#;Q$(S2+1,L)

```

A26-3

```

10 REM PIG LATIN
12 DIM W$(40), L$(40)
15 PRINT "clear"
17 POSITION 0,3
20 PRINT "THIS IS A PIG LATIN PROGRAM"
25 PRINT
30 PRINT "GIVE ME A WORD"
31 PRINT
33 INPUT W$
34 L=LEN(W$)
35 PRINT

```

```

40 REM FIND THE FIRST VOWEL
41 FOR I=1 TO L
42 IF W$(I,I)="A" THEN 50
43 IF W$(I,I)="E" THEN 50
44 IF W$(I,I)="I" THEN 50
45 IF W$(I,I)="O" THEN 50
46 IF W$(I,I)="U" THEN 50
47 NEXT I
50 IF I<>1 THEN 60
51 L$=W$
52 L$(L+1)="LAY"
55 GOTO 80
60 REM FOUND IT
68 L$=W$(I)
70 L$(L-I+2)=W$(1,I-1)
72 L$(L+1)="AY"
80 PRINT " ";L$
90 FOR T=1 TO 1000:NEXT T
99 GOTO 15

```

A27-3

```

10 REM BACKWARD ADDED TO FORWARD
12 DIM N$(20), B$(20), A$(20), L$(10)
15 PRINT "clear"
17 POSITION 0,3
20 PRINT "GIVE ME A NUMBER"
21 INPUT N
22 N$=STR$(N)
35 L=LEN(N$)
40 FOR I=1 TO L
41 B=L-I+1
45 B$(I,I)=N$(B,B)
50 NEXT I
55 B=VAL(B$)
57 PRINT
60 PRINT " ";N
61 PRINT " +" ;B
62 L$="-----"
65 PRINT " ";L$(1,L+1)
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

```

A27-4

```

10 REM MARCHING NUMBERS
12 DIM B$(10), N$(10)
15 PRINT "clear"
17 POSITION 0,3
20 PRINT "GIVE ME A NUMBER "
21 PRINT
22 INPUT N
25 N$=STR$(N)
26 L=LEN(N$)
35 POSTION 0,12
36 PRINT N$
40 FOR I=1 TO 39-L
42 POSITION I-1,12
43 B$=N$(2,L)
44 B$(L)=N$(1,1)
45 N$=B$
46 PRINT " ";N$
50 FOR T=1 TO 100:NEXT T
60 NEXT I

```

A28-2

```

10 REM MAKE THESE CHANGES TO 'MOVE A SPOT'
70 IF X<1 THEN X=1:REM AT LEFT EDGE?
71 IF Y<1 THEN Y=1
72 IF X>38 THEN X=38
73 IF Y>22 THEN Y=22
74 REM MAKE THESE ADDITIONS TO 'MOVE A SPOT'
29 GOSUB 500:REM BORDER
500 REM BORDER
510 SETCOLOR 1,4,8
511 COLOR 2
515 FOR I=0 TO 39
520 PLOT I,0
521 PLOT I,23
530 NEXT I
540 FOR I=0 TO 23
545 PLOT 0,I
546 PLOT 39,I
550 NEXT I
599 RETURN

```

A29-2

```

10 REM SHOOTING STARS
11 REM MAKE THIS CHANGE
213 REM CHANGE LINE NUMBER 213 TO 216, YOU HAVE:
216 IF L=3 THEN GOSUB 400
401 REM ADD
440 COLOR 0:PLOT X,I:REM ERASES STAR
450 I=1:REM ENDS LOOP FOR TRAVEL OF LASER PHOTON

```

A30-1

```

10 REM ARRAYS
12 DIM D(12)
15 PRINT "clear down down down"
20 FOR I=1 TO 12
22 READ D
24 D(I)=D
29 NEXT I
30 PRINT "MONTH NUMBER? <1-12>"
31 PRINT:INPUT M
32 PRINT
35 PRINT "MONTH NUMBER ";M;" HAS ";D(M);" DAYS."
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31

```

A31-3

```

10 REM :::::::::::::: "AIN'T GOT NO ..." ::::::::::::::
12 DIM S$(99), W$(40), N$(40), L$(1)
15 PRINT "clear down down down"
19 REM ----- GET A SENTENCE
20 PRINT "ENTER A SENTENCE:"
22 PRINT:INPUT S$:PRINT
25 L=LEN(S$)
30 REM ----- REMOVE ENDING PUNCTUATION
31 L$=S$(L,L)
32 C=ASC(L$)
34 IF C<35 OR C>90 THEN S$(L,L)=" "
40 NN=0:REM NUMBER OF NEGATIVE WORDS
42 S2=1:REM START LETTER OF WORD
45 FOR I=1 TO L:REM ----- TEST WORDS IN SENTENCE
50 L$=S$(L,L):REM GET A CHARACTER
54 REM IS IT A SPACE?
55 IF L$=" " THEN S1=S2:S2=I+1:GOSUB 200
60 NEXT I:REM NEXT CHARACTER

```

```

65 PRINT:REM ----- PRINT RESULT
69 REM
70 IF NN=0 THEN PRINT "THIS SENTENCE IS POSITIVE,"
71 IF NN=1 THEN PRINT "THIS SENTENCE IS NEGATIVE,"
72 IF NN=2 THEN PRINT "THIS SENTENCE HAS A DOUBLE NEGATIVE,"
73 IF NN>2 THEN PRINT "THIS SENTENCE IS HARD TO UNDERSTAND!"
99 END
100 REM
101 REM ----- SOME TEST SENTENCES
102 REM
111 REM I DON'T EAT JUNK FOOD,
112 REM I NEVER EAT NO JUNK FOOD!
113 REM I DON'T NEVER EAT NO JUNK FOOD!
200 REM
201 REM ----- IS THE WORD NEGATIVE?
202 REM
205 RESTORE
210 W$=S(S1,S2-2):REM          PICK WORD FROM SENTENCE
215 READ N$ :REM              PICK NEGATIVE WORD
220 IF N$="END" THEN RETURN:REM LIST DONE?
225 IF N$=W$ THEN NN=NN+1:REM  COMPARE TWO WORDS
230 GOTO 215:REM              NEXT NEGATIVE WORD
900 REM
901 REM ----- NEGATIVE WORDS
902 REM
910 DATA NO,NOT,NEVER,NONE,NOTHING
911 DATA DON'T,DOESN'T,AREN'T,AIN'T
912 DATA ISN'T,DIDN'T
913 DATA HAVEN'T,HASN'T,HADN'T
914 DATA CAN'T,COULDN'T,SHOULDN'T
915 DATA WOULDN'T,WON'T
920 DATA END

```

A32-2

```

2 GOTO 1000:REM **** CODE--DECODE ****
100 REM
101 REM          MAIN LOOP
102 REM
110 GOSUB 400:REM          GET PASSWORD
115 PRINT "down CODE OR DECODE? <C/D>"
116 GET #6,Y:Y$=CHR$(Y)
120 IF Y$="C" THEN 500:REM CODE MESSAGE
125 IF Y$="D" THEN 600:REM DECODE MESSAGE
130 GOTO 115

```

```

199 END
400 REM
401 REM ----- FORM CODE ALPHABET
402 REM
405 PRINT "INPUT PASSWORD down"
406 INPUT PW$
407 REM REMOVE REPEATED LETTERS
408 F$=PW$(1,1)
410 FOR I=2 TO LEN(PW$)
411 L1$=PW$(I,I)
412 FOR J=1 TO LEN(F$)
415 L2$=F$(J,J)
420 IF L1$=L2$ THEN 430
421 NEXT J
422 F$(LEN(F$)+1)=L1$
430 NEXT I
432 PW$=F$
433 PRINT " down THE SHORTENED PASSWORD IS down"
434 PRINT " ";PW$
435 REM REMOVE PASSWORD LETTERS FROM THE ALPHABET
436 PW=LEN(PW$)
438 K=1
440 FOR I=1 TO 26:L1$=A$(I,I)
441 FLAG=0
442 FOR J=1 TO PW:L2$=PW$(J,J)
445 IF L1$=L2$ THEN FLAG=1
445 NEXT J
447 IF FLAG=0 THEN F$(K,K)=L1$:K=K+1
460 NEXT I
470 A$=PW$
471 A$(PW+1)=F$
475 PRINT "down ALPHABETS down"
480 ? A$; "CIPHER"
485 ? B$; "NORMAL"
499 RETURN
500 REM
501 REM ----- FORM A CODED MESSAGE
502 REM
505 ?"down INPUT MESSAGE, END WITH '*' down"
507 K=1
510 GET #6,Y:Y$=CHR$(Y)
515 IF Y$="*" THEN 590
520 IF Y<65 OR Y>90 THEN P$(K,K)=Y$:GOTO 540
525 Q=Y-64
530 P$(K,K)=A$(Q,Q)
540 K=K+1:Y$
589 GOTO 510

```

```

590 ?:"down";P$
599 END
600 REM
601 REM ----- DECODE A MESSAGE
602 REM
610 ?:"down TYPE IN THE CODED MESSAGE"
612 ?:"down END WITH A '*' SIGN down"
615 GET #6,Y:Y$=CHR$(Y)
617 IF Y$="*" THEN G90
620 FOR I=1 TO 26
625 IF Y$=A$(I,I) THEN ? B$(I,I);:GOTO 615
630 NEXT I
635 PRINT Y$:GOTO 615
690 END
1000 REM
1001 REM          **** CODE--DECODE ****
1002 REM
2003 PRINT "clear down"
2005 DIM A$(26),B$(26)
2007 OPEN #6,4,0,"K:"
2010 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2015 B$=A$
2020 DIM Y$(1),PW$(30),F$(30)
2021 DIM L1$(1),L2$(1),P$(99)
2999 GOTO 100

```

GLOSSARY

argument

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

Like:

INT(N)	has	N	as an argument
LEN (W\$)	has	W\$	as an argument

array

A set of variables that have the same name. The members of the array are numbered. The numbers appear in parentheses after the variable name. See dimension, subscript. Examples:

A(0)	is the first member of the array A
B(7)	is the eighth member of the array B
CD(3,I)	is a member of the array CD

arrow keys

Four keys on the ATARI computer that have arrows on them. They move the input cursor to the left, right, up, and down.

ASCII

Stands for American Standard Code For Information Interchange. Each character has an ASCII number. See ATASCII

assertion

The name of a phrase that can be TRUE or FALSE. The phrase we called "something A" in an IF statement is an assertion. An assertion is also a numerical expression. See expression, TRUE, FALSE, logic, IF, something A. Example:

the assertion	"A" < > "B"	is TRUE
the assertion	3 = 4	is FALSE

ATASCII

A 256 number code used by ATARI and similar to ASCII. Each letter, digit, punctuation mark, and graphics character is identified by a number.

attract mode

After being unattended for about 9 minutes, the computer screen display starts to cycle randomly through various colors. This helps even out the aging of the TV screen color dots.

background

The area of the screen inside the border.

BASIC

Beginners's All-purpose Symbolic Instruction Code. A language originated by John Kemeny and Thomas Kurtz at Dartmouth College in the early '60s.

bell

The early teletype machines had a bell (like the bell on a typewriter). The ATARI makes a "buzz" sound instead.

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

Means to start up the computer from scratch. An easy thing to do with modern computers that have start up programs stored permanently in ROM memory. It was an involved procedure in the early days. Now it usually means to read in the disk operating system programs (DOS) from a disk.

border

The outside area on the TV screen. How much shows depends on the age and adjustment of the TV.

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 jump, IF.

buffer

A storage area in memory for temporary storage of information being inputted or outputted from the computer.

buzzer

A sound the ATARI makes to attract your attention. You can sound the buzzer by an ESC CTRL 2 keypress in a PRINT command.

call

Using a GOSUB calls the subroutine. Putting a function in a statement calls the function. Call means the computer goes and does what commands are in the subroutine or does the calculation that the function is for.

carriage return

On a typewriter, you push the lever that moves the carriage carrying 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 line feed, CRLF.

character

Letters, digits, punctuation marks and the space are characters.

checksum

In some I/O operations, the computer adds together all the character numbers. The resulting sum is the "checksum." If the data was transmitted correctly, the checksum calculated after the data is received will agree with that calculated before the data was sent. See I/O.

clear

Means erase. Used in "clear the screen" and "clear memory."

coldstart

When first turned on, the computer loads certain parts of memory. Some of these will be changed as the computer is used. See warmstart.

column

Things arranged vertically. See row.

command

In BASIC a command makes the computer do some action, such as erase the screen and move the cursor to the upper left for the HOME command. See statement, expression. Some commands need expressions to be complete.

Example:

```
COLOR 2+1
```

concatenation

Means 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 line.

CRLF

Short for "carriage return followed by line feed." This is what is called just a "carriage return" on a typewriter. See carriage return, line feed.

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 ATARI:

INPUT cursor	a square on the screen
PRINT cursor	invisible, "shows" where next character will be printed

data

BASIC has two kinds of data: numerical and string. Logical data (TRUE, FALSE) are types of numerical data.

debug

Means to run a program to find the errors and fix them. You fix the errors by editing the program. See edit.

deferred execution

Means run a stored program. See immediate execution

delay loop

A part of the program that just uses up time and does nothing else. Example:

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

edit

there are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

enter

To put information into the computer by typing, then pushing the RETURN key. The information goes into the input buffer as it is typed. When RETURN is pushed, the computer uses the information.

erase

To destroy information in memory or write blanks to the screen. See clear.

error trap

Part of a program that checks for mistakes in information that the user has entered, or checks to see if computed results make sense.

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 value. Examples:

```
7*X+1  
"DOPE "<> N$
```

FALSE

the number 0. See logic, assertion.

fork in the road

Describes a branch point in the program. See branch.

function

BASIC has a number of functions built in. Each function has a name followed by parentheses. In the parentheses are one or more arguments. The function has a single value (numerical or string) determined by its arguments. See value, argument. The functions treated in this book are:

```
ASC, CHR$, INT, LEN, PEEK, RND, STR$, VAL
```

garbage

A random mess of characters in memory. Usually due to human or machine error.

graphic control characters

The graphics characters you get by pressing CTRL and a letter key. They can be used for graphics in mode 0 and modes 1 and 2 if CHBAS (756 decimal) is modified.

graphics

Means picture drawing.

hue

One of 16 colors that the ATARI hardware can display. See luminance.

immediate execution

When a line that does not start with a number is entered from the edit mode, it is executed as soon as the RETURN key is pressed. If the line has only one command, you usually think of it as "entering a command." But the line may have several statements separated by colons, and thus it is a little program. Then you think of it as "executing a program in the immediate mode".

index

An array name is followed by one or more numbers or numerical variables in parentheses. Each number is an index. Another word for index is "subscript".

integers

The whole numbers, positive, negative and zero.

INVERSE

Printing on the screen as black letters on white background.

I/O

Input/Output. Input from keyboard, tape recorder, etc. Output to screen, printer, tape recorder, etc.

joy stick

A device used in games. It is like the control stick used in early airplanes. It can detect 8 different directions, as well as "centered."

jump

The GOTO command makes the computer jump to another line in the program, rather than execute the next line.

line

There are two kinds of lines in BASIC: Lines that start with a number are stored in the program in memory. Lines that do not start with a number are executed right away (see immediate execution). A line contains one or more statements, separated by colons.

Parts of a line:

```
16 IF 7<=INT(Z) THEN PRINT LEN(Q$)+2;"RAT":GOTO 40
```

16	line number
IF 7<=INT(Z) THEN ... "RAT"	first statement
GOTO 40	second statement
IF 7<=INT(Z) THEN	a command
PRINT LEN(Q\$)=2;"RAT"	a command
GOTO 40	a command
7<=INT(Z)	an assertion
7<=INT(Z)	an expression
LEN(Q\$)+2	an expression
INT(Z)	a function
LEN(Q\$)	a function
Z, Q\$	arguments
Z, Q\$	variables
7, 2, "RAT"	constants
<=, +	operations

line buffer

The storage space that receives the characters you type in. See buffer.

line editing

Retyping parts of a line to correct it. You do this by moving the cursor to the wrong part and then typing the correct characters. Store the corrected line in memory by pressing RETURN.

line feed

Moving the cursor straight down to the next line. The ASCII number 10 signals this command to the screen or printer. See carriage return and CRLF.

line numbers

The number at the beginning of a program line. The line number tells the computer where to store the line. Some lines don't have numbers (the ones that will be executed in the immediate execution mode).

listing

A list of all the lines in a program.

load

To transfer the information in a file on disk or tape to the memory of the computer by using the LOAD or CLOAD command.

luminance

Called "brightness" in this book, it takes on 8 values (even numbers from 0 to 14). It is not quite the same as brightness because the "brighter" dots are also "whiter."

logic

The part of a program that compares numbers or strings. The relations AND, OR, and NOT, and =, <>, <, >, <=, and >= are used. See assertion, IF, something A.

loop

A part of the program that is done over and over again. There are two kinds of loops: FOR...NEXT loops, and "home made" loops that use IF... commands with GOTO commands.

loop variable

Is the number that changes as the loop is repeated. For example:

```
FOR I=1 TO 5:NEXT I      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 a label on the front and the 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 PRINT statement that tells what is expected in an INPUT statement. Example:

```
PRINT "AGE?":INPUT A
```

missile

Part of the "missile-player" display hardware system of ATARI computers. We have not covered this advanced system in this book. The reader is referred to the book DE RE ATARI.

monitor

Has two meanings. We use it to mean a box with a TV type screen that is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

nesting

When one thing is inside of another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

<code>L = INT (LEN (P\$) + 3 , 4)</code>	nested functions
<code>X = 5 * (6 + (7 * (8 + K)))</code>	nested parentheses

number

Is one type of information in BASIC. The numbers are generally decimal numbers. See integer, strings.

operation

In arithmetic: addition, subtraction, multiplication and division, with symbols + , - , * , and / . The only operation for strings is concatenation.

paddle

A game device which has a knob to turn. Not discussed in this book.

pixel

Picture Element. The smallest dot that is placed on the screen in a given GRAPHICS mode.

player-missile graphics

An advanced Atari hardware system for producing moving graphics. See missile.

pointer

A number in memory that tells where in a list of DATA you are at the present moment.

program

There are two kinds. The usual program is a list of numbered lines containing statements. The computer executes the statements (commands) in order when the RUN command is entered. The program is stored in a special part of memory, and only one program can be stored at a time.

A one line program can be entered when the computer is in the edit mode. It does not start with a line number and runs as soon as you press the RETURN key. This one line program does not get mixed with the stored program. But when it runs, it may read or change the variables (if any) that the stored program made when it ran.

prompt

Is a little message you put on the screen with an INPUT to remind the user what kind of an answer you expect. Its name comes from the hint that actors in a play get from the prompter if they forget their lines.

pseudo-random

A number that is calculated in secret by the computer using the RND() function. It is usually called a "random number". Pseudo-random emphasizes that the number really is not random (since it is calculated by a known method) but just is not predictable by the computer user.

punctuation

The characters like period, comma, /, ?, !, \$, etc.

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 10 times.

remark

A comment you make in the program by putting it in a REM statement. Example:

```
REM THE BORDER SETUP SUBROUTINE
```

reserved words

A list of words and abbreviations that BASIC recognizes as commands or functions. The reserved words cannot be used in variable names.

return a value

When a function is used (called), its spot in the expression is replaced with a value (a number or a string). This is called "returning a value."

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.

row

Things arranged horizontally (across).

screen

The TV screen or a similar one in a monitor that is hooked up to the computer. See monitor.

scrolling

The usual way an ATARI 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".

something A

Is a phrase in this book that stands for an assertion in an IF statement. See assertion, IF. Example:

```
IF A>4 THEN GOSUB 500           A>4 is "something A"
```


split screen graphics

Each GRAPHICS mode except mode 0 has an option where 4 lines are reserved at the bottom of the screen for text to be written.

stack

Is a data type used in machine language programming. The 6502 processor has a stack in page \$01 and it holds information about loops, subroutines, and nesting.

starting stuff

is the name given in this book to initialization material in a program. It includes REM's for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things that need be done just once at the beginning of a program run.

statement

The smallest complete section of a program. It starts with a command. The command may have expressions in it.

store

To put information in memory or to save a file on a disk.

string

A type of data in BASIC. It consists of a row of characters. See number.

subroutine

A section of a program that starts with a line called from a GOSUB command and ends with a RETURN command. 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 index.

syntax

Means the way a statement in BASIC is spelled. A syntax error means the spelling of a command or variable name is wrong, the punctuation is wrong or the order of parts in the line is wrong.

text window

See split screen graphics.

timing loop

A loop that does nothing except use up time. See delay loop.

title

The name of a program or subroutine. Put it in a REM statement.

TRUE

Has the value 1. See logic, FALSE, assertion.

typing

Pressing keys on the ATARI. It is different from “entering”. See enter.

value

The value of a variable is the number or string stored in the memory box belonging to the variable. See variable.

variable

A name given to a “box” in memory. The box holds a value. When the computer sees a variable name in an expression, it goes to the box and takes a copy of what is in the box back to the expression and puts it where the variable name was, and continues to evaluate the expression. See variable name.

variable name

A variable is either a string variable or a numerical variable. The name tells which. String variables have names ending in a “\$” sign. Numerical variables do not.

warmstart

After turning on the machine and using it, you may “get into a mess” and want to start over. Pressing the SYSTEM RESET key will re-initialize most of the memory, but not erase the BASIC program you have in memory. See coldstart.

INDEX

INDEX OF COMMANDS AND FUNCTIONS EXPLAINED IN THIS BOOK

AND 161	NEW 14, 24, 26, 45
ASC 118 - 121	NEXT 64, 92, 95
CLOAD 78, 81	NOT 164
CHR\$() 118, 120 - 122	ON 132, 135
CLR 155, 157	OPEN 120, 124
COLOR 112, 114, 129, 152	OR 161, 165
CONT 172, 175	PEEK 134
CSAVE 78, 80, 82	PLOT 112, 116, 117, 149
DATA 102 - 107, 111	POKE 90, 132
DIM 33, 34, 36 - 40, 58, 136, 137, 155 - 157, 159	POSITION 88, 89, 91, 96, 145, 169
DRAWTO 112, 116, 117	PRINT 13, 15, 17, 19, 21, 23, 27, 34, 37, 38,
END 126 - 128, 172, 173	43 - 47, 50, 57, 58, 62, 63, 70, 84, 91, 100, 127,
FOR...TO 92, 94, 95	172, 175, 176
GET 103, 118, 120, 122 - 125, 132, 133, 135, 170	READ 102 - 106
GOSUB 94, 126, 127, 129, 132, 166	REM 14, 21, 26, 27, 58, 85, 86, 126, 167, 171
GOTO 41, 47, 48, 50, 51, 67, 83, 86, 126, 127	RESTORE 102, 105
GRAPHICS 38, 46, 91, 112 - 114, 116, 145	RND 73, 74, 77, 83, 91
IF...THEN 49, 51 - 54, 56, 68, 83, 85, 86, 128	RUN 48, 97 - 99, 101
131, 160, 163	SETCOLOR 38, 112, 113, 152, 153
INPUT 38 - 40, 42 - 44, 47, 50, 56, 58, 77, 103,	SOUND 64, 66, 107, 109
122, 123, 132, 137	STEP 92, 94
INT () 73, 75	STICK 145, 146, 148
LEN 136	STRIG 145, 146, 149
LET 33, 35, 37 - 39, 50, 57, 83, 84, 103, 175	STOP 131, 172 - 174, 176
LIST 23, 32, 85, 87, 97, 175	STR\$() 142, 143
LOCATE 149, 151	

INDEX OF KEYS USED IN THIS BOOK

BREAK 47 - 49, 90, 100, 120, 146, 150, 172, 174
CLEAR 10, 19, 33
INSERT 28, 31 - 33
DELETE BACK S 28, 29, 31 - 33
ATARI KEY 18, 19, 91
RETURN 11 - 14, 16, 18, 30, 32, 33, 40, 97, 99, 120, 122, 123, 126, 127, 166
ARROW KEYS 29, 32
SHIFT 10, 12, 19, 33, 120, 132, 135
ESC 19, 33
CTRL 15, 16, 28 - 31, 33, 88, 91, 120, 132, 135
SYSTEM RESET 46, 49, 50, 150

INDEX OF ERROR MESSAGES

Error Code No.	Error Code Message and Meaning
2	Memory Insufficient Your program has grown too large to hold in memory.
3	Value Error You have put a negative number where a positive one is needed. Or the number is too large.
4	Too Many Variables You are allowed up to 128 different variable names.
5	String Length Error You tried to make a string longer than your DIM statement allowed.
6	Out of Data Error You tried to READ more items than were in your DATA statements.
7	Number greater than 32767 You have a negative number where a positive one is needed, or you have an integer larger than 32767.
8	Input Statement Error You tried to INPUT a letter or punctuation into a numerical variable.
9	Array or String DIM Error You did one of these things: You forgot to do a DIM command. You did DIM twice. You used a number larger than 32767 in DIM. You used a negative number in DIM. You used a larger number in a variable than the DIM allowed. You used a negative number in DIM.
10	Argument Stack Overflow You have used too many nested GOSUBs or too large an expression.
11	Floating Point Overflow or Underflow Error You tried to divide by zero. Or you tried to use a number that was too large.
12	Line Not Found You did a GOTO, GOSUB, or THEN to a line that you forgot to put into the program.
13	No Matching FOR Statement Your program reached a NEXT statement before it came to the FOR that belongs to it.

- 14 Line Too Long Error**
You wrote a line that was too complicated or too long.
- 15 GOSUB or FOR Line Deleted**
Your program reached a RETURN statement but you had deleted its GOSUB since the last run. Or you reached a NEXT statement but you had deleted its FOR since the last run.
- 16 RETURN Error**
Your program reached a RETURN before it reached its GOSUB.
- 17 Garbage Error**
Your program tried to use some memory that had “garbage” stored in it. The bad bytes may have come from a POKE that put stuff in the wrong place. You may be able to heal the hurt by doing a SYSTEM RESET and then taking the POKE statements out. If this doesn’t work, you will have to turn off, then on. You will lose the program in memory.
- 18 Invalid String Character**
You used a string that has an invalid first character. Or in a VAL function you used a string that was not all numerical digits.

INDEX

A

arithmetic 59,60
arrays 155,156,157
arrow keys 28,29,30
Atari key 18,91
ATASCII 118,119

B

backspace 28,29
bells and whistles 15
break key 47,48,49
bugs 176
buzzer 15,16,19

C

calculator 97,100
CLEAR 10,155,157
colon 83,85,86
color graphics 38,41,112,113,114
comma 43,45
concatenation 136,139
CONT command 173,175
count-down loops 94
CTRL (control) 16,30,31
cursor 10
 keys 28,29
 invisible 90

D

DATA 102,106
debugging 176
deferred execution 98
delay loops 64,65
delete 28,31
DIM 34,36,39,155,156

E

edit mode 97,99,100
empty line 16
END 126,127,128
equal 33,55,63
erase 10,26
error traps 170
ESC key 19
execution 98

F

false 160,161,162
FOR/NEXT loops 92,93
forks 54
functions 143,144,145

G	
GET.....	122,123
gluing	137
GOSUB.....	126,127,128
GOTO.....	47,48
graphics	43,46,149,150,151
color	38,41,112,113,114
jumps	90,91
keyboard	91
H	
hidden cursor.....	43,44
I	
IF/THEN	51,52,68,69
immediate execution	99
INPUT	38,40
insert	28,31
integer.....	73,75,77
inverse.....	18
J	
joy stick	145,146
jumps	
graphics	90,91
program	48,49
K	
keyboard	132
L	
LEN	136
LET.....	33,37,40
lines	
add	24
edit	25,30
erase	10,26,90
LIST.....	22,121
logic.....	157,165
loops	
count-down.....	94
FOR/NEXT.....	92,93
nested.....	94,95
variables.....	95
lower case.....	15,19
M	
memory.....	12,23,34-36,39,58
N	
nested loops	94,95
NEW	11,14
not equal	55
numbering lines.....	14
numbers.....	57,68,69,142,143
numerical variables	60

O	
ON/GOTO	132
OPEN	124,125
P	
PEEK	134,
pictures	27
pitch	109
pixel	149
PLOT command	115,116
POKE	133
position	88,89
PRINT	13,43,44
program	12,13
Q	
question mark	83,84
R	
random numbers	73,74,77
READ	102,104
REM	14,26
RESTORE	105
RETURN	126
RUN	13
run mode	97,99,101
S	
SAVE to tape	78,79
scrolling	169
semi-colon	43,45
shortcuts	83,84,85
snipping and gluing (strings)	137
sound	66,107,108,110
strings	136,142,143
constant	17
substrings	136,137
variables	34
STOP	131,173,174
subroutines	125,126,128
system reset	49
T	
tape	
load to	81
save to	78,79
true	160,161,162
U	
user friendly programs	166,167,168
Z	
zero	12



RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia



DATAMOSTTM
INC

ISBN 0-8359-3670-

8943 Fullbright Ave., Chatsworth, CA 91311. (213) 709-1202