
NOT JUST FOR KIDS!
EDWARD H. CARLSON
$19.95
KIDS & THE IBM PC / PCjr
IBM
POW
5 FOR Y=1 TO 5
10 FOR X=1 TO 255
20 PRINT CHR(X);
30 NEXT X
40 NEXT Y
50 END

KIDS AND THE IBM-PC/PCjr

KIDS AND THE IBM-PC/PCjr

by

Edward H. Carlson

Department of Physics and Astronomy
Michigan State University

Illustrated by
Paul D. Trap

DATAMOST™

8943 Fullbright Ave., Chatsworth, CA 91311-2750
(818) 709-1202

ISBN 0-88190-265-9

TABLE OF CONTENTS

INTRODUCTION

GRAPHICS, GAMES AND ALL THAT

ADVANCED PROGRAMMING

ACKNOWLEDGEMENTS

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

This book is the seventh in a series that started with KIDS AND THE APPLE. Each book has been written to fit the strengths of the computer in question and modified in response to what I have learned about teaching children as time has gone by.

I helped prepare and teach in "The Computer Camp" summer camp at Michigan State University for the last three summers. I am deeply grateful to my fellow staff members at the summer camp: Mark Lardie, Mary Winter, John Forsyth, and Marc Van Wormer, each of whom shared their experiences with me and helped provide insight into the children's minds.

I thank Kevin Forsyth for helping in the preliminary work on this book, for pointing out features unique to the IBM, and for rewriting many of the assignments to run on this machine.

It is a pleasure to acknowledge the contributions of Dave Gordon, a remarkable publisher and valued friend. I thank all his staff at DATAMOST, especially Arlon Dorman, who guides the books to rapid completion, and Marcia Carrozzo, who carefully edits and assembles these books.

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

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

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

TO THE KIDS

This book teaches you how to write programs for the IBM 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.

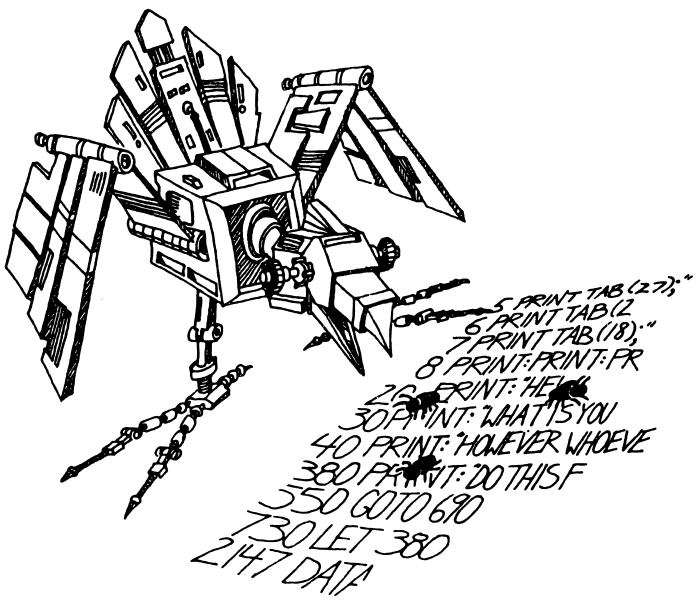
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

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, go back and reread the lesson carefully, from the top. You may have overlooked some detail. After trying hard to get unstuck by yourself, you may go ask a parent or teacher for help.

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

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



TO THE PARENTS

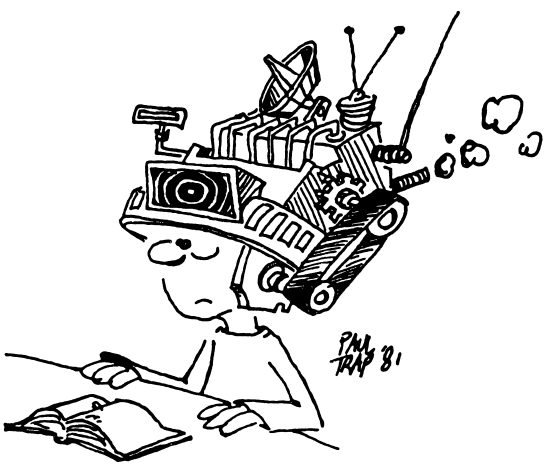
This book is designed to teach BASIC on the IBM to youngsters in the range from 10 to 14 years old. It gives guidance, explanations, exercises, reviews and "quizzes." Some exercises have room for the student to write in answers that you can check later. Answers are provided in the back of the book for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

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

Learning to program is not easy because it requires handling some sophisticated concepts, as well as 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 This Book: The book is divided into 33 lessons for the kids to do. Each lesson is preceded by a NOTES section which you should read. It outlines the things to be studied, gives some helpful hints and provides questions which you can use verbally (usually at the computer) to see if the skills and concepts have been mastered.

These notes are intended for the parents, but 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 working.



TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches BASIC and the features of the IBM computer.

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 provide 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: "chunking" ideas into "mind-sized bites", organizing such modules in a hierarchial system, looping to repeat modules, and conditional testing (the IF...THEN statement).

Verbal and visual metaphor is used to make the new material familiar to the student. Each concept is tied to the student's everyday experiences through the choice of examples, the language chosen to express the idea and through cartoons.

ABOUT PROGRAMMING

There is a common misconception about programming a computer. Many people think that ability in mathmatics is required — not so. Computing is analogous to the childhood activities of playing with building blocks and writing an English composition.

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

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

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

At larger scales, creativity comes into full scope and many other latent abilities of the child are developed. School skills such as arithmetic and language arts are utilized as needed, and thus strenghened. But the strongest features of programming 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).

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

ABOUT THE BOOK

This book is written for the IBM-PC and PC*jr* computers. Most of the book describes features present in cassette BASIC, which is "functionally equivalent" on the two machines. Some commands, such as those for music and graphics, are those described for the Advanced BASIC of the PC. Since the commands in the Cartridge BASIC of the PC*jr* are a "superset" of Advanced BASIC, the description should be correct there also.

The book was written while using an IBM-PC II with 64K of memory on the main board and a 320 K double sided disk drive running DOS 1.1 and Advanced BASIC. The monitors were a Sanyo color and a Zenith green screen interfaced to the IBM Color/Graphics Monitor Adapter.

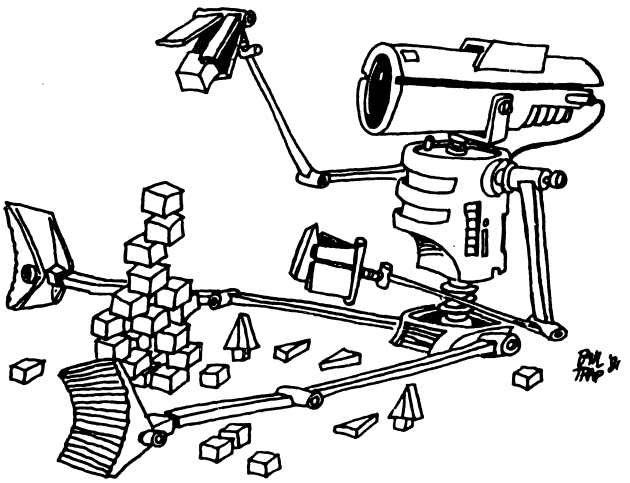
Refer to the IBM manuals if you have a difficulty that may be due to your computer's differing from the PC configuration stated above.

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 6, 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, debugging and user friendly programming.

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

Lessons 14: SAVING TO DISK, can be studied any time after the first lesson.



INSTRUCTOR NOTES 1 NEW, PRINT, REM AND RUN

There are many little questions your student may have in the beginning, so pull up a chair and help in the familiarization.

If you have a PC*jr*, some features of the keyboard may differ from those described in this book. Use the Keyboard Adventure program built into your machine to learn about your keyboard.

You cannot damage the computer with what you type in. If something goes wrong and all else fails, turn the computer off, wait six seconds, then turn it on and start again.

Instructions are given for starting up in Advanced BASIC with a disk you prepare for the student's exclusive use. Instructions for making the disk are given in the appendix "Disk Care." If you want different start-up instructions, cross out inappropriate lines and fill in the blank lines in the first lesson.

Help your student find the Ctrl, Enter, Home, Shift and quote keys.

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. Enter key.
3. Commands NEW, REM, PRINT and RUN.
4. What is a program. Numbered lines.
5. The cursor is sent home with the Home key.
6. The screen can be cleared using Ctrl-Home keys.
7. Memory can be cleared with NEW.
8. What is seen on the screen and what is in memory are different. At first, this may be a hard concept for the student to understand.
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. Make the program disappear from the TV screen but stay in memory.

3. RUN it.

4. Erase the program from memory.

5. Write a program that will print HELLO.

6. Make it run.

7. Erase it from memory but leave it on the screen.

LESSON 1 NEW, PRINT, REM AND RUN

HOW TO GET STARTED.

Open the disk drives and take out any disks, put them into their envelopes and put them away.

Put your STUDENT disk into the drive (drive A if you have two drives). Reach back on the right side of the computer and turn it on. You will see a small flashing light on the screen. You will hear the disk drive start up and then stop.

You will see a message. The last word is:

```
Ok
```

Below Ok is a flashing line. This line is called the "cursor." When you see it flashing, it means the computer is waiting for you to type something in.

Cursor means runner. The little line 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.

HOME

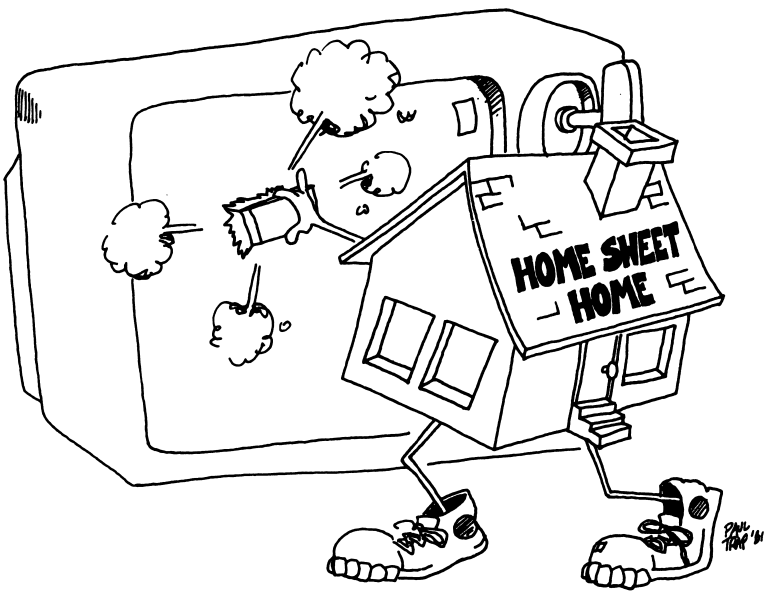
Press the "Home" key. It is on the "7" key on the right side of the keyboard. The cursor jumps to its home in the top left corner of the screen.

ERASING THE SCREEN

Type some more. It covers what you typed before. The screen is a mess. Let's erase it.

Two keys used together erase the screen.

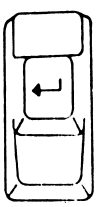
Do this: Find the "Ctrl" key. It is beside the "A" on the left end of the keyboard. Hold down the Ctrl key and press the Home key. The screen is erased.



COMMAND THE COMPUTER

Type: give me candy

and press the "Enter" key. The Enter key is next to the Home key and has a bent arrow on it like this:



(If you make a mistake, press the Ctrl and Home keys to clear the screen and start over.)

The computer printed

```
Syntax error
Ok
```

When the computer prints Syntax error, it means the computer did not understand you.

The computer understands only about 160 words. You need to learn which words the computer understands.

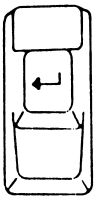
Here are the first four words to learn:

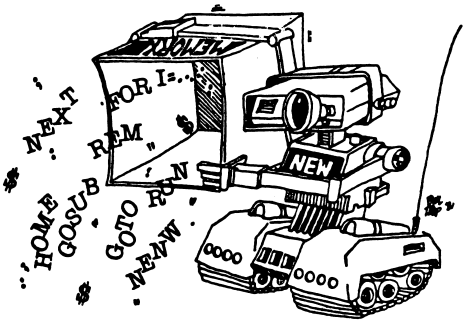
NEW PRINT REM RUN

THE NEW COMMAND

Type: new

and press ENTER.



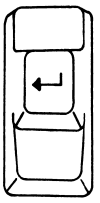


NEW empties the computer's memory so you can put your program in it.

HOW TO ENTER A LINE

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

1) type a line
2) then press the Enter key.



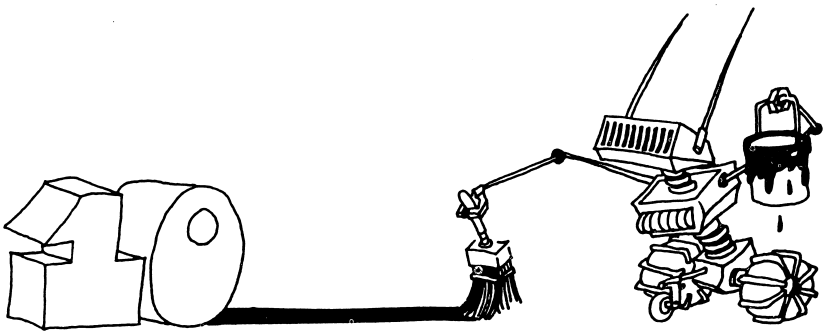
Clear the screen and enter this line:

```
10 print "hi"
```

The " marks are quotation marks. To make " marks, find a Shift key. There are two. Each has a fat arrow pointing upward. Hold down a Shift key and press the key that has the " on it. The " key is two keys to the right of the "L" key.

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

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



THE NUMBER 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 1O print "hi"

WHAT IS A PROGRAM?

A program is a list of commands for 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.

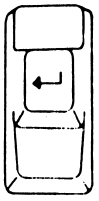
HOW TO RUN A PROGRAM

A moment ago you put this program into memory:

```
10 print "hi"
```

Now enter: run

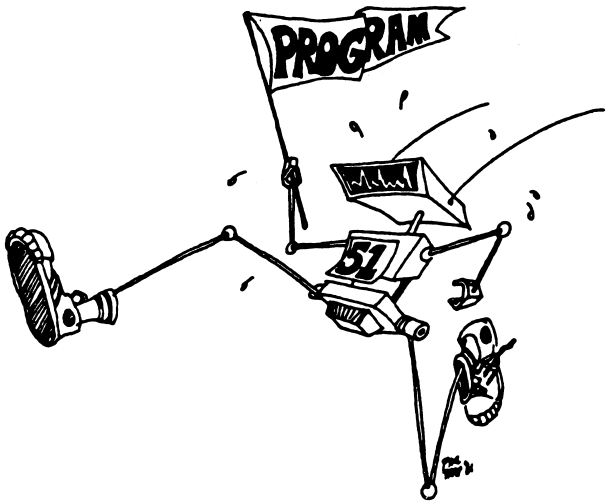
(Did you remember to press the ENTER key?)



The RUN command tells the computer to look into 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
Ok
```



HOW TO NUMBER THE LINES IN A PROGRAM

Clear the screen. Command new to empty memory. Enter this program:

```
1 rem hello
2 print "hi"
3 print "friend"
```

This program has three lines. Each line starts with a command. You have already learned the PRINT command.

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

Usually you will skip numbers when writing the program.

Like this:

```
10 rem hello
15 print "hi"
20 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.

RUN the program you have entered. The computer does the commands in the lines. It starts with the lowest line number and goes down the list in order.

Assignment 1:

1. Use two keys to erase the screen.

2. Use the command NEW. Explain what it does.

3. Write a program that uses REM once and PRINT twice. Then use the command RUN to make the program obey the commands.

INSTRUCTOR NOTES 2 BEEPS AND STRINGS

The BEEP statement makes the computer "beep." We wish to make plenty of "bells and whistles" available to the student to increase program richness.

The CLS command used in a program clears the screen and homes the cursor, just as the Ctrl-Home keys do from the keyboard.

The COLOR statement changes both the foreground and the background colors of the characters printed from that point on. It doesn't change any characters already on the screen. The third number in the COLOR statement changes the border color.

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.

QUESTIONS:

1. How do you do each of these things:

   Make the computer "beep"?
   Erase the screen?
   Empty the memory?
   Print your name?

2. What is a "string"?

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

4. What does the computer mean when it prints Syntax error?

5. Write a program to print FIRE on a red background and make the computer beep.

LESSON 2 BEEPS AND STRINGS

Enter: new (did you press the Enter key?)

Press the Ctrl Home keys

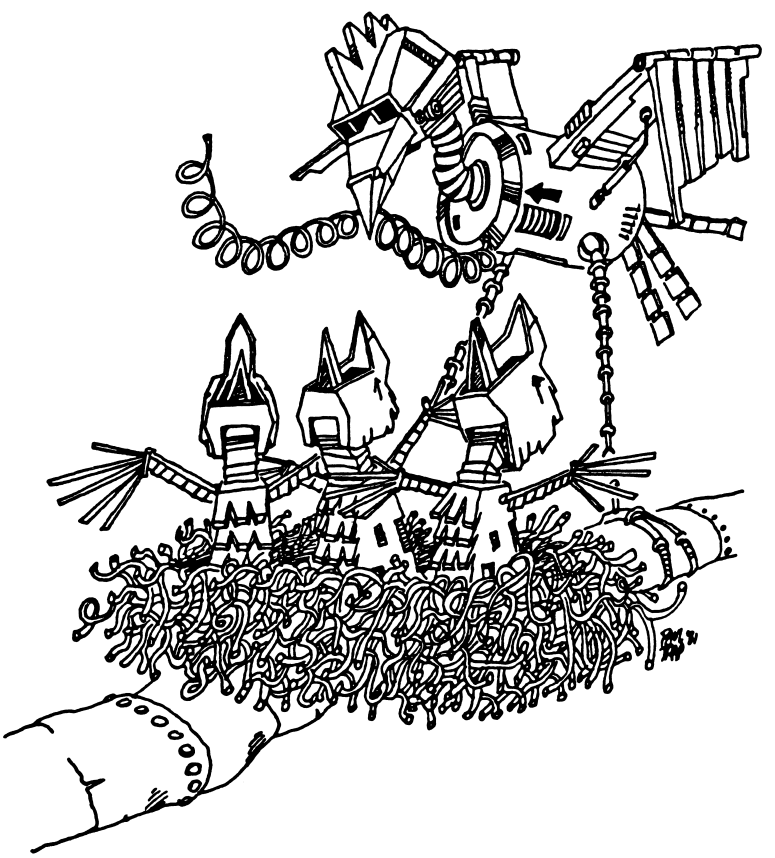
NEW empties the memory and the Ctrl Home keys erase the screen. You are ready to start this lesson.

THE COMPUTER BEEPS LIKE A BIRD.

Enter this program:

```
10 rem bird
20 cls
25 print " ---O---"
30 beep
run
```

Did the computer "beep"? Line 30 makes the computer beep.



CLEARING THE SCREEN IN A PROGRAM

The CLS statement clears the screen. Example:

RUN:

```
10 rem wipe that smile off!
20 print "Smile"
30 beep
40 cls
50 print "That is better"
```

PRINTING AN EMPTY LINE

RUN this:

```
10 rem skipping
15 cls
20 print "here is the first line"
30 print
40 print "one line was skipped"
```

Line 30 just prints a blank line.

COLOR THE SCREEN RED

RUN this:

```
10 REM color screen
20 COLOR 0,4
```

Now hold down the Enter key until the whole screen is red.

If your screen didn't turn red, try adjusting your color monitor.

To get other colors, try other numbers from 1 to 7 in place of the 4 in the COLOR command. (Leave the other number zero for now.)

PUT A FRAME AROUND THE SCREEN

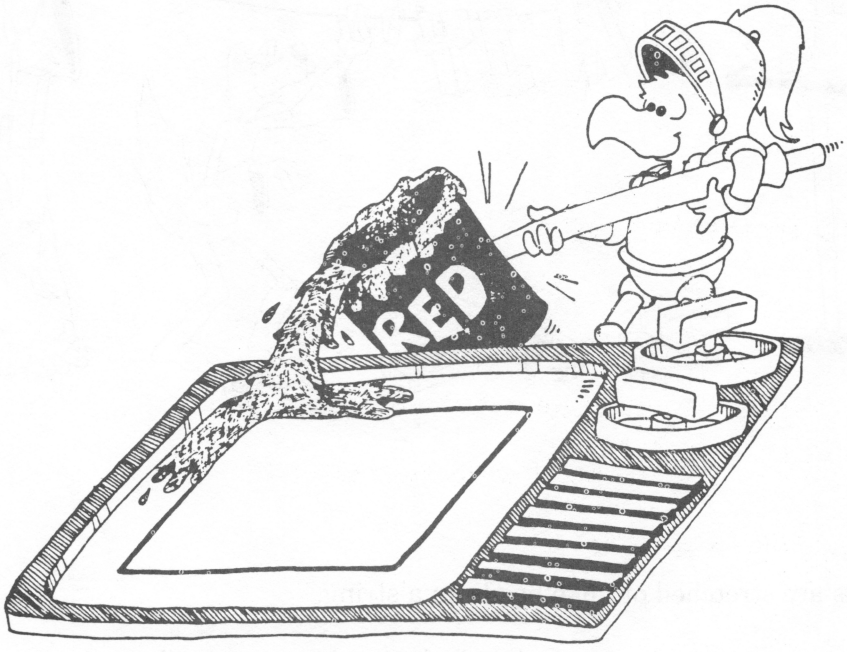
RUN this:

```
10 rem color border
20 color 0,4,3
```

Try other numbers from 0 to 7 in place of the "3".

BACK TO BLACK AND WHITE

RUN: 10 color 7,0



STRING CONSTANTS

Look at these PRINT statements:

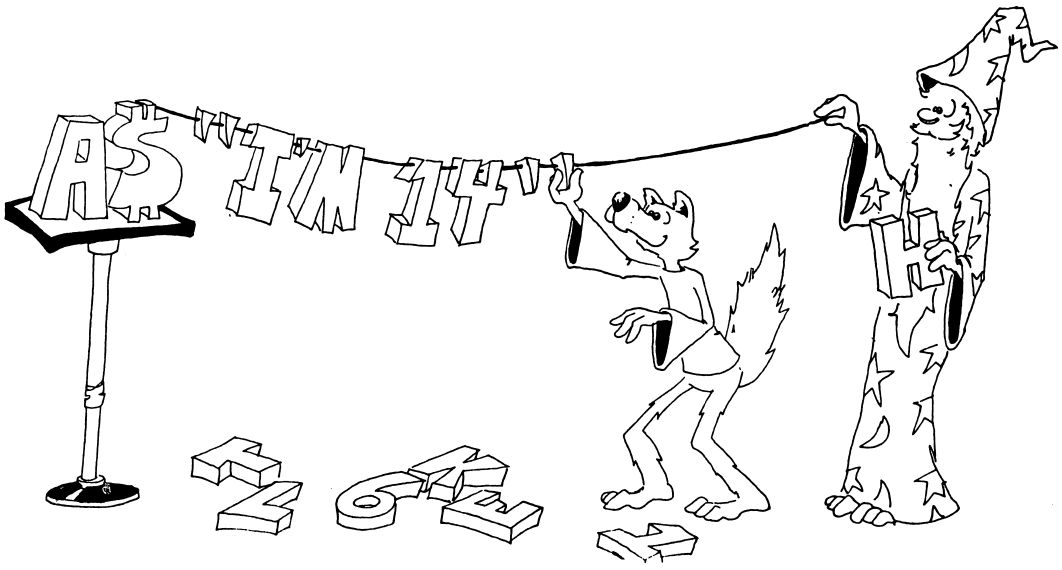
```
PRINT "Joe"
PRINT "#s47%*$"
PRINT "19"
PRINT "3.14159265"
PRINT "I'm 14"
PRINT " "
```

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

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

```
10 PRINT " "
```

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 all in a row.

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

Assignment 2:

1. Write a program that prints your first, middle, and last names. Make the letters black on a blue screen.

2. Now add a "beep" before it prints each name.

3. Now make the screen border change color before it prints each name.

INSTRUCTOR NOTES 3 LIST AND MEMORY

In this lesson:

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

Your student needs to understand that the program is stored in memory, even when it is not visible on the screen, and that LIST just lists the program to the screen. The special uses like LIST 100-300 and LIST -300 will be taken up later.

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 all of the program in memory on the screen?

3. How can you replace a wrong line with a corrected one?

4. Suppose you want to put a line in between two lines you already have in memory. How do you do this?

5. Explain how the computer puts program lines in "boxes" in memory. What does it write on the front of the box?

LESSON 3 LIST AND MEMORY

Start each lesson with NEW to erase the memory, and press the Ctrl Home keys to erase the screen.

Now enter:

```
10 rem house
20 print "listen"
30 beep
40 print "Did you hear the doorbell?"
```

Run this four line program. Then press Ctrl Home to erase 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.

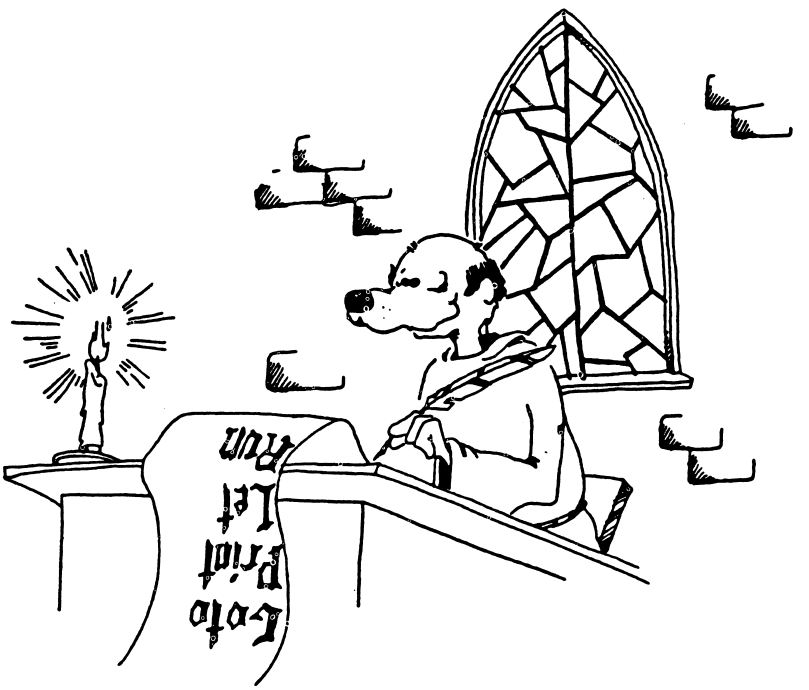
LISTING THE PROGRAM

To show the whole stored program, enter:

```
list
```

To show line 30 of the program, enter:

```
list 30
```



CAPITAL LETTERS

The computer lists commands in capital letters, even when you typed in lower case letters. From now on in this book, we will use capital letters for commands (like LIST, PRINT, REM, RUN, NEW), but you can type in lower case letters if you want.

THE MEMORY

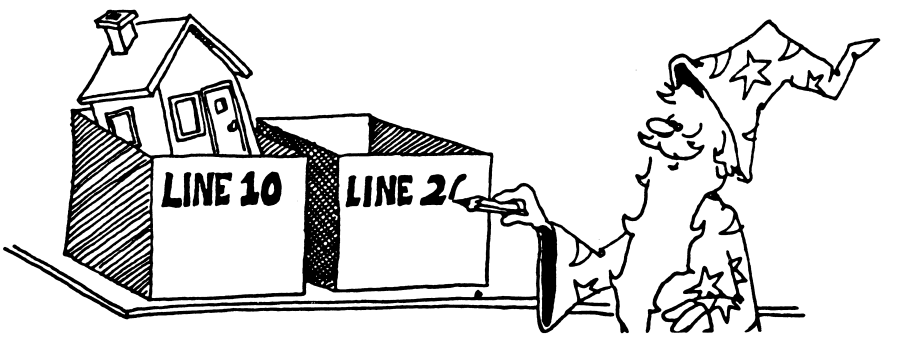
The computer's memory is like a shelf of boxes. On the front of each box goes its name. At the start, all the boxes are empty and no box has a name.

When you entered: 10 REM house

the computer took the first empty box and wrote the name "Line 10" on the label. Then it put the statement

REM house

into the box and put the box back on the shelf.



When you entered: 20 PRINT "listen"

the computer took the second box and wrote "Line 20" on its label. Then it put the statement

PRINT "listen"

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.

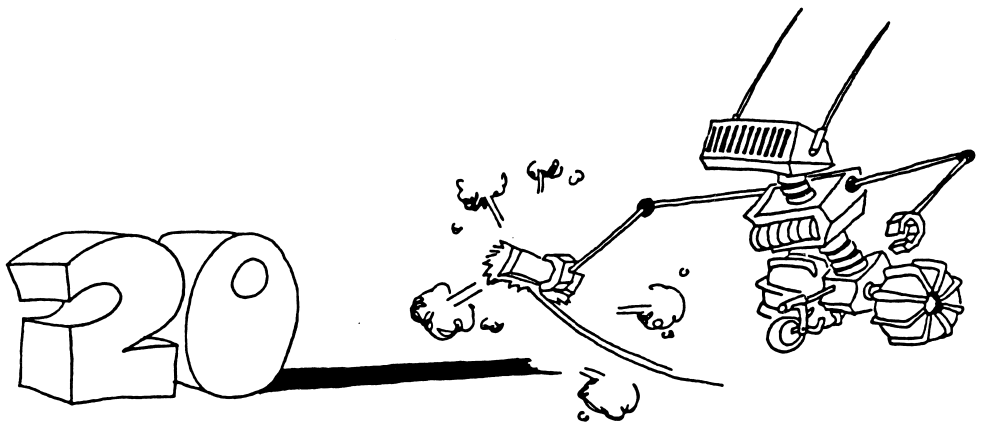
When you enter only a line number with nothing after it, the computer finds the box with that line number on the front. It empties the box and erases the line number off the front of the box.

How do you erase the whole program? (Look at the beginning of this lesson 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.

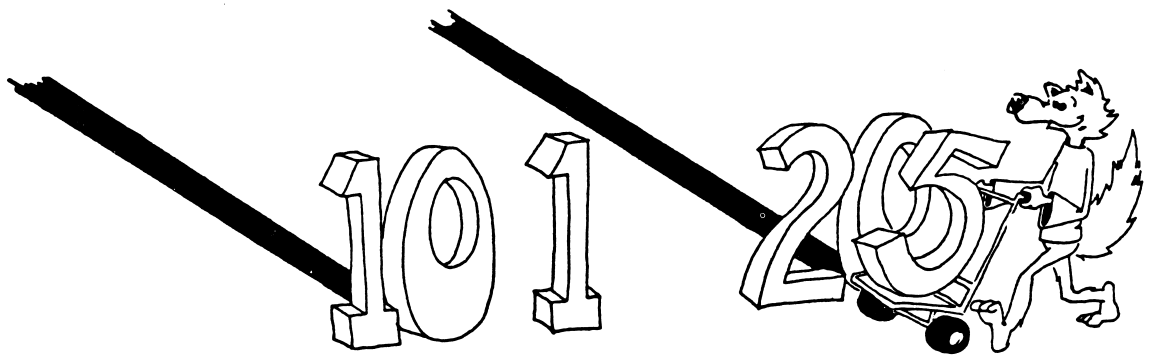
Enter NEW and this:

```
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.

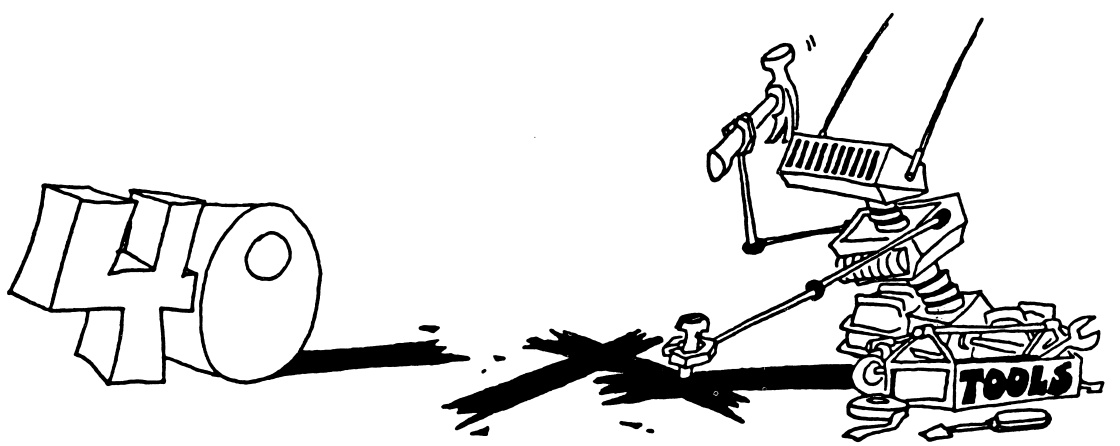


FIXING A LINE

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 a title on your program.

Enter NEW and this:

```
10 REM Lazy
20 cls
30 PRINT "line 10 does nothing"
35 REM this line does nothing
RUN
```

What does line 30 do? _____

What does line 35 do? _____

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 CLS
20 PRINT
30 PRINT " XXXXXX"
40 PRINT "XXXXXXXXXXXX"
50 PRINT "  O          O"
```

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

Assignment 3:

1. What command will list line 10 of the program?

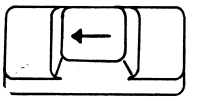
2. How do you tell the computer to list the whole program on the screen?

3. What does the computer do (if anything) when it sees the REM command? What is the REM command used for?

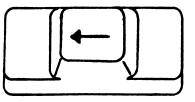
4. Add a COLOR command to the car picture program so that a blue car on a red background is drawn.

5. Use CLS, BEEP, REM and PRINT to draw three flying birds on the screen. Make each bird peep after it is drawn.

INSTRUCTOR NOTES 4 THE KEYBOARD AND EDITING

This lesson concerns the arrow keys, the rubout key  and the delete key.

The arrow keys are used in moving the cursor to any spot on the screen. Characters on the screen are not affected by the cursor moving over them. Wherever the cursor stops, you can type in new characters. For now, characters cannot be inserted, only replaced by others or deleted.

There are two deletion modes: Rubout  and Delete.

The difference is explained in the text.

A repaired program line, when all is satisfactory, can be entered into the computer by pressing Enter, as usual.

You can even change the line number, and then you have two identical lines with different numbers. Sometimes you need several similar lines. It is much easier to create them by modifying a single line, using the cursor keys, than to type each from scratch.

The arrow keys can be used to fix any line that you see on the screen. If the line you want to fix is not on the screen, put it there with LIST.

Holding any key down a short time starts the "auto repeat" feature of the keyboard. This is very useful for making repeated characters, such as a line of characters or spaces in a line, or for moving the cursor fast with the arrow keys.

QUESTIONS:

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

2. Have your student demonstrate the following:

   How to edit a line. This includes using the arrow keys to move the cursor to the interior of the line, modifying characters there, and pressing Enter.

3. Demonstrate use of the Rubout key.

4. Demonstrate use of the Delete key. Explain how the Rubout key and the Delete key differ.

5. Using the repeat feature of the keyboard.

LESSON 4 THE KEYBOARD AND EDITING

THE CURSOR IS A FLASHING LINE

The little flashing line is called the "input cursor." It shows you where the next letter you type will appear on the screen.

THE ARROW KEYS MOVE THE CURSOR

Find the four arrow keys on the right side of the computer. They are on number keys 2, 4, 6, and 8.

These keys move the cursor. Press one. Hold it down. Use the arrow keys to move the cursor all over the screen.

(If the cursor doesn't move, but numbers are printed, then fix it by pressing the "Num Lock" key just once.)

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

REPEATING KEYS

Hold down a cursor arrow key. The cursor goes whizzing along!

This works for most keys on the keyboard. Try holding down the "H" key. You see:

hhhhhhhhhhhhhhhhhhhhhhhhh

Do this: Clear the screen.
Move the cursor to the middle of the screen.
Type your name there.
Now put a box of "stars" around your name.
Hold down the shift and the "*" key to make the top and bottom of the box.

FIXING MESSED UP LINES

The cursor arrow keys help you fix errors in typing.

Enter: 10 rem zragon

We want dragon

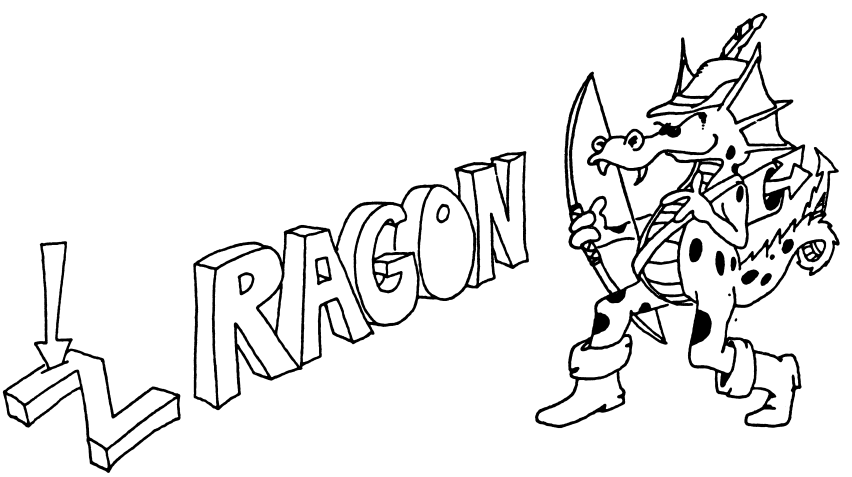
Use the arrow keys to move the cursor onto the z.

Type a d instead.

Now the line is correct, reading:

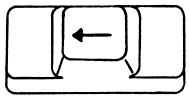
10 rem dragon

Press Enter to store the correct line in the memory.



ERASING LETTERS

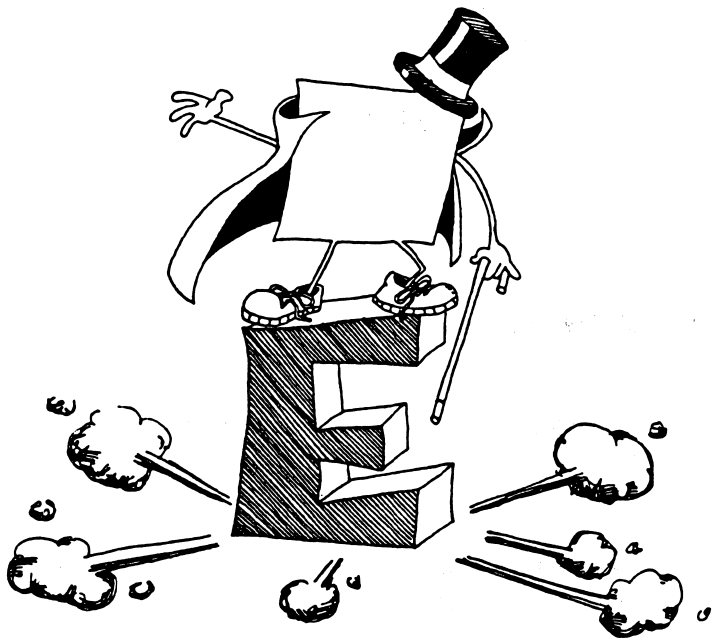
There are two erase keys. One is next to the Num Lock key and has an arrow on it like this:



We call it the "Rubout" key. It is like rubbing out a mistake with an eraser.

The other is the Del key. Del stands for "delete" which means "take away."

Enter: `10 rem aaaaaaaaaaEiiiiiiiiiii`



Move the cursor over the E. Press the Del key. It erases the letter E that the cursor was on.

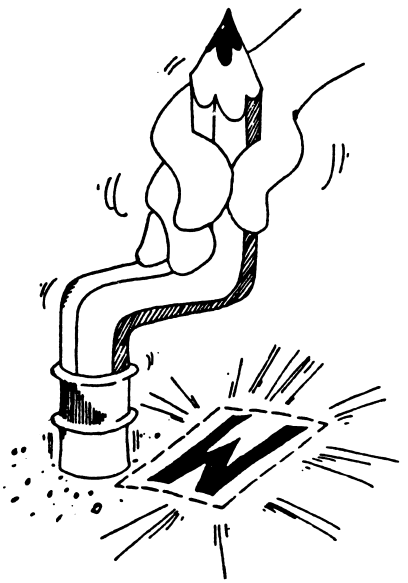
Now hold the Del key down. The cursor sits there eating up all the i letters!

Again: `10 rem aaaaaaaaaaWiiiiiiiiiii`

Move the cursor over the W. Press the Rubout key. It erases the a next to it!

Now hold the Rubout key down. The cursor goes whizzing off to the left, erasing all the a letters.

NOTICE: The Rubout key does not erase the W. It always erases the letters to its left.



WHAT IS THE DIFFERENCE?

Del	erases what is under it and then sits there eating up letters from the right.
Rubout	erases what is next to it on the left, and then goes whizzing along to the left, erasing as it goes.

THE CLS COMMAND IN A PROGRAM LINE

RUN:

```
10 REM vanishing junk
12 cls
20 print
22 print " Nice clean screen!"
```

The CLS command clears the screen and puts the cursor into the home spot on the screen.

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 the Enter key to enter the line into memory.

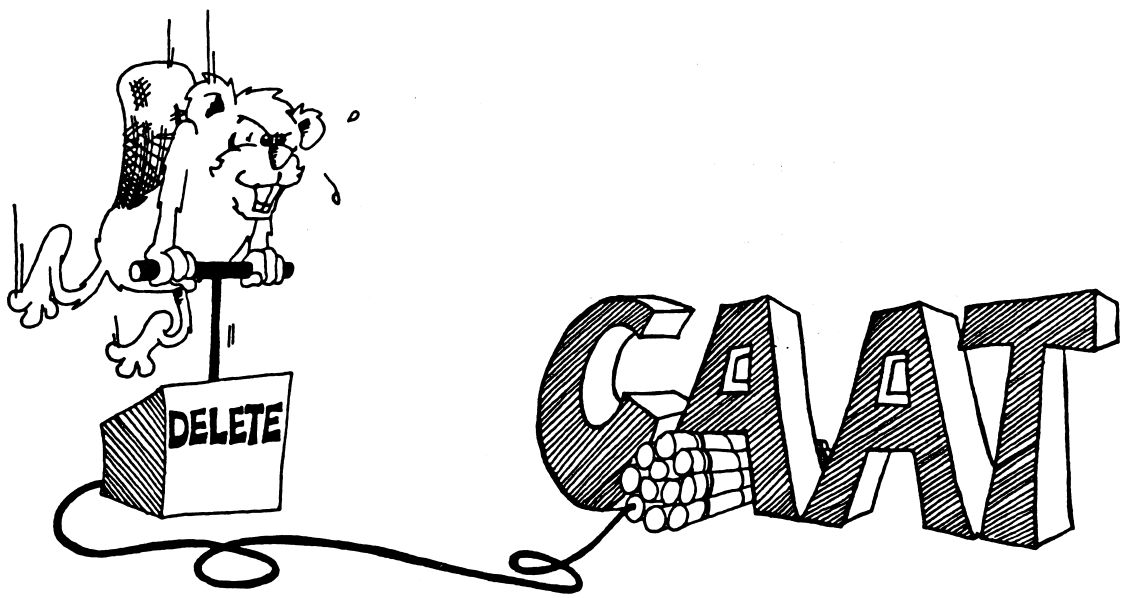
2. Fix this line to read CAT:

```
10 REM CAAAT
```

   First fix it using the Del key. Then write it again and fix it using the Rubout key.

3. Draw a large "smiley face."

4. Write a program that makes a valentine. Use COLOR 4,7 to make red letters on a white background.



INSTRUCTOR NOTES 5 THE INPUT STATEMENT

This lesson is about the INPUT command and string variables.

We introduce the input command in its simplest form:

```
INPUT A$
```

without a message in quotes in front. This allows the student to concentrate on the central feature of an INPUT.

Similarily, we will give only the essential feature of each command for the whole of the introduction of the book (through Lesson 14). We want the student to "see the forest" before going into details. The commands required for interesting programs are:

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

String variables are introduced using the "box" concept again. For the time being, variable names are restricted to one letter. This allows faster typing and puts off discussion of the complicated naming rules until after our sprint to the RND command.

The "two hats" of the student, programmer and user of the programs, cause much confusion as the student reads the assignments. PRINT is the programmer speaking, while the user can only speak when invited by an INPUT command put there by the programmer.

QUESTIONS:

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

2. How does the program ask the user to type in something?

3. How do you know the computer is waiting for an answer?

4. What is a letter with a dollar sign after it called?

5. Write a short program that uses CLS, PRINT and INPUT.

6. Are you in trouble if the computer answers with Redo from start after an input? What made it do that? What do you do next?

LESSON 5 THE INPUT STATEMENT

Use INPUT to make the computer ask for something.

Enter:

```
10 REM Talky-Talk
15 cls
20 PRINT "say something"
25 INPUT A$
30 PRINT
35 PRINT "did you say "
40 PRINT A$;
50 PRINT "?"
```

RUN it. When you see a question mark, type "hi" and press the ENTER key.

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

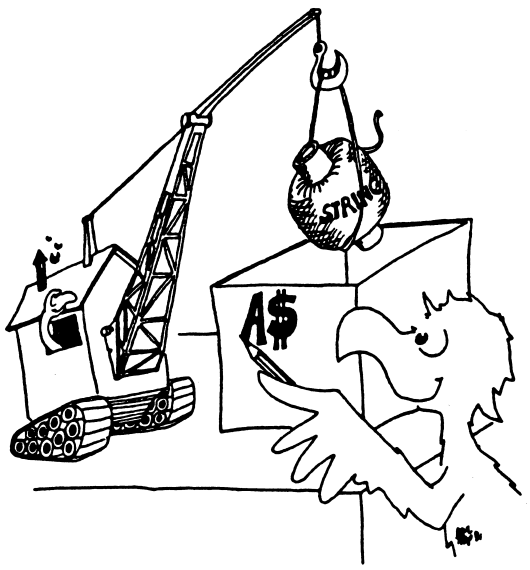
When you type hi, the computer stores this word in a box named A$.

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

RUN the program again and this time say something funny.

STRING VARIABLES

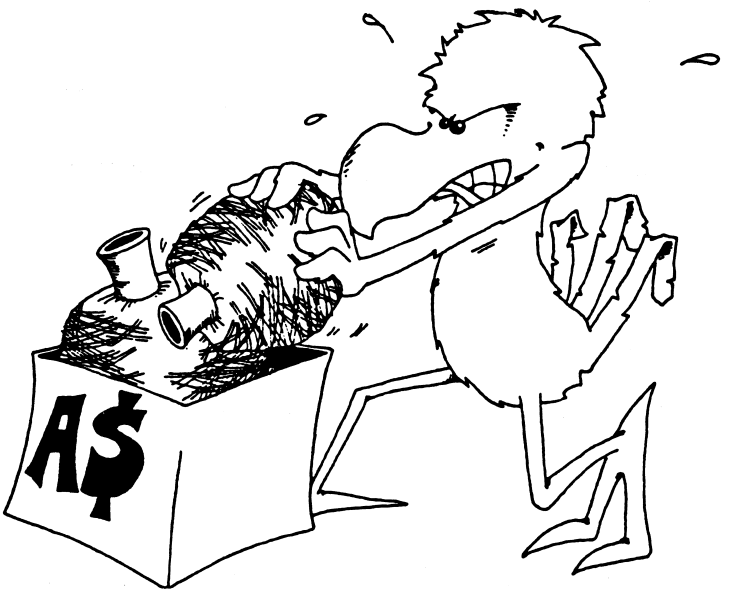
A$ is the name of a "string variable." The computer stores string variables in memory boxes, just like the boxes it puts program lines into. The name is written on the front of the box and the string is put inside the box.



RULE: A string variable name ends in a dollar sign, "$". You can use any letter you like for the name and then put a dollar sign after it.

A$ is called a variable because you can put different strings into the box at different times in the program.

The box can hold only one string at a time.



Putting a new string into a box automatically erases the old string that was in the box.

ERROR MESSAGES FROM INPUT

Run this two times:

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

Try these answers:

```
HI
HI, THERE
```

RULE: Do not put any commas into the string you type in answer to the computer.

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

```
?redo from start
```

and wait. This means that the computer wants you to try again, but do not put any commas into the answer you type.

YOU WEAR TWO HATS, USER AND PROGRAMMER

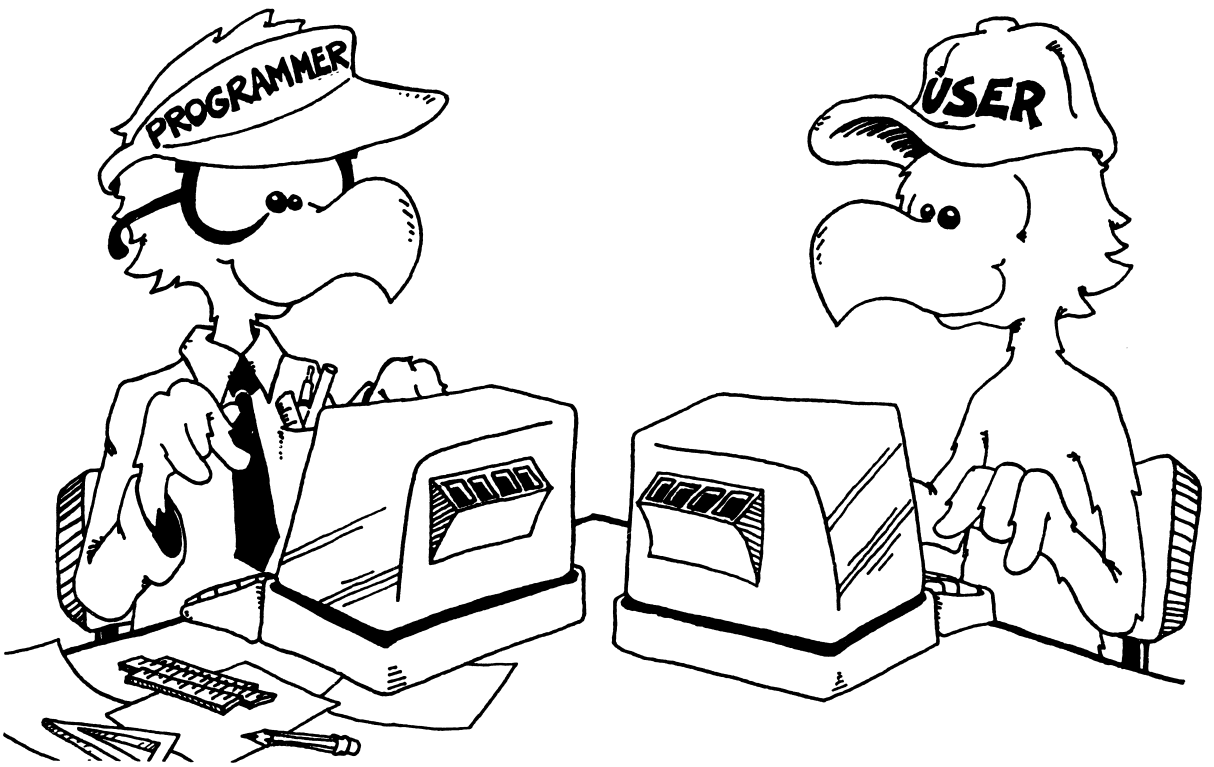
You are a PROGRAMMER when you write a program. The person who runs the program is a USER.

Of course, if you run your OWN program then YOU are the USER.

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

When the programmer writes an INPUT command, she is asking the USER to say something to the computer.

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



Assignment 5:

1. Write a program that asks for a person's name and then says something silly to the person, by name.

2. Write a program that asks you to INPUT your favorite color and put it into a box called C$. Now the program asks you your favorite animal and puts this into box C$ too. Have the program print C$. What will be printed? RUN the program and see if you are right.

INSTRUCTOR NOTES 6 TRICKS WITH PRINT

In this lesson:

PRINT with a semicolon at the end
PRINT with semicolons between items
the "invisible" PRINT cursor

We don't discuss the use of commas in a PRINT statement.

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 flashing line. 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 and the next PRINT will put its first character exactly in 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 numeric constants, variables, and expressions. Numeric constants and variables have not yet been introduced. The items are separated by semicolons.

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

QUESTIONS:

1. Which cursor is a little flashing line? 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. Show how to use the Ins key to put the "o" into "dg" to make "dog."

LESSON 6 TRICKS WITH PRINT

ONE LINE OR MANY?

Enter this program:

```
10 REM food
20 PRINT
30 PRINT "toast"
40 PRINT "and"
50 PRINT "jam"
```

and RUN it.

Each PRINT command prints a separate line.

Now enter:

```
30 PRINT  " toast ";
40 PRINT  " and ";
```

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

RUN it.

What was different from the first time?

_____

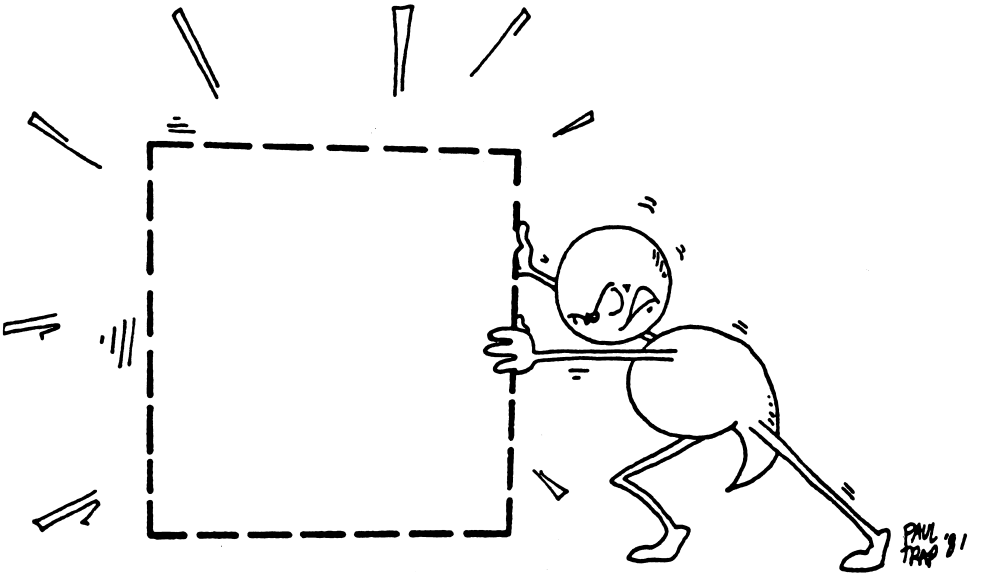
_____

THE HIDDEN CURSOR

Remember the flashing line? It is the INPUT cursor and shows where the next letter will appear on the screen 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 REM famous
20 PRINT "enter a name"
30 INPUT A$
35 cls
40 PRINT "enter another"
50 INPUT B$
60 cls
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"
```

Don't forget the spaces after "rock" and "and."

THE INSERT KEY

In the last lesson you learned how to erase letters. There were two ways to erase: use the Rubout key or use the Del key.

The opposite of "erase" is to "insert." Insert means to "stick in another letter."

Try this: 10 REM draon

We left the g out of dragon!

To fix it, put the cursor over the o. Press the Ins key. Then press the g key. Now the line is fixed to read:

10 REM dragon

RULE: Put the cursor on the letter to the right of the spot you want the inserted letter to go.

INSERT A LOT!

Try this. Fix: 10 REM dn to be dragon

RULE: You have to press Ins only once, then type as many letters in as you want.

To stop the Ins thing, just move the cursor with any of the arrow keys. Try this:

Fix: 10 REM dn to be dragon smoke

How? First put the cursor on n, press Ins, add rago. Then use the right arrow key to move the cursor to the space after dragon and type smoke.

Assignment 6

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 three print commands to print on one line.

3. Type these lines and then fix them using the INS key.

```
10 REM wizrd      (make it wizard)
10 REM cmptr      (make it computer)
10 REM kybd       (make it keyboard)
```

INSTRUCTOR NOTES 7 THE LET STATEMENT

The LET statement is introduced using the concept of memory boxes. Concatenation using the "+" symbol is called "gluing the strings."

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

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

Used so far:

NEW, PRINT, REM, RUN, BEEP, CLS, COLOR, LIST, INPUT, LET

Special keys discussed so far:

Enter, Shift, Ctrl and four cursor keys, Rubout, Del, Ins, Num Lock

QUESTIONS:

1. LET puts things in boxes. So does INPUT. How are they different?

2. In this program:

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

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

3. If you RUN this little program:

```
10 LET H$="fat"
20 LET K$=" sausage"
30 LET P$=A$ + K$
```

   what is in each box after the program RUNs?

LESSON 7 THE LET STATEMENT

The LET statement put things into boxes. Enter and RUN:

```
10 CLS
20 LET Q$="truck"
40 PRINT Q$
```

Here is what the computer does:

Line 10 — The computer clears the screen.

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

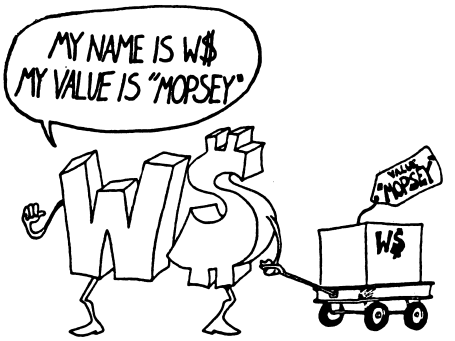
Line 40 — The computer sees that it must print whatever is in box "Q$". It goes to the box and makes a copy of the string "truck" that it finds there. It puts the copy on the monitor screen. The string "truck" is still in box "Q$".

NAMES AND VALUES

This line makes a string variable:

```
30 LET W$="MOPSEY"
```

The name of the variable is W$.
The value of the variable is put into the box.
In this line, the value of W$ is MOPSEY



ANOTHER EXAMPLE:

Enter and RUN:

```
10 LET D$="pickles"
20 LET A$=" and "
30 PRINT "what goes with pickles?"
35 INPUT Z$
40 CLS
50 PRINT D$;A$;Z$
```

Explain what the computer does in each line.

10 _____

20 _____

30 _____

35 _____

40 _____

50 _____

GLUING THE STRINGS

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

```
10 CLS
20 LET W$="har de "
25 LET X$="har "
30 LET L$=W$ + X$
40 PRINT L$
50 PRINT
60 LET L$=L$ + X$
70 PRINT L$
```

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

40 _____

70 _____

Now RUN the program to see if you were right.

RULE: The "+" sign sticks two strings together.

Assignment 7:

1. Write your own program which uses the LET command and explain how it stores things in "boxes."

2. Write a program that inputs two strings, glues them together and then prints them.

INSTRUCTOR NOTES 8 THE GOTO COMMAND AND THE BREAK KEY

The GOTO command allows a "dumb" loop that goes on forever. It also helps in the 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 Ctrl-Break keys do this nicely.

GOTO is tolerant of "spaghetti" programming. Examples of "spaghetti" are shown to the students, and although some fun is had with them, the idea is to make the student aware of the mess that undisciplined use of GOTO can make.

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 "Incredible ";
20 PRINT "Blue Machine "
30 GOTO 20
```

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

4. Write a short program that "beeps," 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 CLS
20 PRINT "your name?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

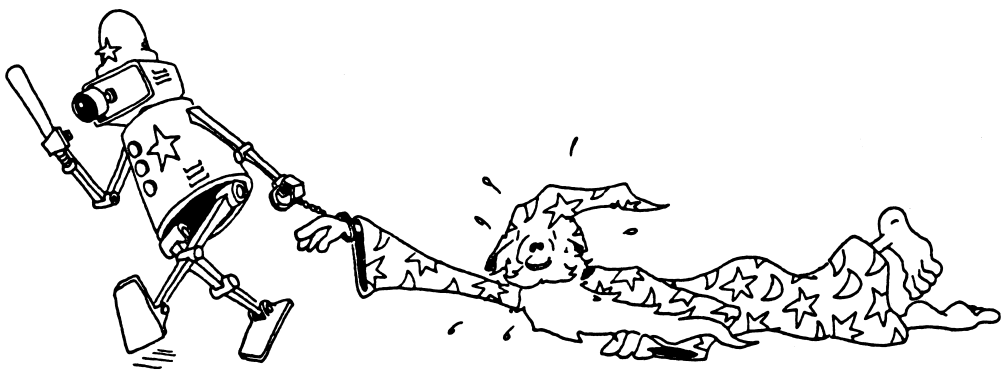
RUN this program. It never stops by itself! To stop your name from whizzing past your eyes:

hold down the Ctrl key and press the Break key.

Break is written on the front of the Scroll Lock 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.



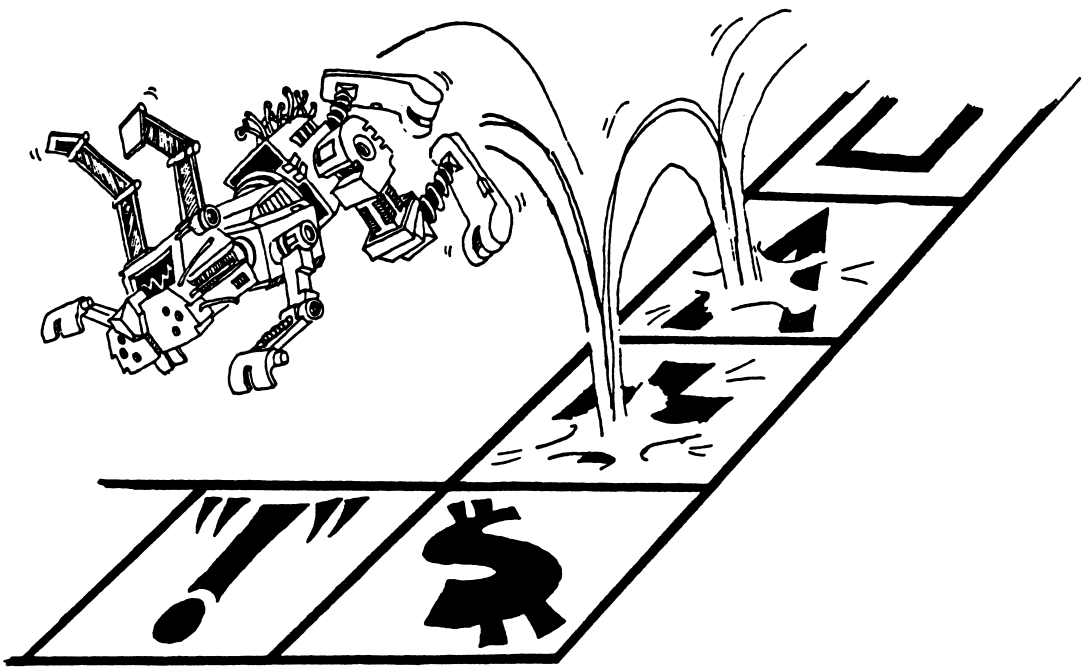
MORE JUMPING

Enter:

```
10 REM shut up!
20 PRINT "say something"
30 INPUT S$
35 PRINT
40 PRINT "did you say ' ";S$;" '?"
45 PRINT
50 GOTO 30
```

RUN the program. Type an answer every time you see the ? and the flashing cursor. Press the Ctrl Break keys to end the program.

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



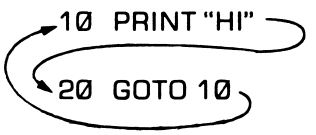
KINDS OF JUMPS

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 Ctrl Break to stop it.

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

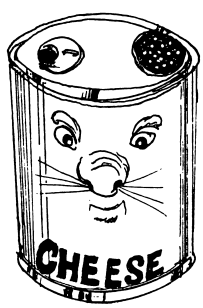
THE Ctrl Break KEY

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

A CAN OF SPAGHETTI

Look at this:

```
10 REM --- Spaghetti ---
20 GOTO 70
25 PRINT "a"
26 GOTO 50
30 PRINT "S"
31 GOTO 25
40 PRINT "c"
41 GOTO 90
50 PRINT "u"
51 GOTO 40
70 PRINT "Spaghetti"
71 GOTO 30
90 PRINT "e"
99 REM --- end ---
WHEW!
```



This is not a good, clear program!

It is a "spaghetti" program.

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

Assignment 8:

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

```
10 REM > > > Forked Tongue > > >
20 GOTO 40
30 PRINT "n"
31 GOTO 60
40 PRINT "s"
41 GOTO 30
50 PRINT "E"
51 GOTO 99
60 PRINT "a"
61 GOTO 90
90 PRINT "k"
91 GOTO 50
99 PRINT "B i t e"
```

2. Rewrite the snake program above, leaving out the GOTOs and so making the program "clean and lean."

3. Write a program that prints "Teen Power" over and over.

4. How do you stop your program?

5. Write another program that prints your name on one line, then a friend's on the next, over and over. Sound a "beep" as each name is PRINTed. Stop the program with the Ctrl Break keys.

6. Write a program that uses each of these commands: CLS, BEEP, PRINT, INPUT, LET, GOTO. It also should glue two strings together.

INSTRUCTOR NOTES 9 THE IF . . . THEN STATEMENT

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.

"Phrase A" is used for the assertion being tested for truth. "Command C" is used for the command to be executed if the assertion is true.

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

```
A$ = B$
C$ <> D$
```

On 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:

```
<    >    =    =<    =>    <>
```

will be treated in later lessons.

QUESTIONS:

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

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

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

3. What is meant by "phrase A"? By "command C"? Where is the "fork in the road" in an IF statement?

LESSON 9 THE IF STATEMENT

Clear the memory and enter:

```
10 CLS
20 PRINT "Are you happy? (yes OR no)"
30 INPUT A$
40 IF A$="yes" THEN PRINT "I'm glad"
50 IF A$="no"  THEN PRINT "Too bad"
```

RUN the program several times. Try answering yes, no or maybe. What happens?

YES _____

NO _____

MAYBE _____

THE IF STATEMENT

The IF statement has two parts:

```
10 IF phrase A THEN command C
```

First, the computer looks at "phrase A".

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

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

It looks like this:

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

or

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

THE "IF" IN ENGLISH AND BASIC

In English:

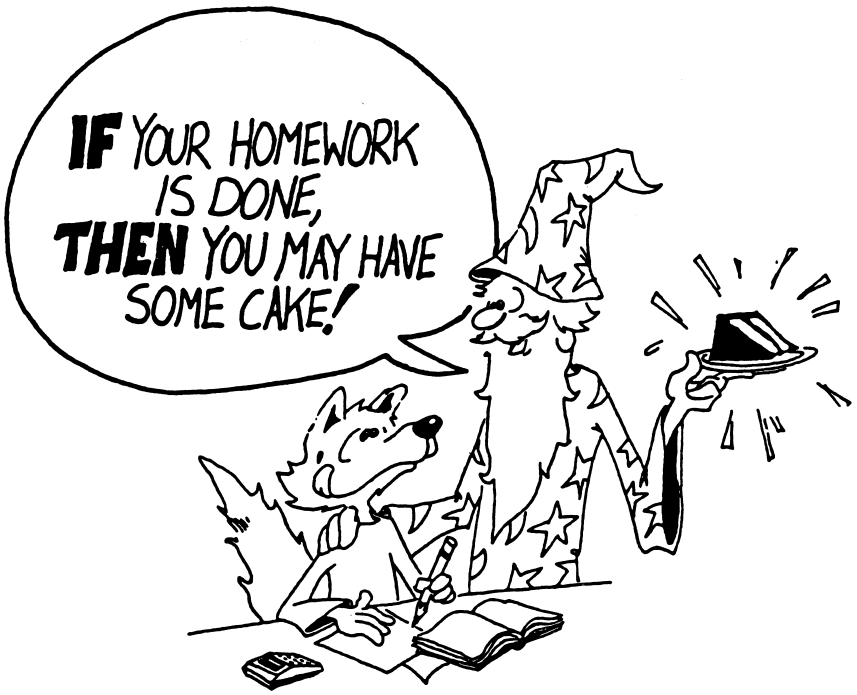
IF your home work is done, THEN you may have some cake.

In BASIC:

```
40 IF A$="done"  THEN PRINT "eat cake"
```

Assignment 9A:

1. 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."



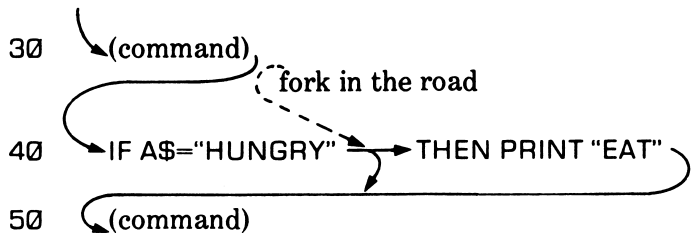
A FORK IN THE ROAD

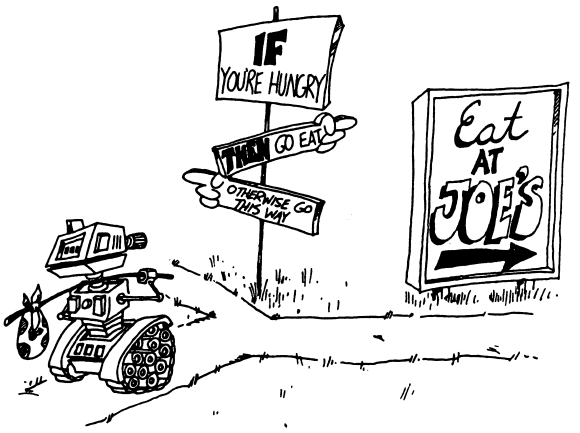
When it sees IF, the computer must choose which road to take.

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

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





THE "NOT EQUAL" SIGN

Two signs: = means "equal"
< > means "not equal"

To make the "< >" sign:

hold down the shift key
then press the "<" key, then the ">" key.

USING THE < > SIGN

40 IF phrase A THEN command C

Phrase A is a phrase that is TRUE or FALSE.

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

Put it in an IF command:

```
40 IF B$< >"FIRE"      THEN PRINT      "Feed him hot chili."
```

If the B$ box contains COLD
then B$ is not equal to FIRE
so the expression B$< >"FIRE" is TRUE.
The computer prints Feed him hot chili.

Or

If the B$ box contains FIRE
then the phrase B$< >"FIRE" is FALSE
so computer will not print anything.

Here is how it looks in a program:

```
10 PRINT "With dogs it's a cold nose."
11 PRINT
12 PRINT "With dragons, it's . . . "
13 PRINT
15 PRINT "How is your dragon's breath?"
16 PRINT
20 PRINT "(Enter 'fire' OR 'cold')"
30 INPUT B$
40 IF B$< >"FIRE" THEN PRINT "Feed him some hot chili."
50 IF B$= "FIRE" THEN PRINT "Watch out!"
60 PRINT "Nice dragon"
```

Assignment 9B:

1. Write a "pizza" program. Ask what topping is wanted. Make the computer answer something silly for each different choice. You can choose mushrooms, pepperoni, anchovies, green peppers, 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 INPUTting guesses into string G$. Use two IF lines, one with a "phrase 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

Numeric 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" which 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 numeric 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. What are 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 / symbol.

4. Give an example of a program line that would have a Type Mismatch error due to mixing a string variable with a numeric variable.

5. Explain what is meant by the "name of a variable" and the "value of a variable" for numeric variables. For string variables.

LESSON 10 INTRODUCING NUMBERS

INPUT, LET AND PRINT

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

```
10 REM bigger
15 CLS
20 PRINT "Give me a number."
30 INPUT N
40 LET A=N+1
45 PRINT
50 PRINT "Here is a bigger one."
60 PRINT A
```

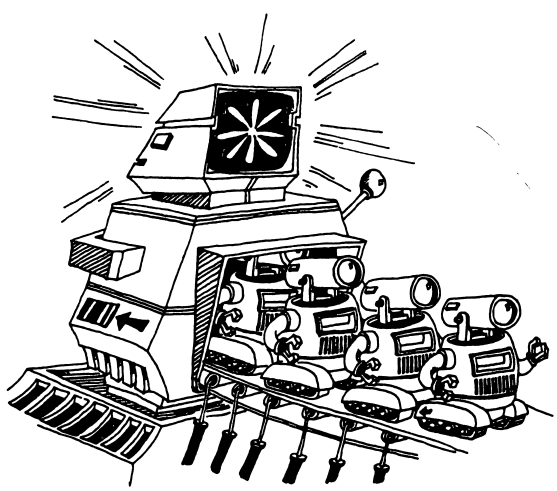
ARITHMETIC

+	addition	*	multiplication
–	subtraction	/	division

The + and – signs are on the far right side of the keyboard.

Computers use "*" instead of "x" for a multiplication sign.
Try this. Change line 40 so that N is multiplied by 5.

Computers use "/" for a division sign. Answers are in decimals.
Try this. Change line 40 so that N is divided by five. What do you say in line 50?

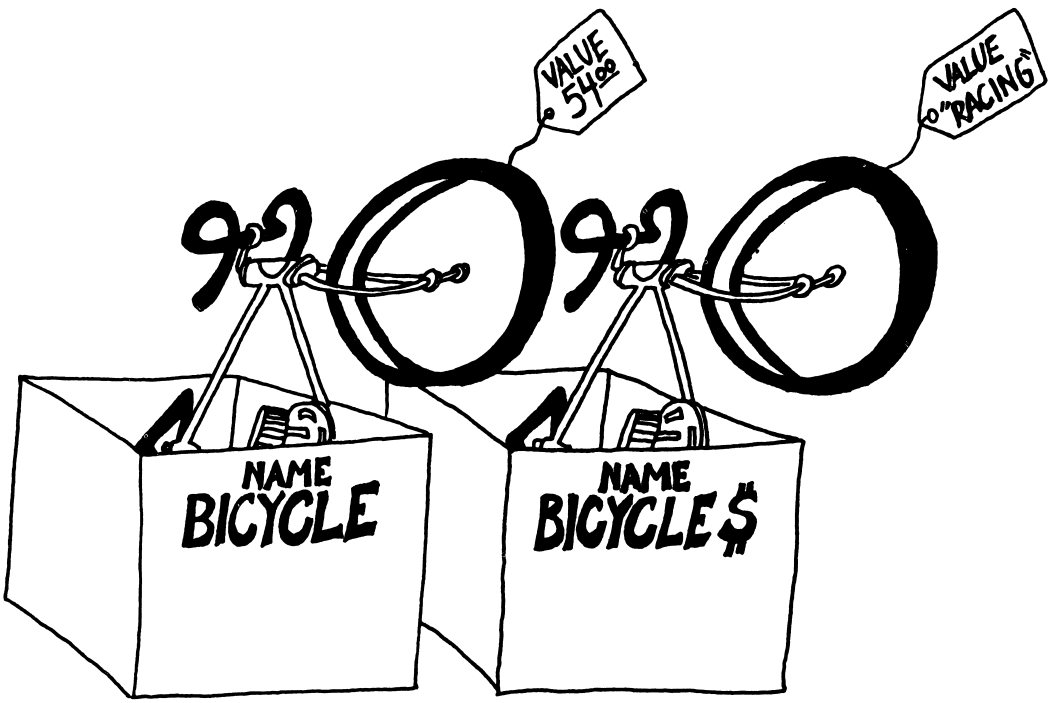


VARIABLES

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=2001
20 LET B=1983
30 LET C=B-A
40 PRINT "How much longer, Hal?"
50 PRINT C$;" years."
```

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 numeric constants: 5, 22, 3.14, -50

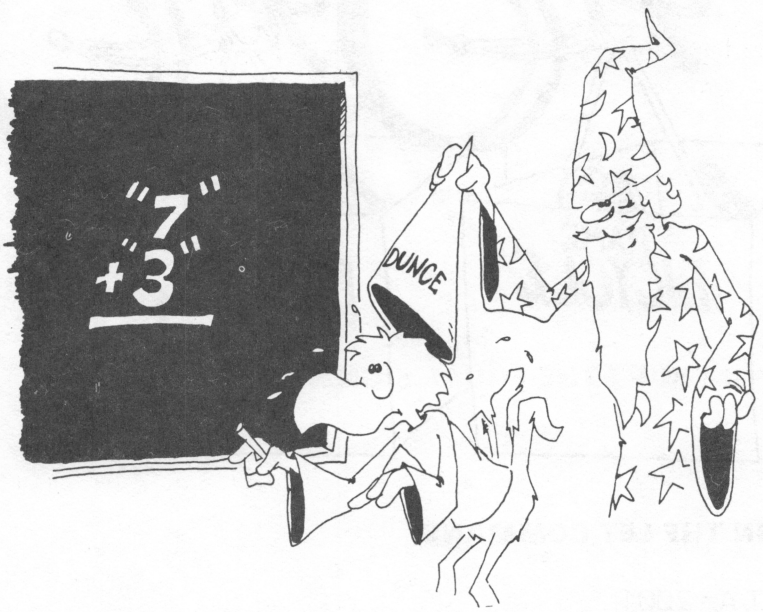
Some string constants: "HI", "7", "TWO", "3.14"

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"
```

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

```
Type mismatch error in 10
```



There are two types of variables: string and numeric.
You cannot put a string in a number box or a number in a string box.

Enter:

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

Lines 10 and 20 are OK, line 30 is wrong. What will the computer do when you run 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"
PRINT A;B$
```

Run this line.
```
10 PRINT 5/2;"is equal to 5/2"
```

A FUNNY THING ABOUT THE EQUAL SIGN

The "=" sign in computing does not exactly mean "equals." 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.

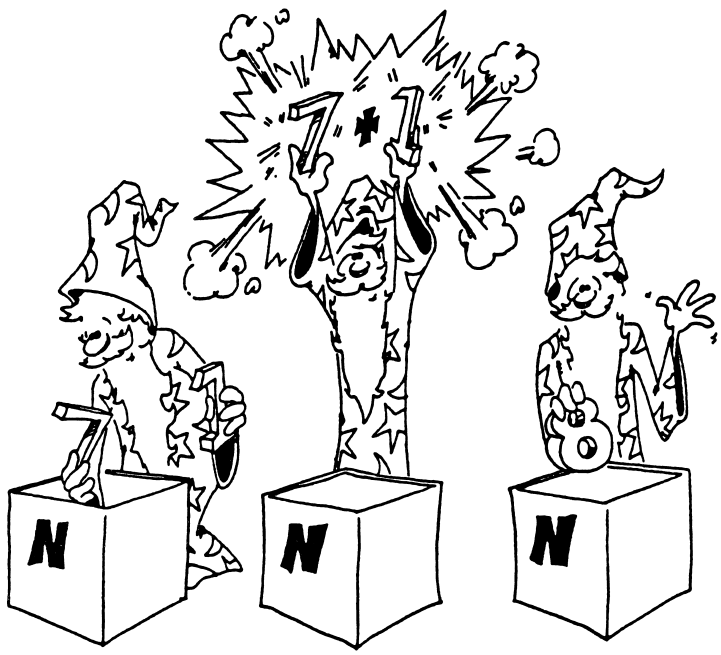
But it is OK in computing to say N=N+1 because the "=" sign really means "replace." Here is what happens:

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 in the box.

Another way to say the same thing is:

	10 LET	N	=	N	+	1
means	LET	(new N)	equal	(old N)	plus	one



DON'T BE BACKWARD!

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

Arithmetic:	N = 3 3 = N	is the same as
BASIC	LET N = 3 LET 3 = N	correct wrong
BASIC	LET N = B LET B = N	is not the same as Why not? (what is in each box after the line runs?)

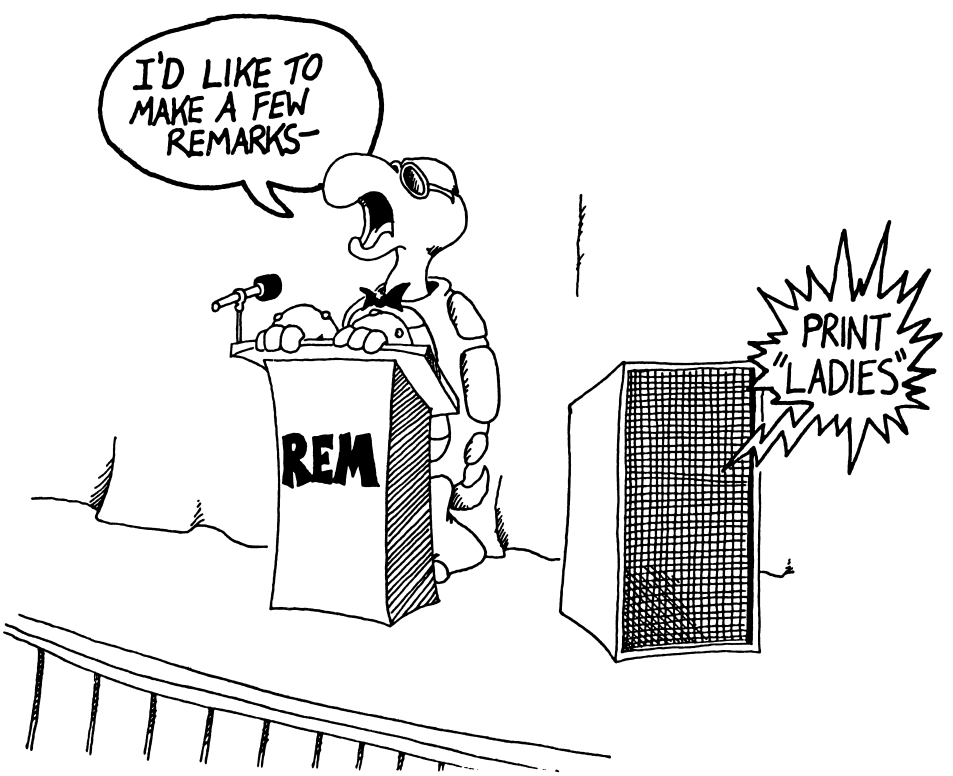
LET N = B means _____

LET B = N means _____

Assignment 10:

1. Write a program that asks for your age and the current year. Then subtract and print out the year of your birth. Be sure to use PRINT statements to tell what is wanted and what the final number means.

2. Write a program that asks for two numbers and then prints out their product (multiplies them). Be sure to use lots of PRINTs to tell the user what is happening.



INSTRUCTOR NOTES 11 TAB AND DELAY LOOPS

The TAB command mimics the familiar tab function of a typewriter. The lesson treats "delay loops" which are useful in themselves, but are really just empty FOR . . . NEXT loops.

TAB must be used in a PRINT command. Several TAB commands can be used in one PRINT statement, but the arguments in the parentheses must increase each time. That is, TAB cannot be used to move the cursor back to the left. Later we address the LOCATE command which allows placement of the cursor anywhere on the screen.

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

This lesson introduces delay loops used to slow the program down so that its operation can be more easily observed. They are also called timing loops. The loops are given as a unit, without explanation of how they work.

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

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

QUESTIONS:

1. Show how to write a delay loop that lasts for about two seconds.

2. Will this work for a delay loop?

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

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

```
10 PRINT "Hi";TAB(10);"good looking!"
10 TAB(5);PRINT "OH-OH!"
10 PRINT TAB(15);"nope";TAB(1);"not here"
```

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

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

LESSON 11 TAB AND DELAY LOOPS

THE TAB COMMAND

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

(The printing cursor is invisible.)

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

Try this:

```
10 PRINT "123456789abcdef"
20 PRINT TAB(3);"y";TAB(7);"z"
```

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

CAREFUL! Try this: 10 TAB(5)

You see Syntax error. You have to use TAB( ) in a PRINT statement. You cannot use TAB( ) by itself.

YOU CANNOT TAB BACKWARDS

Try this:

```
10 PRINT "123456789ABCDEF"
20 PRINT "a";TAB(9);"b";TAB(3);"c"
```

The TAB( ) command can only move the printing to the right. You cannot move back to the left. If you try to go back, the computer prints on the next line instead.

YOUR NAME IS FALLING!

```
10 CLS
15 LET N=1
20 PRINT "Your first name."
30 INPUT W$
40 PRINT TAB(N);W$
50 LET N=N+1
60 GO TO 40
```

Press BREAK to stop the run.

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

```
15 LET N=30
50 LET N=N-1
```

FAT NUMBERS

Numbers have a space glued on each side before they are printed on the screen.

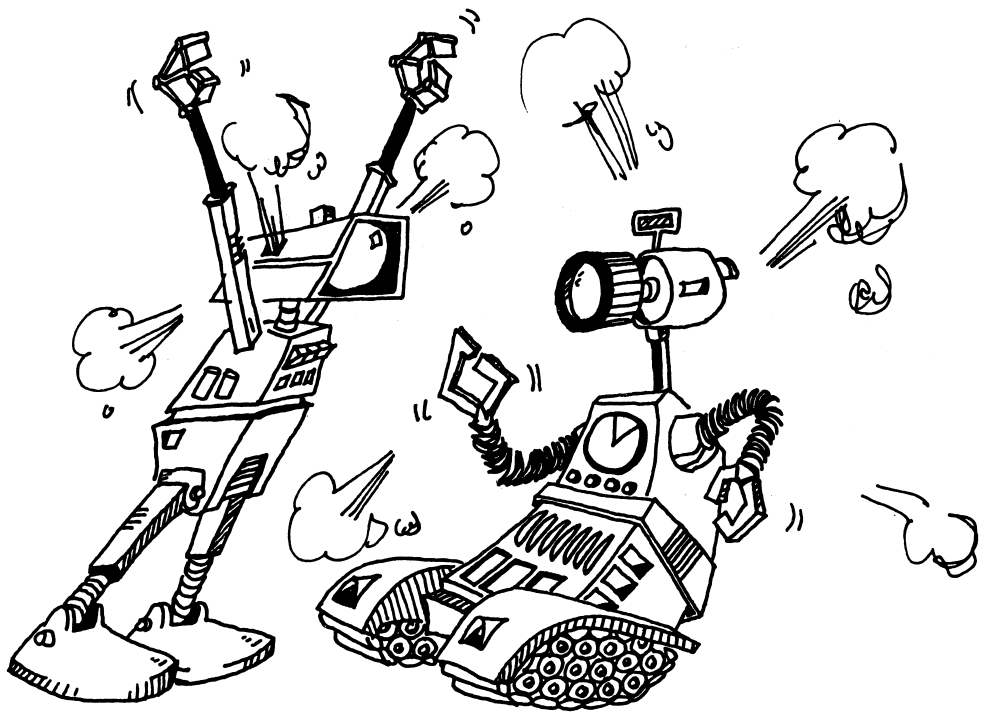
Try this:

```
10 PRINT "123456789"
20 PRINT 1;2,3;-1;-2;-3
```

(If the number is negative, a "-" sign instead of a space is put on the left.)

FUNCTIONS DON'T FIGHT BUT THEY HAVE ARGUMENTS

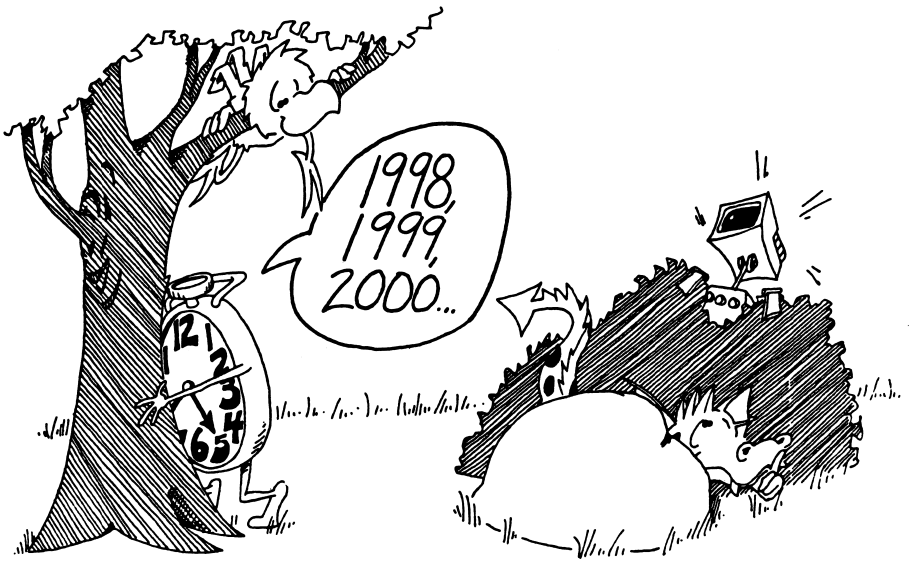
TAB( ) is a function. We will study other functions like RND( ), INT( ), LEFT$( ), etc. The number inside the ( ) is called "the argument of the function." TAB( ) says "move the cursor over" and the argument tells "where to move it to."



Assignment 11A:

1. Write a program that asks for last names and nicknames. Then print the last name starting at column 5 and the nickname at column 15. Use a GOTO so the program is ready for another last name - nickname pair.

2. Write an "insult" program. It asks your name, then it beeps and writes your name. Then it TABs over in the line and prints an insult.



DELAY LOOPS

Here is a way to slow down parts of a program. It is a "delay loop."

RUN this program:

```
10 REM Game
20 cls
30 PRINT "hide"
40 FOR I=1 TO 2000: NEXT I
50 PRINT "coming ready or not"
```

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

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

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

```
10 REM ---- tick tock ----
20 cls
30 PRINT "wait how long? "
34 INPUT S
36 T=S*1000
40 FOR I=1 TO T:NEXT I
45 PRINT
46 BEEP
50 PRINT S;"seconds are up"
```

Assignment 11B:

1. Write a "slow poke" program that prints out a three word message with several seconds between each word. Have the computer peep before each word.

INSTRUCTOR NOTES 12 THE IF . . . THEN STATEMENT WITH NUMBERS

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

$$=\qquad >\qquad <\qquad <>$$

It is a good idea to get the student to pronounce these expressions out loud. "A < B" makes a lot more sense when pronounced "A is less than B," than when just allowed to flow in the eyeballs. Of course, the "point" of the < and the > symbols (that is, the little end) is at the side of the smaller of the two numbers.

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

A "home made" loop is demonstrated in the GUESSING GAME, but is not discussed. The loop starts in line 50 and goes to 80. The exit test is made in line 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 STATEMENT WITH NUMBERS

Try this:

```
10 REM *** Teenager ***
15 cls
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 phrase A is true THEN do command C

"Phrase A" can have these arithmetic symbols:

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

Each "phrase A" 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"

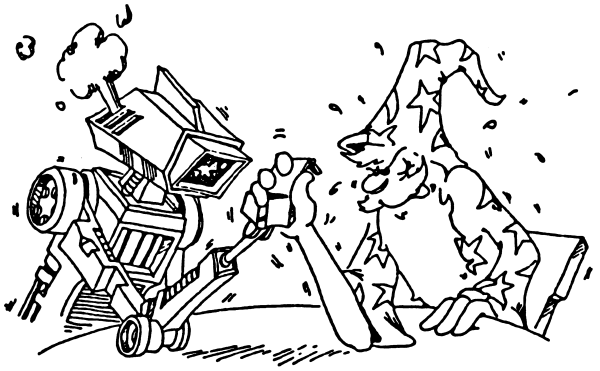
5 < 7 is pronounced "five is less than seven"

PRACTICE

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

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

A=B	T	F	B=C	T	F
A>B	T	F	B<C	T	F
A<B	T	F	B< >C	T	F
A=C	T	F	A< >B	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 56)

GUESSING GAME

```
10 REM --- GUESSING GAME ---
15 cls
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 cls
50 PRINT TAB(12);"Make a guess";
55 INPUT G
60 IF G< THEN PRINT "Too small"
65 IF G> 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 a disk, read Lesson 15.

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 _____

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"
```

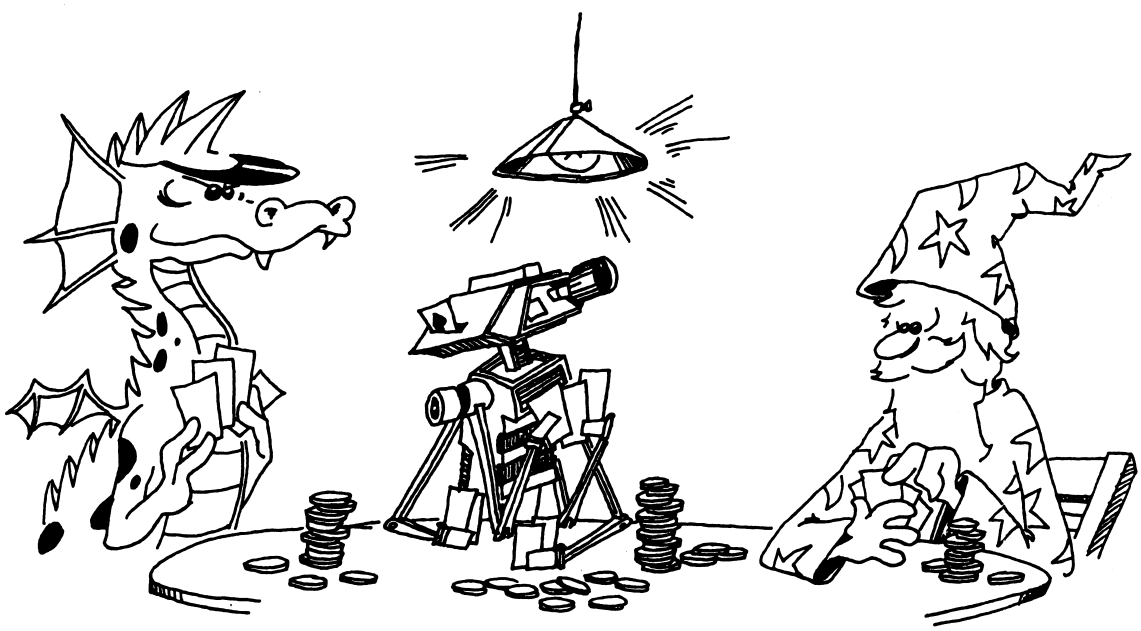
What will it print if line 10 is changed to:

```
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. Add to the GUESSING GAME program so that it prints "You're Hot" whenever the guesser is close to the right number.

5. 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 pseudo-random decimal numbers larger than 0 and smaller than 1.0. Such numbers are directly usuable 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 directly usable.

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

After extending the random number to a range larger than 0 to 1, conversion to an integer is desired. The INT function does this by simply truncating the number, "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 "rounding off" may be familiar to your student. INT will round off a number if you add 0.5 to it first.

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.)

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 cls
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.

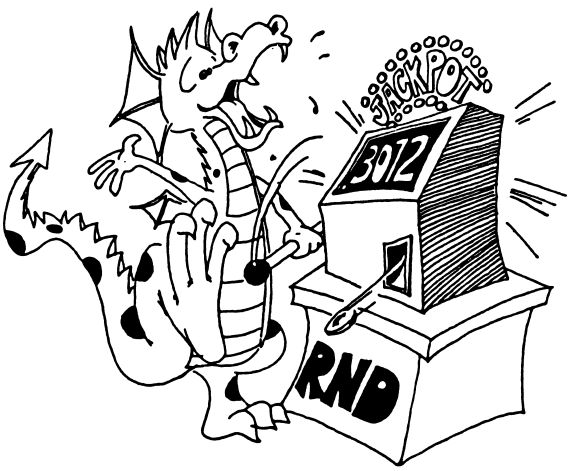
It doesn't matter what number you put in the parentheses, as long as it is larger than zero. I choose "9" because it is near the "( )" signs on the keyboard, making it easy to type (9).

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
30 PRINT N
40 IF N<46 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 may want whole numbers like 7 and 23 rather than decimal numbers like 7.03 and 23.62. So we use the INT function.

THE INT FUNCTION

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

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

HOW IT WORKS

Use this one line program:

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

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

ROUNDING OFF NUMBERS

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

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

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

RUN:

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

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

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 //////
20 cls
30 LET R=RND(9)
40 PRINT "Random number";TAB(15);R
50 LET S=R*6
55 PRINT "Times 6";          TAB(15);S
60 LET I=INT(S)
65 PRINT "Integer part";     TAB(15);I
70 LET D=I+1
75 PRINT "Die shows";        TAB(15);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(8))`

Here is how to save a lot of room.

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

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

RANDOM NUMBERS IN THE MIDDLE

Suppose your game has a funny die that shows only 6, 7, or 8 when you roll it.

RUN this: `10 LET D=INT(RND(9)*6)`

How it works:

expression	**makes numbers like** small	large
RND(9)	0.01	0.99
RND(9)*3	0.03	2.97
INT(RND(9)*3)	0	2
INT(RND(9)*3)+6	6	8

EVERY PROGRAM RUN IS DIFFERENT

Each time you start RUNning a program, you want different cards or dice to show.

But you will always get the same number unless you do something about it! Command RANDOMIZE early in the program to make the computer choose different random numbers which are different from those on the last run.

Example:

```
10 REM mixed up
20 RANDOMIZE
30 PRINT INT(RND(8)*100)
```

RUN the program several times without line 20, then several times with line 20.

RANDOMIZE asks you to input a number for a "seed."

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 SAVE TO THE DISK

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

The commands:

SAVE	LOAD
FILES	KILL

are introduced.

Other commands used in this chapter are:

NEW	REM
LIST	PRINT
CLS	

This lesson can be used anytime after Lesson 3.

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

However, your own 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 she is particularly proud of.

QUESTIONS:

1. What is a "file"?
2. How long can a file name be?
3. Can punctuation marks be in a file name? Can the file name have spaces in it?
4. How can you check that the program got on a disk OK?
5. What happens to the program already in memory if you LOAD another program?
6. Does the file name have to be the same as the program name?
7. If a program is put into a file, is it still in memory?

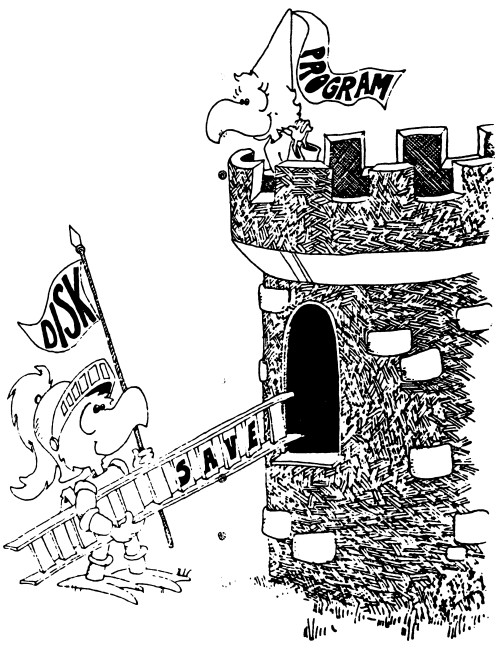
LESSON 14 SAVE TO DISK

ENTERING A PROGRAM

If you already have a program in the computer, skip to SAVING A PROGRAM.

If not, enter:

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



SAVING A PROGRAM

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

Enter:

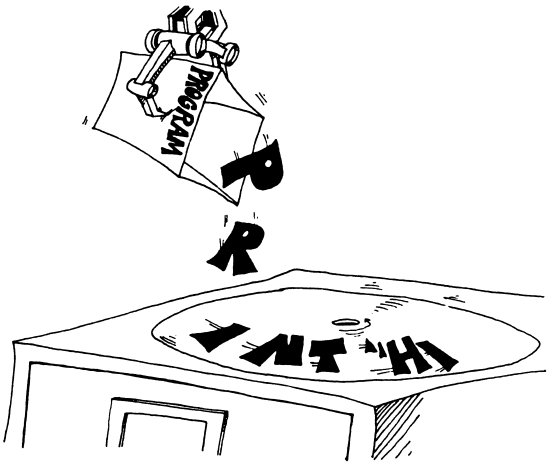
```
SAVE "HI"
```

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

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

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

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



A "LIFE SAVER"

If you mess things up, press the Ctrl Break keys to get back to normal.

THE FILES COMMAND

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

Enter: FILES

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

HI .BAS

FILE NAMES

Each file has a name that is one to eight characters long. The file name can be followed by an "extension." The extension starts with a period and has one, two, or three characters in it. If you have not given your file name an extension (and the file is a BASIC program), the computer automatically gives it the extension .BAS.

Try using this extension.

Enter: SAVE "hi.xyz"

Then enter FILES

You will see the name

HI .XYZ

in the list of files.

LOADING THE PROGRAM

Now that we are sure the program is on the disk, it is safe to erase it from memory.

Enter: NEW
Press: Ctrl Home
Enter: LIST

The LIST shows nothing because NEW erased the program from the computer's memory. Let's get the program back.

Enter: LOAD "hi"

We hear the whirring and see the red light, but is our program now in memory?

Enter: LIST to find out.

ERASING A FILE

So far, so good. But what if we change our minds and want to throw a file away?

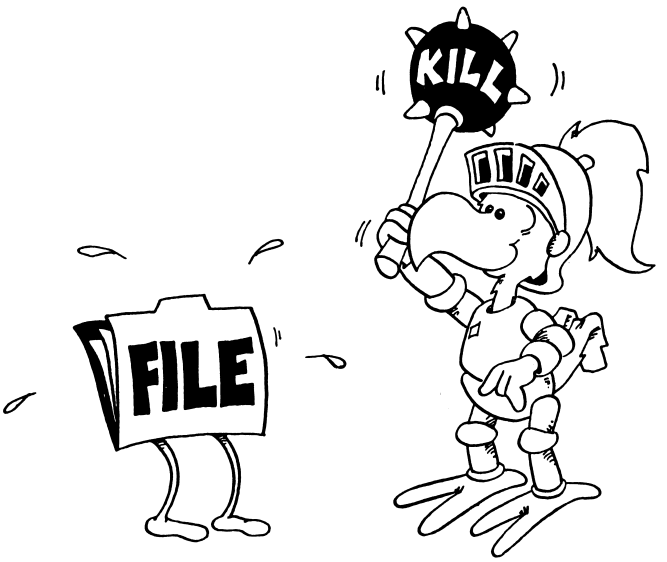
Use: KILL "hi.bas" and then

Enter: FILES

to see if it is really gone from the disk.

CAREFUL! You must put the period and three letter "extension" on the name or else KILL will give you the error message:

File not found



LEGAL FILE NAMES

The file name:

can have up to eight characters plus an extension.
can have numbers in it.
can have punctuation in it, BUT . . .
CANNOT HAVE A COMMA OR PERIOD IN IT (unless the period is in the extension).
can even have spaces in it.

The extension:

starts with a period.
has one, two or three characters in it.

Try saving the short program using these names, then do a FILES to see what names were used.

```
abcdefghijklm
a.bbbc
a!"#$%&'()*=
a123456789
a s d f
```

This name will give a Bad file number error message:

```
a.s.d.f.g
```

And this one a Too many files error message:

```
cat,dog
```

GOOD FILE NAMES

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

COMMANDS

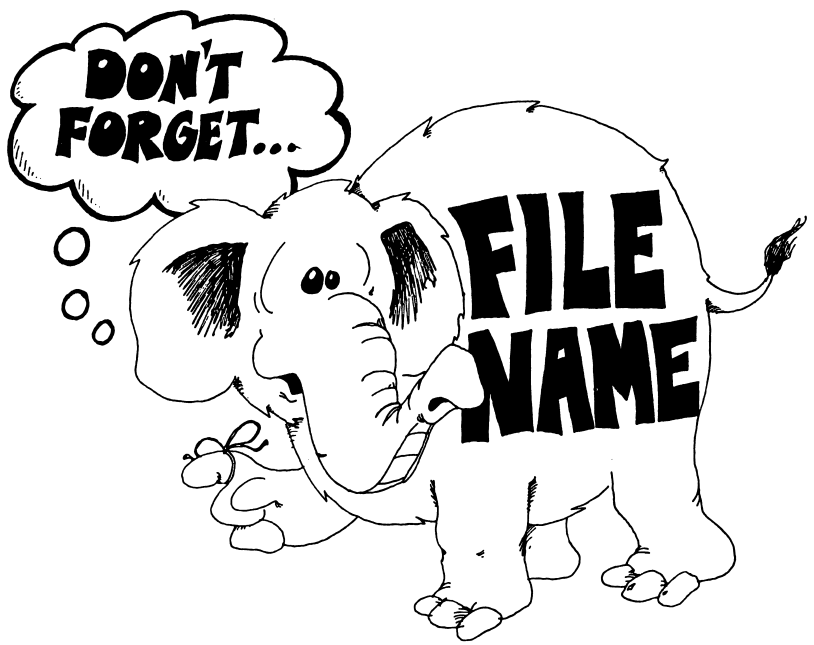
These four commands are used with files:

```
SAVE "filename"          or  SAVE "filename.ext"
LOAD "filename"          or  LOAD "filename.ext"
FILES
KILL "filename.ext"
```

Remember, "filename" is the name of a file, and "ext" is the extension.

Assignment 14:

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



INSTRUCTOR NOTES 15 SOME SHORTCUTS

?	used for PRINT
LET	omission
:	used between statements on a line
INPUT	used with a message
INPUT	error message
LIST X-Y	
THEN 33	instead of THEN GOTO 33
	commands compared to statements

Having reached RND and the SAVEing of programs on disk, the sprint is over. All the elements are in place so that the student can write substantial programs.

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

The colon 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.

The colon can mess up a program as well as improve it. Be careful about adding other statements onto a GOTO, a REM, or a IF line.

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

QUESTIONS:

1. What shortcut does the ? give?

2. How can you tell that the word LET is missing from a LET command?

3. An INPUT command has a message in quotation marks. What punctuation mark must follow the message quotes?

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

5. What is wrong with each of these lines?

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

6. If the computer prints ?Redo from start after the user answers an input, what three things could be wrong?

LESSON 15 SOME SHORTCUTS

A PRINT SHORTCUT

Instead of typing PRINT, just type a question mark.

Enter:
```
10 ? "HI"
LIST
```

The computer substitutes the word PRINT for the question mark.

A LET SHORTCUT

These two lines do the same thing:

```
10 LET A=41      and    10 A=41
```

also these two:

```
20 LET B$="HI"   and    20 B$="HI"
```

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

AN INPUT SHORTCUT

Instead of:

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

You can do:

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

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

ANOTHER INPUT SHORTCUT

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

RUN:

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

You see: LOCATION? on the screen.

You enter two numbers with a comma between them.

```
LOCATION? 5,6
```

Another example:

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

After the ? type:

```
APRIL,29,1983
```

ERROR MESSAGE IN INPUT

If you do not enter enough answers, or you entered too many, the computer says:

```
?Redo from start
?
```

and shows the flashing cursor. You must enter all the things asked for, with commas between them.

Example:

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

?MAY,1
?Redo from start
?May,1,1984
```

ANOTHER WAY TO GET AN ERROR MESSAGE

RUN: `10 INPUT N, A$` Try these pairs of answers:

```
1,  B
B,  1
1,  1
B,  B
```

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

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

A LIST SHORTCUT

There are five ways to use the LIST command:

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

A THEN SHORTCUT

Instead of: `10 IF A=B THEN GOTO 33`

Use: `10 IF A=B THEN 33`

A COLON SHORTCUT

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

Instead of

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

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

When you LIST the line, you see:

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

WHEN TO USE THE COLON SHORTCUT

Use the shortcut:

1. To make the program clearer.

   Put similar statements on the same line. Example:

   Instead of:

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

   write:

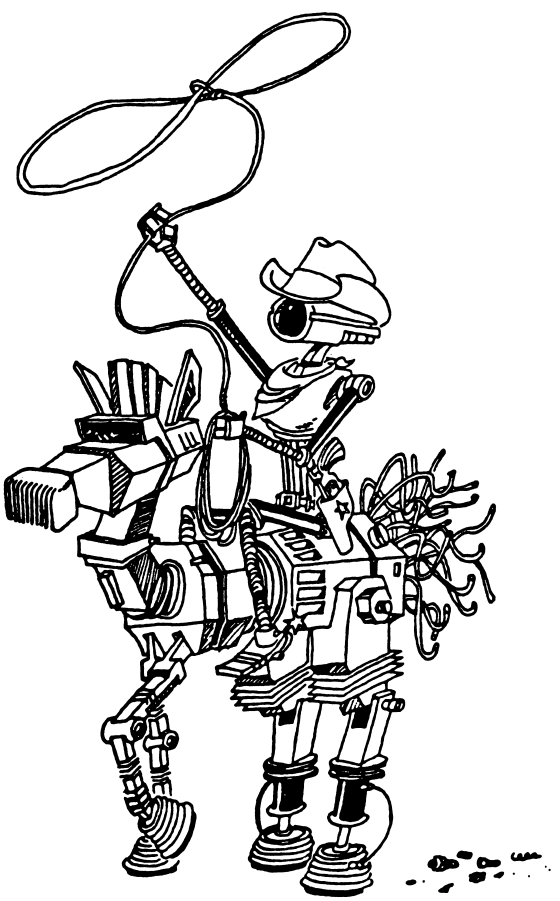
```
10 X=0:Y=0:Z=0
```

2. To make the program shorter.

3. To put a REM on the end of the line.

   Example:

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



THE COLON AFTER AN IF 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 that doesn't belong on the end of an IF line.

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 ignores everything else on a line after reading REM.

Correct:

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

Wrong:

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

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

A REM SHORTCUT

Instead of typing REM, you can just type a single quote (').

```
10 INPUT N$ ' The name of the user
```

is the same as:

```
10 INPUT N$: REM The name of the user
```

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, SAVE, LOAD

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 CLR:PRINT:Z=55
```

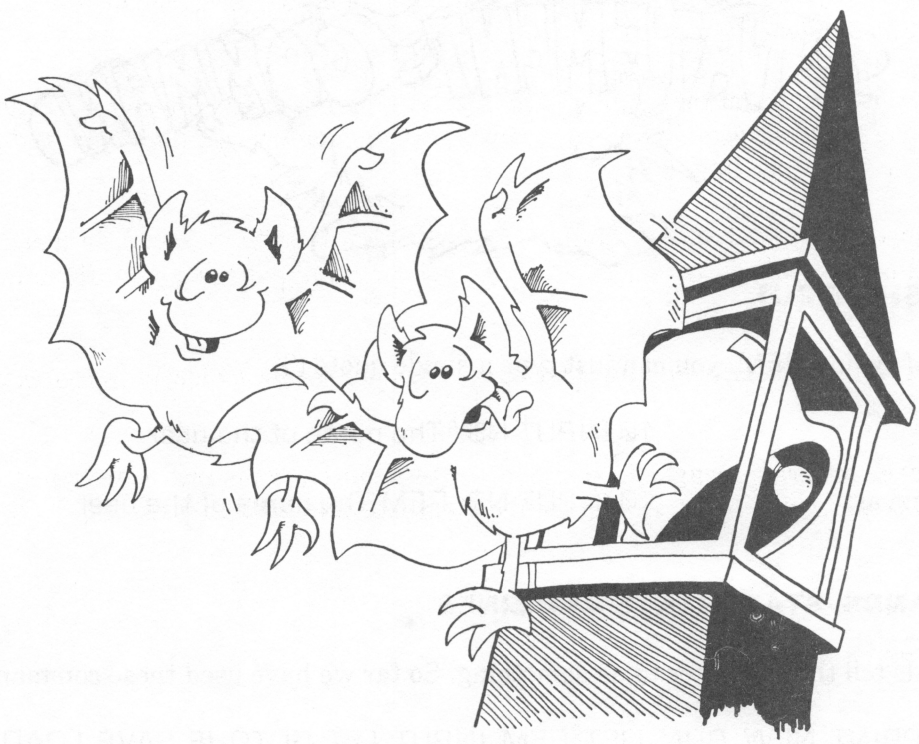
is a line with three statements.

Assignment 15:

1. Write a program that uses each of these shortcuts at least once.

2. Write a "vacation" program. It asks how much you want to spend. Then it tells where you should go or what you should do.

3. Write a "crazy" program that asks your name. The program has three funny ways of saying you are crazy. The program randomly chooses one of these and prints it after your name.



INSTRUCTOR NOTES 16 MOVING GRAPHICS AND LOCATE

The LOCATE statement is used to move the output cursor to any point on the screen.

LOCATE allows flexible manipulation of text on the screen and also allows a form of graphics.

To make effective use of LOCATE, you need to think of the screen as a 40 (or 80) character across by 24 line down array. Remember the phrase "row, column" (which row, which column) to get the order of arguments in the LOCATE R,C command. (We will find that the order is reversed when we get to graphics. There we need the phrase "X,Y" with X across and Y down.)

Although two lessons on the graphics commands are presented later, we show graphics here to get used to several ideas, one being the "down, across" graph paper, and another being how to make moving pictures.

Moving objects can be displayed and moved by using LOCATE in a loop. It is necessary to erase the old object with a space character of background color before the new object is drawn.

It is best to delay the erasing until just before the new object is drawn, to reduce flicker in the picture.

The Sketcher program brings all this together in an interesting way. However, it uses two ideas from later lessons, ASCII numbers and the INKEY$ variable.

QUESTIONS:

1. If you want to print the next word on line 12 at the left, what statement do you use?

2. If you want to print the next character on line 6, indented by 20 spaces, what statement do you use?

3. How can you print "never again!", wait a second then erase just the word "again"?

4. Show how to print the two words "FAT" and "CAT" on the same line with "CAT" printed first, starting at space 25, and then after a delay, "FAT" printed starting at 5.

LESSON 16 MOVING GRAPHICS AND LOCATE

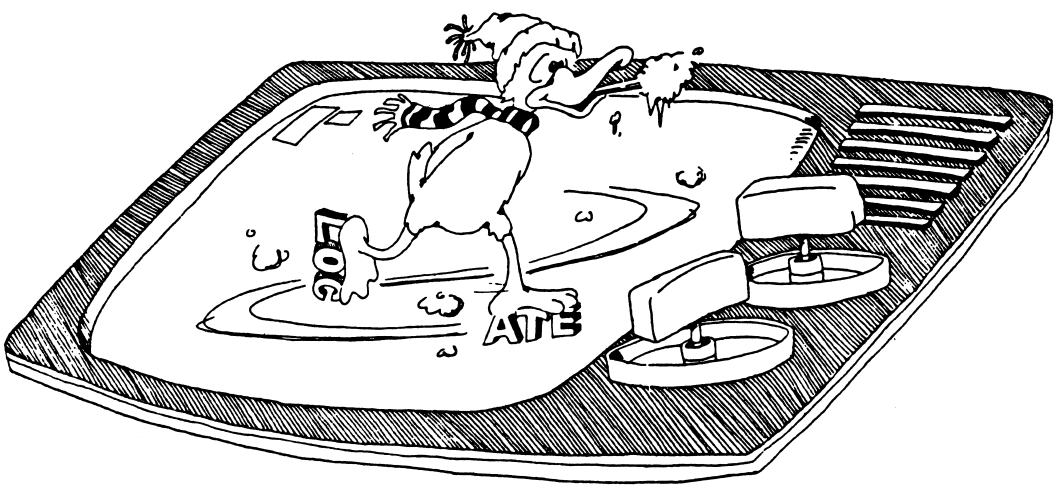
There is room for 24 lines of typing on the screen. The lines are numbered from 1 at the top to 24 at the bottom.

Each line can hold 40 or 80 characters (depending on whether you are using a color monitor or a green screen). They are numbered from one on the left to 40 or 80 on the right.

RUN this:

```
10 REM LOCATE demo
15 CLS
20 LOCATE 10,0:PRINT "line 10 first"
25 FOR T=1 TO 900:NEXT T
30 LOCATE 1,0:PRINT "line 1 next "
35 FOR T=1 TO 900:NEXT T
40 LOCATE 17,0:PRINT "line 17 last "
```

The first number in LOCATE tells which row the printing cursor will go to.



JUMPING ANYWHERE ON THE SCREEN

RUN:

```
10 REM column and row
15 CLS
20 INPUT "which row"; R
30 INPUT "which column"; C
40 LOCATE R,C:PRINT "*";
45 FOR T=1 TO 500:NEXT T
50 GOTO 20
```

Press Ctrl-Break to stop the program.

The second number in LOCATE tells which column the printing cursor will go to.

ERASING WHAT YOU WRITE

```
10 REM jumping here
15 CLS
20 C=INT(RND(9)*35):REM or 75 for green screen
25 R=INT(RND(9)*23)
30 LOCATE R,C: PRINT "here"
50 FOR T=1 TO 400:NEXT T
60 LOCATE R,C: PRINT "      "
80 GOTO 20
```

How do you make the program stop?

COLOR SKETCHER PROGRAM

This program lets you draw pictures in color.

Run:

```
10 ,                                  Color Sketcher
11 SCREEN 0,1: CLS: KEY OFF: WIDTH 40
15 GOTO 100                           ' initialize
20 X=20: Y=12                         ' center of screen
22 F=1: P=0: S=0
24 CLS
25 C$="d"
30 C$=INKEY$                          ' get a keystroke
35 IF C$=" " THEN 20                  ' erase screen
42 IF C$="d" THEN F=1                 ' draw mode
43 IF C$="e" THEN F=0                 ' not draw mode
44 IF C$<" " THEN 30
45 C=ASC(C$)
46 IF C>47 AND C<56 THEN P=C-48: COLOR P,0
48 IF F=0 THEN LOCATE Y,X: PRINT " ";
50 IF C$="i" THEN Y=Y-1               ' move dot up
```

```
51 IF C$="m" THEN Y=Y+1 '_____
52 IF C$="j" THEN X=X-1  '_____
53 IF C$="k" THEN X=X+1  '_____
60 IF X<1 THEN X=1   '     don't move above top of screen
61 IF Y<1 THEN Y=1       '_____
62 IF X>40 THEN X=40     '_____
63 IF Y>23 THEN Y=23     '_____
75 LOCATE Y,X: PRINT CHR$(219);
99 GOTO 30
100 '                    INSTRUCTIONS
102 '
103 COLOR 7,0            ' white letters
105 CLS
109 PRINT: PRINT: PRINT
110 PRINT " i, j, k, m keys move the dot": PRINT
120 PRINT " Space bar erases the picture": PRINT
130 PRINT " Press 'd' to draw lines": PRINT
140 PRINT " Press 'e' for no lines": PRINT
150 PRINT " Keys 0 to 7 for colors": PRINT
180 FOR T=1 TO 9000: NEXT T ' give time to read it
199 GOTO 20
```

When you enter this program, you can omit most of the REMs.

Save to disk. Fill in the blank REM's.

In line 30 the INKEY$ variable gets a keystroke from the keyboard. INKEY$ is explained in Lesson 29.

Line 35 erases the screen when the space bar is pressed.

Lines 42 and 43 set the variable F used in line 48. When F=1, the dot will leave a line behind it as it moves. When F=0 the dot flashes, but does not leave a line behind.

Line 45 changes C$ to an ASCII number. ASCII is explained in lesson 25.

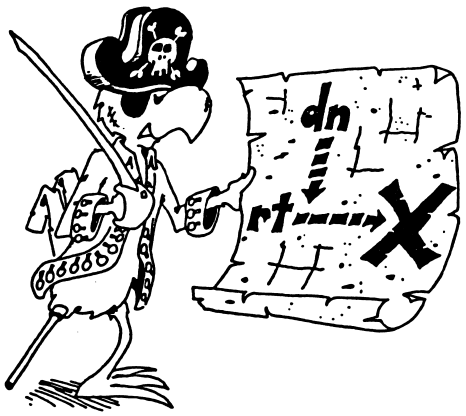
Line 46 picks a color number and calls it P. The ASCII number for "zero" is "48", for "one" is "49" and so forth. Subtracting 48 from C gives a number 0 to 7 to use as a color.

Line 48 uses F to see if the dot just put on the screen needs to be erased before another dot is put down. If so, a space is printed.

Lines 50 to 53 tell whether to move the dot up, down, right or left.

Lines 60 to 63 make sure the dot doesn't move off the screen.

Line 75 plots the dot.

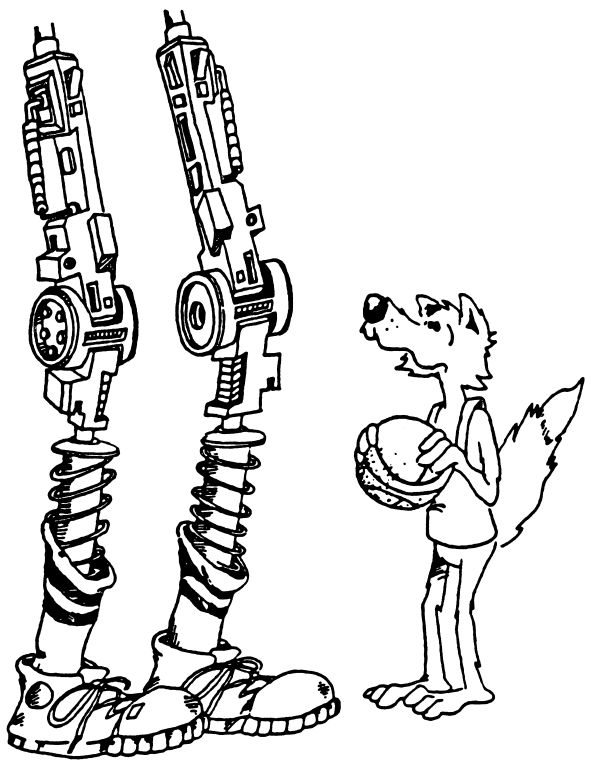


Assignment 16:

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 LOCATE to write your name in a large "X" on the screen.

3. Write a program that makes a bird flap its wings and fly across the screen. Make it peep as it goes.



INSTRUCTOR NOTES 17 FOR-NEXT LOOPS

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

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

IBM BASIC, unlike some other versions, detects whether the exit condition of a FOR . . . NEXT loop is satisfied before the loop is run even once. This makes for cleaner logic in your programs, but may make an unexpected bug if you copy certain non-IBM programs into IBM BASIC.

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 like any other numeric variable. The STEP value, the ending value, and the address of the first statement after the FOR are put on a stack.

From 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 like any other variable by BASIC, it can be used or changed in the body of the loop. Jumping into the middle of a loop is usually a disaster. Jumping out of a loop before reaching NEXT is commonly done, but in some cases (especially where subroutines are involved) may give hard-to-find bugs.

QUESTIONS:

1. What is the "loop variable" in this line?

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

2. Write a loop that prints the numbers from 0 to 20 by twos.

3. Write a "Ten Little Indians" program loop that prints from 10 down to zero Indians.

4. Write a pair of nested loops to print MINI in the outside loop and HA in the inside loop. Print three of the MINIs and for each of them print two of the HAs.

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

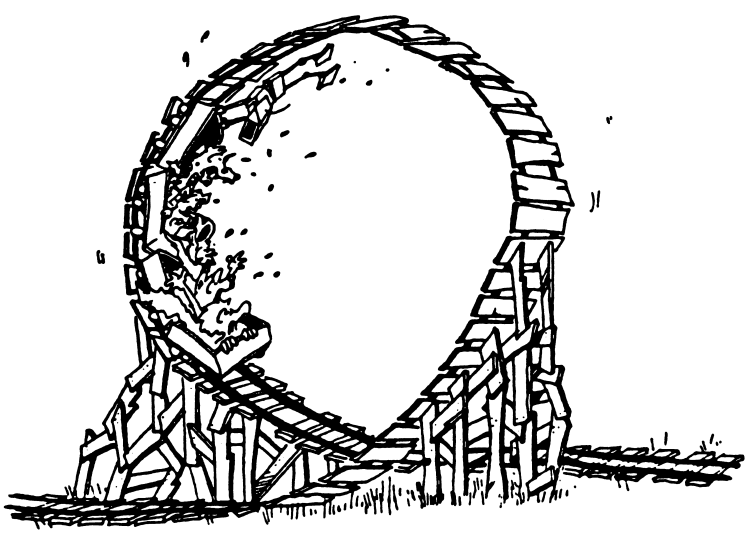
The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

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

MARK UP YOUR LISTINGS

Show where the loops are by arrows:

```
10 REM on paper
20 CLS
30 FOR I=0 TO 7
40 PRINT I
50 NEXT I
```



THE STEP COMMAND

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

```
30 FOR I=0 TO 30 STEP 2
```

Assignment 17A:

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

COUNT DOWN LOOPS

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

Try this:

```
10 '              *** APOLLO 11 ***
20 CLS
30 PRINT " T minus 12 seconds and counting"
40 FOR I=11 TO 0 STEP -1
50 PRINT I: BEEP
60 FOR J=1 TO 740: NEXT J ' timing loop
70 NEXT I
80 PRINT " All engines running. Lift off."
81 FOR J=1 TO 800: NEXT J: PRINT
82 PRINT " We have a lift off."
83 FOR J=1 TO 800: NEXT J: PRINT
84 PRINT " 32 minutes past the hour."
85 FOR J=1 TO 800: NEXT J: PRINT
86 PRINT " Lift off on Apollo 11.": PRINT
```

NESTED LOOPS

In this program, we have one loop inside another.

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

The inside loop is in line 60.

These are "nested loops." 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 740: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

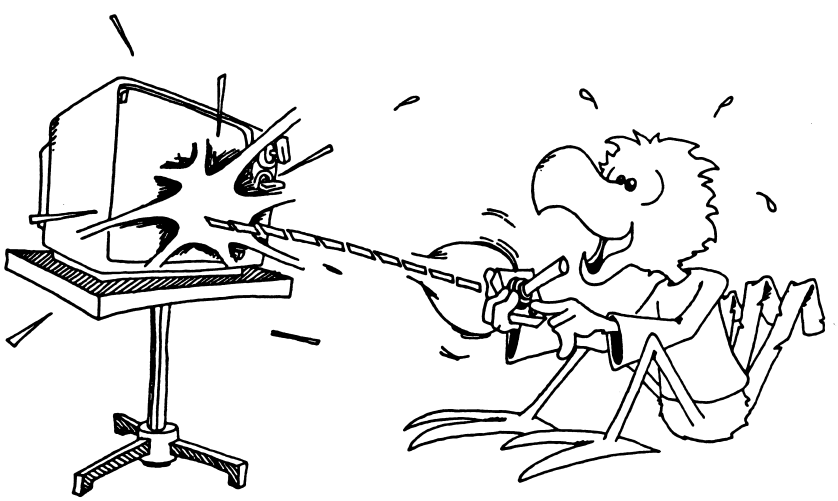
The inside loop must be all the way inside:

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
```



NONSENSE LOOPS

IBM BASIC skips loops that are not supposed to run.

RUN:

```
10 REM nonsense
15 PRINT I
20 FOR I=5 TO 3
25 PRINT I
30 NEXT I
40 PRINT I
```

This is a nonsense loop. Line 20 says that I should start at 5 and get bigger, until it is larger than 3. But it is bigger than 3 in the first place!

So you want the computer to skip the whole loop, jumping from line 20 to line 40 — it does.

In line 15, it prints 0. This is because the variable I has not been defined yet. When a variable that hasn't been defined is used, it is given the value "0".

Line 20 sets I to 5, then notices that the loop should not run, so it skips down to the line after the NEXT, which is line 40.

Line 40 prints the value of I, which is 5.

Assignment 17B:

1. Write a program that prints your name 15 times.

2. Now make it indent each time by two more spaces. It will go diagonally down the screen. Use TAB in a loop.

3. Now make it write your name 23 times, starting at the bottom of the screen and going up. Use LOCATE in a loop.

4. Now make it write your name on one line, your friend's name on the next and keep switching until each name is written five times.

INSTRUCTOR NOTES 18 DATA, READ AND RESTORE

You put data in the DATA statement at the time you write the program. READ gets data from the DATA statements and RESTORE puts the pointer back to the beginning of a DATA statement.

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 is in the box.

You must READ the data to be able to use it. It must be read in order. If you want to skip some data that is in a given DATA statement, 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.)

In the IBM-PC you can skip data by arranging it in different DATA statements, and pointing to the one you want with a RESTORE nnn statement, where nnn is the line number of the DATA statement.

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. Moreover, it is 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? Where to put the READ statements?

3. Can you put numeric data and string data in the same DATA statement?

4. Can you change the items in a DATA statement while the program runs?

5. The idea of a "pointer" helps in thinking about DATA statements. Explain how.

LESSON 18 DATA, READ AND RESTORE

TWO KINDS OF DATA

There are two kinds of data in your programs:

1. The data you INPUT through the keyboard.

```
10 REM First kind of data
20 CLS
30 PRINT "Your pet peeve"
35 INPUT P$
37 CLS
40 PRINT "Really!"
50 PRINT "You don't like ";P$;"?"
```

   In this program, P$ is data entered by the user while 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 CLS
30 X=57
40 Y$= "FLAVORS"
50 PRINT X;Y$
```

   In this program, X and Y$ are data stored in the program by the programmer when 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
20 CLS
30 DATA Sunday,Monday,Tuesday,Wednesday,Thursday,
   Friday,Saturday
40 READ D1$,D2$,D3$,D4$
60 PRINT D1$,D2$,D3$,D4$
```

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

STRANGE RULES

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

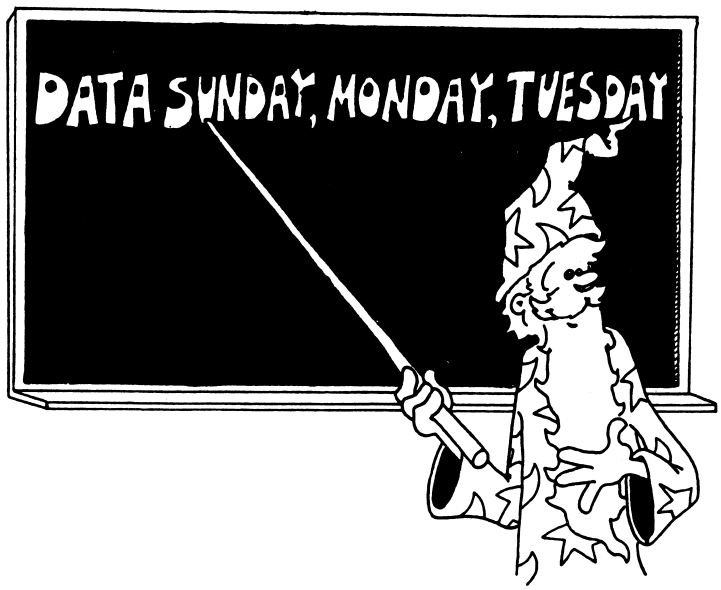
Do this: Change line number 30 in the above program to line number 90. RUN the program. It works just the same.

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

Do this: Break the DATA statement into two:

```
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" AT DATA COMMAND

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

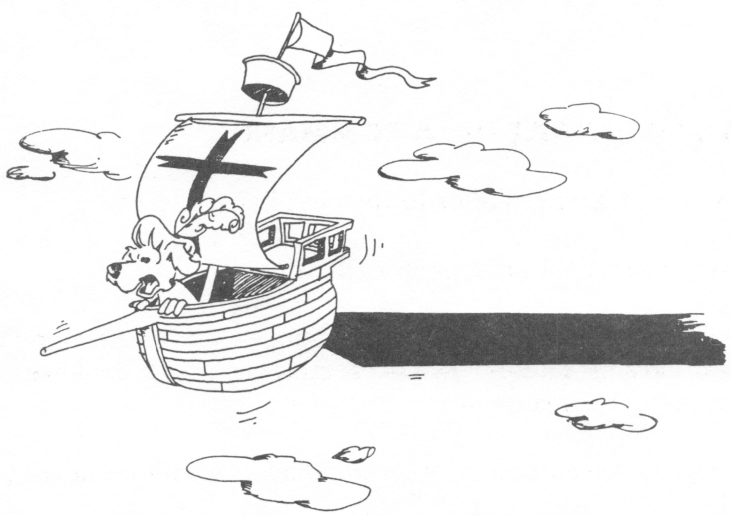
When the pointer reaches the last item in the last DATA statement in the program, there are no more items left to read. If you try to READ again, you will see an error message:

```
Out of data error in
```

BACK TO SQUARE ONE

At any point in the program you have only three 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.
3. You can command RESTORE nnn: The nnn is a line number of a DATA statement. The READ pointer is put on the first item in that DATA statement.



MIXTURES OF DATA

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

You must be careful to have the correct kind of variable to match the kind of data in your READ command.

Correct:

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

Wrong:

```
70 DATA 77,fuzz
75 READ B$          "OK,"  B$ box holds 77
80 READ N           Type mismatch error in 70
```

You can't put "fuzz" in a number box.

Assignment 18:

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

2. Write an invisible message program. Write messages in two different colors, say red and white. When the screen is red, the white message is visible, but the red one is not. A new message becomes visible when the screen changes to white.

INSTRUCTOR NOTES 19 SOUND

The SOUND command makes a tone of specified pitch and duration.

Remember that the BEEP command just makes a short attention getting sound whose pitch and length are not under programmer control.

The IBM-PC computer does not have full sound effects capability, mainly because it lacks a "white noise" generator. It also lacks an "envelope" generator that could control the attack and decay of notes. It can, however, make musical tones that are accurately in pitch, because pitch can be specified to five digits of precision.

Two arguments are needed in the SOUND command. The first is the pitch in cycles per second, or Hertz (Hz). This variable takes values from 0 to 65535. Of course, human perception is in the range 50 to 20000 Hz., or less in older folks.

The second argument is a length number from 0 to 65536. The number is the duration of the sound in terms of a clock that "ticks" 18.2 times a second.

Music is most easily made if you use the PLAY command described in Lesson 23.

When using sound in graphics situations, you get the most elaborate effects if you intersperse the sound commands with the "move the graphics" commands.

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

QUESTIONS:

1. What does the statement SOUND 500,30 do?

2. Which pitch numbers give deep sounds? Which give high notes?

3. What is the largest number that you can use for making a long note?

LESSON 19 SOUND

The IBM-PC computer has three sound commands.

BEEP SOUND PLAY

BEEP plays one note, always the same note for the same length of time. Use it to get the user's attention.

SOUND plays a single note. You can choose both the pitch (from very low to very high) and the length of the note.

We will tell you about PLAY in Lesson 23.

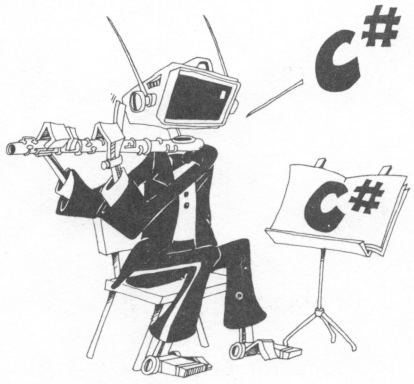
All the sounds are played through the speaker in the computer.

PLAYING ONE NOTE

RUN:
```
10 REM sound
15 FOR I=1 TO 50 STEP 10
20 SOUND 1,50
25 NEXT I
30 FOR I=50 TO 1000 STEP 100
35 SOUND I,50
40 NEXT I
50 FOR I=1000 TO 5000 STEP 1000
55 SOUND I,50
60 NEXT I
```

The first number after SOUND is the pitch. Any number from 0 to 65535 can be used. Larger numbers give higher notes. In fact, the number gives the pitch in Hertz (cycles per second).

You can play simple musical tunes using SOUND statements.



NOTES OF DIFFERENT LENGTH

The second number in the SOUND statement gives the length of the note. Any number from 0 to 65535 can be used.

A length of 20 is about one second, 40 is two seconds, and so on.

MAKING MUSIC

Here is a tempered scale of musical notes:

note	number
C (below middle C)	130.810
C #	138.592
D	146.833
D #	155.564
E	164.814
F	174.614
F #	184.997
G	195.998
G #	207.653
A	220.000
A #	233.082
B	246.942
C (middle C)	261.626
C #	277.183
D	293.665
D #	311.127
E	329.628
F	349.228
F #	369.995
G	391.996
G #	415.305
A	440.000
A #	466.164
B	493.883
C (above middle C)	523.251

Try this: `10 SOUND 440,100`

This plays "A above middle C" for about five seconds.

You may not be able to hear anything for pitch numbers much above 10000.

MUSICAL RESTS

To make a rest in your music, use:

```
SOUND 30000,L
```

Where L is a number to tell how long the rest is.

```
10 REM One Voice
20 PRINT " one voice"
22 INPUT " how long? <1 TO 200> ",D
25 INPUT " what pitch <37 TO 10000>"; P
40 SOUND P,D
45 FOR T=1 TO 1000:NEXT T
50 GOTO 20
```

SOUND EFFECTS

Run:

```
10 REM sound effect
20 PRINT" what does this sound like?"
30 FOR I=2000 TO 500 STEP -30
40 SOUND I,1
50 NEXT I
```

Assignment 19:

1. Make the sound of:

   a truck horn
   a laser gun

2. Write a program to play a short tune, like "Mary Had A Little Lamb." Use a DATA statement to store the pitch numbers.

INSTRUCTOR NOTES 20 COLOR

This lesson explains the COLOR command.

A different style of drawing is used in color than in black and white. Color drawings look best when large areas are painted in one color. When using individual characters, letters or graphics, some color combinations do not give very crisp results. You should experiment to find suitable combinations.

If you have a color monitor, the student should set up its controls for pleasing color by following the instructions in the lesson. Some monitors do not allow all the colors produced by the computer to show. The 16 tints (including black and white) allow very colorful effects to be produced.

Drawing pictures character by character is quite tedious. Use of graph paper to block out the picture first is often helpful. We 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. See the "Jumping J" program in a latter lesson.

QUESTIONS:

1. What does the command COLOR Ø,7 do?

2. How do you put red letters on a black background?

3. If you drew a blue ball on a white background, how would you erase the ball?

4. How many colors are there to choose from?

5. What range of numbers are allowed for X and Y in the command LOCATE X,Y?

LESSON 20 COLOR

ADJUSTING THE MONITOR FOR COLOR

If your monitor is black and white, skip to RAINBOW BALL.

Otherwise load the colorbar program from your student disk and RUN it.

```
load"colorbar.bas"
RUN
```

THE COLORS ARE:

0 BLACK	8 GREY
1 BLUE	9 LIGHT BLUE
2 GREEN	10 LIGHT GREEN
3 CYAN (BLUE-GREEN)	11 LIGHT CYAN
4 RED	12 LIGHT RED
5 MAGENTA (REDISH-PURPLE)	13 LIGHT MAGENTA
6 BROWN	14 YELLOW
7 WHITE	15 INTENSE WHITE

Adjust the tint control on your color monitor so that each colored bar is the correct color. Try to get the correct shade for the yellow, cyan, and magenta bars. Some monitors will not allow all colors to be correct at the same time.

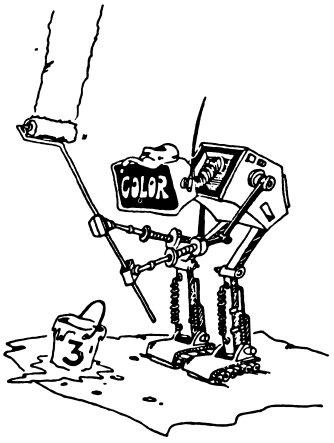
THE COLOR STATEMENT

You already know how to use the COLOR statement to make a colored background (with black writing on it).

Like this:

```
10 REM colored backgrounds
20 COLOR 0,4
30 PRINT "black letters on red"
```

The second number after the word COLOR is the color of the background.



COLORED LETTERS ON A COLORED BACKGROUND

The COLOR statement has three numbers:

character color	0-31
background color	0- 7
border color	0-15

Example: COLOR 4,2

means red (=4) letters on a green (=2) background

RUN:

```
10 REM colored letters on red
15 FOR I=0 TO 7
20 COLOR I,4: REM "4" is "red"
30 PRINT "colored letters"
40 NEXT I
```

Notice that some colors do not look good on a red background.

Now try all colors of letters on all background colors. Add to the above program:

Add:

```
12 FOR J=0 TO 15
20 COLOR I,J
45 FOR T=1 TO 2000:NEXT T
50 NEXT J
```

FLASHING WORDS

The first number in the COLOR statement gives different colored letters when numbers 0 through 15 are used. If numbers 16 through 31 are used, the letters flash on and off!

Change: 15 FOR I=0 TO 31

and RUN again.

Assignment 20A:

1. Add a loop to the above program so that the border color changes. You need a color statement with three variables in it, like:

   COLOR I,J,K

   and K changes in a loop, changing the border color.

JUMPING RAINBOW SENTENCE

RUN:

```
10 REM jumping rainbow sentence
15 COLOR 7,0: CLS: SCREEN 0,1
20 FOR I=1 TO 9
30 READ W$
40 X=INT(RND(9)*23)+1
41 Y=INT(RND(9)*39)+1
50 C=INT(RND(9)* 6)+1
51 COLOR C,0
55 LOCATE X,Y
60 PRINT W$
65 FOR T=1 TO 1000: NEXT T
70 NEXT I
80 COLOR 7,0
99 DATA watch,out,your,pop,is,spilling,on,your,keyboard
```

RAINBOW BALL

```
10 REM rainbow ball
15 CLS: SCREEN 0,1
16 C=1
20 FOR I=1 TO 39
30 LOCATE 12,I
31 PRINT "O";          'print ball
40 FOR T=1 TO 50: NEXT T
41 LOCATE 12,I: PRINT " ";          'erase ball
50 C=C+1: IF C>15 THEN C=1          'change color
51 COLOR C,0
60 NEXT I
```

The loop from 20 to 60 moves a ball across the screen, changing its color as it goes.

Lines 30 and 31 put the ball on the screen in a new spot.

Then line 40 waits for a moment.

Line 41 then erases the ball that was just printed.

Line 16 sets the starting color to "blue," the number 1.

Line 50 increases the color number by one each time the ball is moved. When the color reaches "intense white", number 15, it is changed back to the first color, blue.

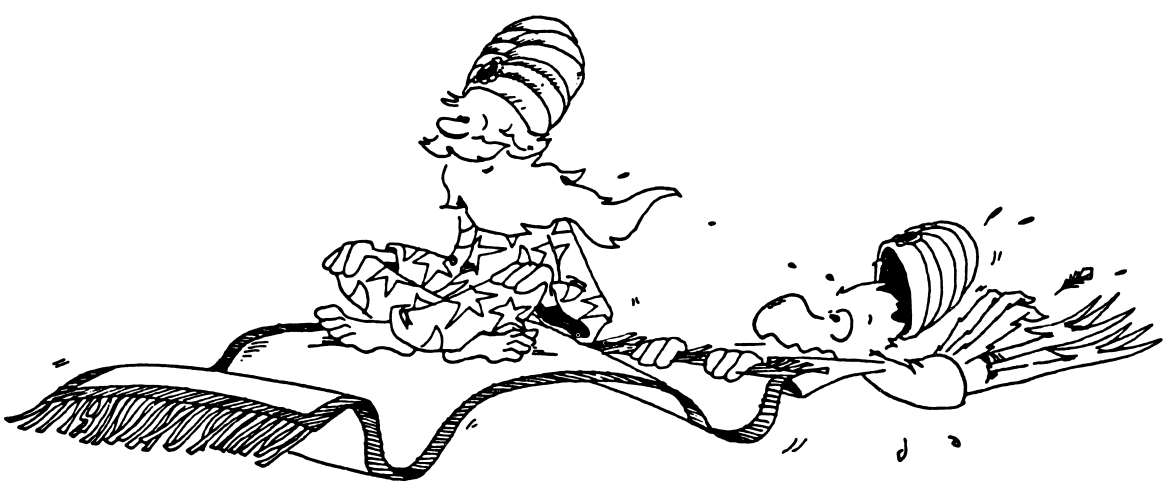
Assignment 20B:

1. Change the RAINBOW BALL so it falls, instead of moving across the screen.

2. Add to the number guessing game in Lesson 13 so that a large 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.

3. Write a program to draw "Sinbad's Magic Carpet." Let the user choose how many colors in the rug, and what colors, then draw a pattern on the screen.

4. Make a program to write your name in a big "X" on the screen. Make the "X" cover a red heart, and then make both move flashing across the screen.



INSTRUCTOR NOTES 21 DRAWING PICTURES

This lesson illustrates SCREEN, LINE, CIRCLE and PSET for line drawings.

The IBM-PC can make medium resolution graphics (320 x 200 dots on the screen) in four colors.

First you have to "warn" the computer that graphics commands will be used by saying SCREEN 1. SCREEN Ø tells the computer to go back to text only display. You can still PRINT and use LOCATE in graphics 1 mode, a real convenience.

If you have a green screen monitor, SCREEN 2 gives higher resolution (a 640 by 200 screen) and the restriction to only two colors doesn't hurt. We will not describe graphics 2 mode per se. It is similar to graphics 1 mode; consult the IBM BASIC manual.

Think of the screen as a graph with the axes crossing at the "home" position (upper left). The X axis runs horizontally with X values from 0 to 319. Y is vertical with values from 0 to 200. The commands PSET, LINE, and CIRCLE all use the notation (X,Y) for points needed in the command.

Give the center point and the radius for CIRCLE. In the next lesson, a color is added. Read the IBM BASIC manual to see how to make "pie" charts with sectors of a circle.

LINE needs two points, the start and the end. The next lesson tells how to pick a color for the line, and how to use LINE to make rectangles.

PSET plots a single point.

BASIC does not object if part of the picture you specify is off the screen. For example, you can say CIRCLE (-2Ø,-3Ø),12Ø and put part of a circle on the screen. (Even the center is off the screen!)

Two powerful graphics commands will be omitted from this book. They are the DRAW command, and the PUT and GET commands which allow animated graphics.

QUESTIONS:

1. Where on the screen will PSET 16Ø,1ØØ put a dot?

2. How do you draw a circle centered on the screen?

3. How many dots can you fit across the screen?

4. How many dots can you fit down the screen?

5. How do you draw a large "X" on the screen?

LESSON 21 DRAWING PICTURES

The SCREEN command fixes the screen so you can use the PSET, LINE, and CIRCLE commands to draw pictures. It needs the IBM Advanced BASIC and the Color/Graphics Monitor Adapter.

The CIRCLE command lets you draw a circle on the screen.

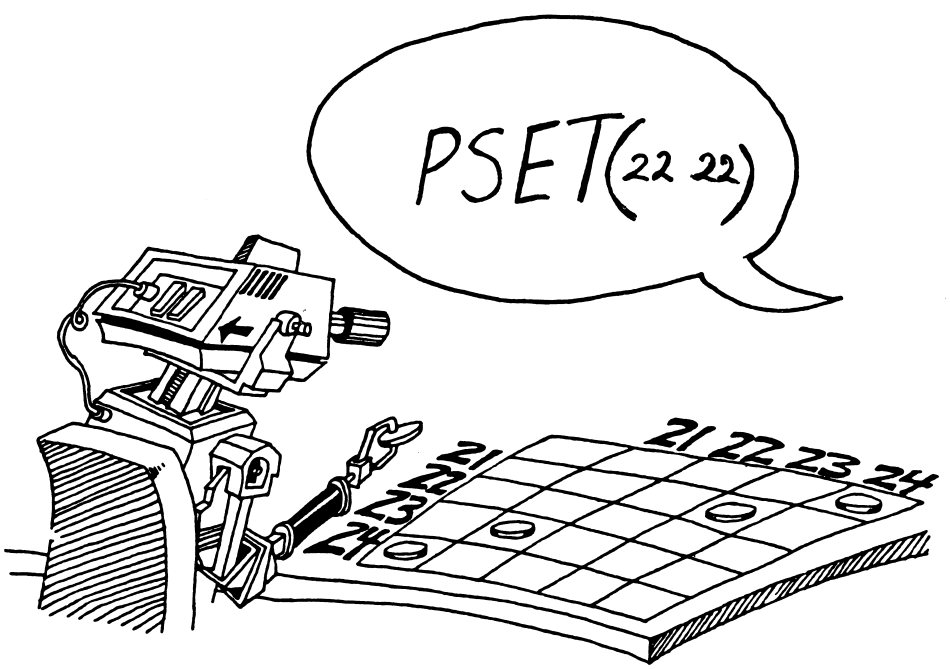
The LINE command lets you draw a line on the screen.

The PSET command lets you put dots on the screen.

RUN:
```
10 REM circle, line, and dots
12 SCREEN 1: CLS
20 CIRCLE(160,100),90
30 LINE (0,0)-(320,200)
40 FOR I=50 TO 150 STEP 10
50 PSET (I,150-I)
60 NEXT I
```

THE SCREEN COMMAND

SCREEN 1	Make the screen ready for medium resolution graphics 320 points across by 200 points down.
SCREEN 0	Change the screen back to text only 40 letters across by 25 lines down.



THE PSET COMMAND

The screen is like a sheet of graph paper with 320 squares across and 200 squares down.

PSET (X,Y) puts a dot on the screen. X tells how far from the left. Y tells how far down from the top.

PSET 160,100 puts a dot in the center of the screen. Try it.

PSET 319,199 puts a dot in the lower right corner of the screen. Try it. If you can't see the dot, maybe it is off the edge of your screen. Try PSET (310,190) instead.

How do you tell the computer to put a dot in the other three corners of the screen?

upper left _____

upper right _____

lower left _____

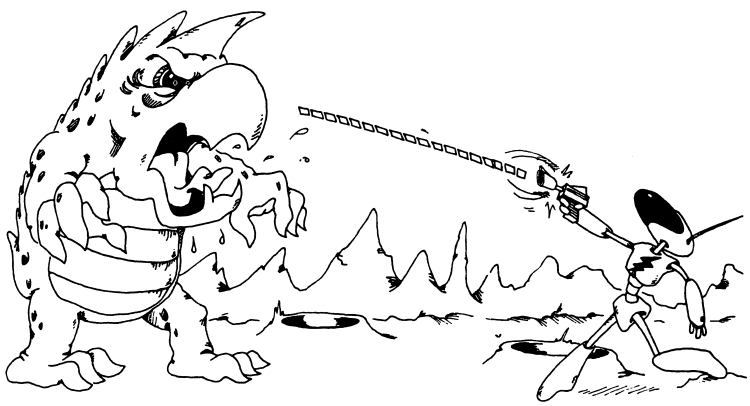
lower right PSET (319,199)

Try them and see if they work.

RUN:

```
10 REM measles
12 SCREEN 1: CLS
20 FOR I = 1 TO 100
30 X=INT(RND(9)*320) ' a number from 0 to 319
31 Y=INT(RND(9)*200) ' _____
33 PSET(X,Y)
35 FOR T=1 TO 50: NEXT T
40 NEXT I
50 SCREEN 0
```

fill in the blank REM on line 31



THE LINE COMMAND

The LINE command draws a line between two points on the screen.

RUN:
```
12 SCREEN 1: CLS
20 LINE (20,150)-(310,10)
```

RUN:
```
10 REM splat
12 SCREEN 1: CLS
15 FOR I=1 to 25
17 X=INT(RND(9)*300)+10
18 Y=INT RND(9)*180)+10
20 LINE (160,100)-(x,y)
30 NEXT I
```

THE CIRCLE COMMAND

The CIRCLE command draws a circle with a center point given by (x,y) and a radius r.

```
10 REM circles
12 SCREEN 1: CLS: ?: ?: ?
20 INPUT " center at x= <0 to 320> ";x
22 INPUT " center at y= <0 to 200> ";y
24 INPUT " radius r= <0 to 320> ";r
30 CLS: CIRCLE(X,Y),R
35 LOCATE 12,1: PRINT" radius";R
40 FOR T=1 to 2000: NEXT T
99 GOTO 10
```

Try these values for x, y and r:

x	y	r
160	100	95
0	0	150
160	0	120
–20	–20	70

Try other values that you pick.

MIXTURES OF LETTERS AND GRAPHICS

You can mix them all up on the screen, using PRINT, PSET, LINE, and CIRCLE in the same program. You can use LOCATE to place the printing where you want it on the screen.

Assignment 21:

1. Use CIRCLE to draw a snowman, LINE for his arms, and PSET for eyes, nose, mouth and buttons.

2. Use LINE to draw your school's initials. SAVE to disk.

INSTRUCTOR NOTES 22 COLOR GRAPHICS

This lesson tells how to paint in four colors (medium resolution graphics) and how to use the LINE command to make rectangles.

You cannot just pick four colors by the numbers introduced in previous lessons. The IBM-PC uses the idea of a "palette," to which we add the idea of "brushes."

There are only two palettes allowed, 0 and 1, each with three fixed colors, numbered 1, 2, and 3. Palette 0 has green, red and brown. Palette 1 has cyan, magenta and white. Palette 1 colors are obtained by adding blue to the palette 0 colors. [This may seem strange as you remember your "water color" days in kindergarten, but "brown" is supposed to be "yellow," and when adding colored lights instead of pigments, yellow (equals red plus green) plus blue does give white.]

To the three fixed colors on a palette, the user specifies the background color, 0-15, as before. Brushes: Brush "0" dips into the background color. Brushes 1, 2, and 3 dip into the colors 1, 2, and 3 on the palette.

Colors can be used for outline and for solid colors. The outline color is placed as a brush number in the PSET, CIRCLE, or LINE statements. The solid color is made by the PAINT command which has a point (x,y), a brush number for the solid color and finally, a brush number for the border to be filled. The border number is important in the PAINT command because it specifies at which color (brush number) the filling in of color will stop.

The LINE command also allows you to draw rectangular "boxes." Just add the letter B (for "box") on the end of the command. Adding the letter F (for "fill") makes the box a solid color.

QUESTIONS:

1. How do you make a rectangle with upper left corner at (30,10) and lower right corner at (60,110)?

2. How many palettes can you choose from? What colors are on each one?

3. What does the (30,40) mean in the statement:

```
PAINT (30,40),3,1
```

   What does the "3" mean?
   What does the "1" mean?

4. What does the 2 mean in:

```
CIRCLE (90,112),55,2
```

5. How would you paint the above circle a solid white? (Your answer should say something about "palette," something about "borders" and something about PAINT.

LESSON 22 COLOR GRAPHICS

PICK YOUR PALETTE

RUN:

```
10 REM dueling artists
12 CLS: SCREEN 1
20 FOR B=0 TO 15
22 FOR P=0 TO 1                    ' palette 0 or palette 1
25 COLOR B,P                       ' B is background color
30 CIRCLE (50,50),70,3             ' outlined in color 3
31 PAINT (50,50),1,3               ' color 1 to outline 3
32 CIRCLE (80,99),80,1             ' outlined in color 1
33 PAINT (91,120),2,1              ' color 2 to outline 1
34 LINE (90,70)-(280,150),3,BF     ' solid color 3
49 FOR T=1 TO 500: NEXT T
60 NEXT P,B
98 SCREEN 0
```

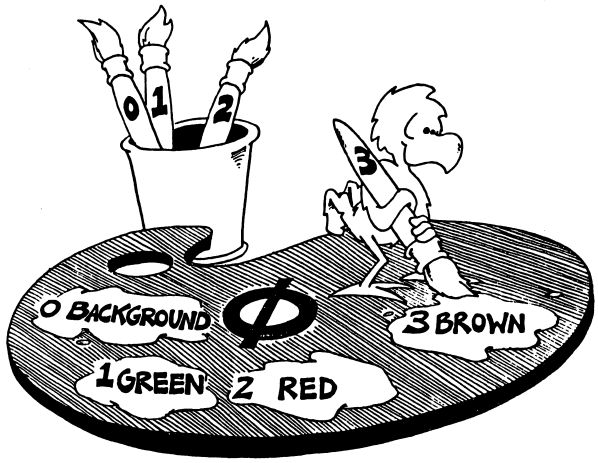
After you give the SCREEN 1 command to use graphics, pick a palette of colors with the COLOR command.

Example: 15 SCREEN 1: COLOR 13,1

background color 13, palette number 1

There are two palettes. Each palette has only four colors; the background color you choose and three more that come with the palette.

brush	palette 0	palette 1
0	background	background
1	green	cyan (blue-green)
2	red	magenta (purple)
3	brown	white



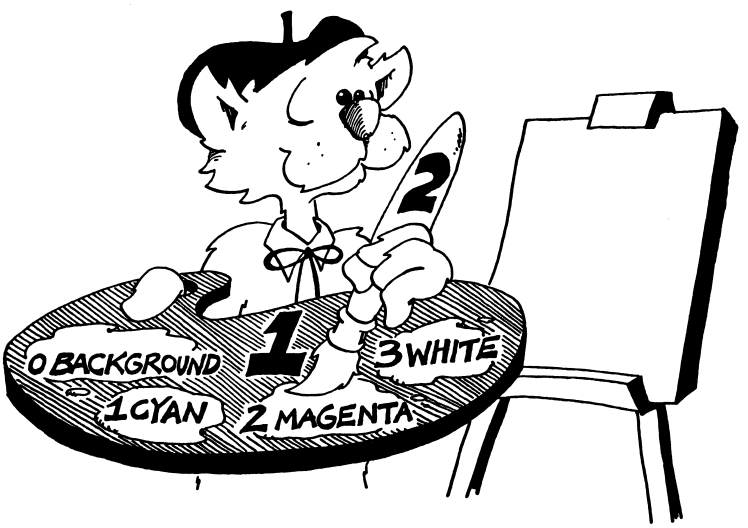
DIP YOUR NUMBERED BRUSH AND PAINT AN OUTLINE

After you choose which background color and which palette you want, you can paint a colored line, circle, or dot.

Example:

```
20 SCREEN 1:COLOR 15,0          'white background, palette 0
30 LINE (20,20)-(100,100),2     'red line on white
40 CIRCLE(160,100),50,1         'green circle on white
50 PSET (160,100),3             'brown dot on white
```

The brush number is the last number in the LINE or PSET command and tells what color you will paint with on the palette. The brush number follows the radius in the CIRCLE command.



COLORED BOX IN OUTLINE

The LINE command can also draw a rectangle. Just add the letter B (for box) after the brush number.

RUN:

```
20 SCREEN 1: COLOR 15,1         ' palette 1 on white
30 LINE (60,80)-(120,160),1     ' cyan line
40 LINE (50,70)-(130,170),2,B   ' magenta box
```

enter SCREEN 0 to get back to text mode

SOLID COLOR FOR THE BOX

Change line 40 above to paint inside the box.

Change:

```
40 LINE (50,70)-(130,170),2,BF
```

The letter F stands for fill and means you fill in the box outline with color.

SOLID COLORS FOR YOUR DRAWINGS

The PAINT command fills in any shape outline with color. Make the outline by drawing lines with LINE, or circles with CIRCLE.

RUN:

```
10 REM paintbox
12 KEY OFF: SCREEN 1: COLOR 7,0
19 REM a triangle outline in green
20 LINE (50,50)-(100,0),1
22 LINE (50,50)-(100,100),1
24 LINE (100,0)-(100,100),1
30 LOCATE 12,20: PRINT"GREEN OUTLINE"
40 FOR T=1 TO 1000: NEXT T
50 LOCATE 14,20: PRINT"USE BRUSH 2 TO FILL WITH RED"
60 FOR T=1 TO 1000: NEXT T
70 REM Brush 2 (red). Fill to green (1) boundry.
71 REM Place point inside boundry
72 PAINT (60,60),2,1
80 FOR T=1 TO 1000: NEXT T
81 SCREEN 0
```

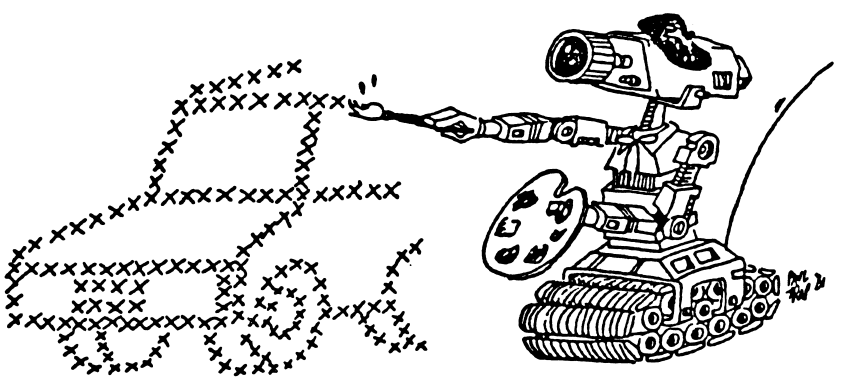
```
10 REM vanishing dots
12 SCREEN 1:KEY OFF:COLOR 15,0
14 RANDOMIZE:CLS
20 FOR X= 30 TO 320 STEP 50
22 FOR Y= 30 TO 200 STEP 40
30 CIRCLE(X,Y),19,1
40 NEXT Y,X
50 FOR I=1 TO 1000
52 X=INT(RND(9)*320)
54 Y=INT(RND(9)*200)
56 C=INT(RND(9)*4)
58 PAINT(X,Y),C,1
60 NEXT I
```

```
10 REM CIRCLES
12 RANDOMIZE
15 SCREEN 1
16 COLOR 15,1
18 R=INT(RND(9)*80)+5
20 X=INT(RND(9)*200)+50
22 Y=INT(RND(9)*100)+50
24 C=INT(RND(9)*4)
26 B=INT(RND(9)*4)
30 CIRCLE(X,Y),R,B
40 PAINT(X,Y),C,B
50 IF RND(9)>.97 THEN S=S+1
51 IF S>1 THEN S=0
58 IF RND(9) > .02 THEN 70
60 BB=INT(RND(9)*16)
70 COLOR BB,S
99 GOTO 18
```

Assignment 22:

1. Draw your school's initials, and make them flash on and off with your school's colors.

2. Draw the flag of your country and color it correctly.



INSTRUCTOR NOTES 23 MUSIC

The PLAY statement reads notes and other instructions from a string and plays music over the computer's speaker. PLAY requires Advanced BASIC.

PLAY has two modes; MF (foreground mode) in which the execution of the program pauses while the music plays, and MB (background mode) in which the computer goes on executing BASIC statements. In background mode, it takes a few seconds before the computer stores all the music information and starts on the next program steps.

Only the notes that are white and black keys on a piano are accepted by the PLAY command. For example, B# is OK, but B flat is not. The octave symbol sets the octave (0 to 6) from which the notes are chosen. Middle C is the lowest note in octave 3.

The lengths of the notes are set by the symbol Ln in the string, where n is an integer. A table of values on n vs. traditional note names, such as "quarter note" is given. This method also allows for "triplets" where three notes are played in the same time as two are ordinarily.

The "dotted" notes in musical notation are also denoted here by a dot placed after the note's name.

Rests are signified by Pn, in exactly the same style as Ln.

The overall tempo is set (or changed) by TEMPO symbols in the string.

The music sounds a little mechanical, but can be improved by using staccato and legato. The staccato sounds as expected, but legato really comes out as a "tie." That is, two legato C quarter notes played together would actually sound like a half note.

After you have mastered music making as described in this lesson, read the information about PLAY in the BASIC manual because there are a few more minor tricks that you may find useful.

QUESTIONS:

1. Where does the PLAY statement get the notes it plays?

2. What does a "dot" after a note mean?

3. What does "b#" mean in a PLAY string?

4. What does "ML" mean in a PLAY string?

5. What is a "triplet" of quarter notes? How do you tell the computer to play them?

6. What is the difference between "background" and "foreground" for the PLAY statement?

LESSON 23 MUSIC

ROW, ROW, ROW YOUR BOAT

Enter:

```
10 REM row your boat
20 M$="t100 o3 L4 c c L6c L16d L4e L6e L16d L6e L16f L2g
   o4 L12 ccc o3 ggg eee ccc L8g f e d L2c"
40 PLAY M$

RUN.  SAVE to disk
```

The computer can play a whole tune with just one command, PLAY.

But first you have to put all the notes of the tune in a string where the PLAY statement can find them.

It can PLAY the tune while it is doing something else, like moving graphics in a game.

WHAT GOES IN THE PLAY STRING

You have to put a symbol in for each note and rest.

NOTES	a	b	c	d	e	f	g		(natural notes)
	a#		c#	d#		f#	g#		(sharp notes)
	a-♭	b-♭		d-♭	e-♭		g-♭		(flat notes)

REST	full note rest	P1	(P for "pause")
	half note rest	P2	
	"triplet half" rest	P3	
	quarter note rest	P4	
	"triplet quarter" rest	P6	
	eighth note rest	P8	
	"triplet eighth" rest	P12	
	sixteenth note rest	P16	
	"triplet sixteenth"	P24	

DOTS notes or rests can be followed by a "." which means hold it half again as long.

Example: "cdep1c#d#f.p1."
means "c d e p1 c#. d#. f. "

(It is easier to read if I spread out the symbols with spaces. You may omit the spaces in the program.)

Put in the following symbols just once and they hold the "notes" for the music from the time you put the symbols in, until you change by putting in a new symbol.

OCTAVE

o0 o1 o2 o3 o4 o5 o6 o7
octave o1 is the lowest
octave o3 starts with the note "middle C"
octave o7 is the highest

LENGTH

full note	L1
half note	L2
"triplet half"	L3
quarter	L4
"triplet quarter"	L6
eighth	L8
"triplet eighth"	L12
sixteenth	L16
"triplet sixteenth"	L24

TEMPO

T 32 to T 39	very slow	Larghissimo
T 40 to T 59	Largo	
T 60 to T 65		Larghetto
		Grave
		Lento
T 66 to T 75		Adagio
		Adagietto
T 76 to T107	slow	Andante
		Andantino
T108 to T119	medium	Moderato
		Allegretto
T120 to T167	fast	Allegro
		Vivace
		Veloce
T168 to T208		Presto
T209 to T255		Prestissimo

MUSIC STYLE

MF	Foreground:	The program stops while the music plays.
MB	Background:	The program continues to run while the music plays. (Not more than 32 notes and rests.)
MN	Normal	not legato or staccato
ML	Legato	ties each note to the next
MS	Staccato	notes clipped off

Examples:
```
"mf ms t100 l8 o4 c d e f g a "
"mf ml t150 l8 o3 c d e f g a "
```

EXTERNAL Xvar; Where "var" is a string variable name

Example:
```
10 M$="c d e d e f "
20 PLAY "ms xM$; ml xM$; t220 xM$; "
```

LEGATO AND STACCATO

Assignment 23:

1. Change the string in the "Row, Row, Row Your Boat" program so that it plays very fast, very slow, an octave higher, an octave lower, staccato, legato.

2. Change the first program in this lesson so it plays another tune.

3. Crossing friends. While music plays, make your name move down the screen while a friend's name moves along the screen. Use the MB symbol in the string. The names cross in the middle.

INSTRUCTOR NOTES 24 PRETTY PROGRAMS, GOSUB, RETURN

This lesson covers subroutines.

Like GOTO, GOSUB causes a jump to another line number. The only difference is that in GOSUB, control returns to the calling line after the subroutine is done executing. This is accomplished by storing the statement address following the GOSUB statement on a stack. When the computer encounters a RETURN statement, it pops the address off the stack and returns control to that statement.

Subroutine calls can be nested at least nine deep.

The END command can be put anywhere in the program and you can use as many END statements as you wish. All END does is return control to the edit mode.

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

GOSUB was put into BASIC for making modules. This lesson shows modular construction in a graphics program. The same subroutine that writes the letter "J" also erases it.

The JUMPING J exercise allows the student to try many different effects in the moving graphics display.

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 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

RUN this program then save it to disk or tape:

```
100 REM Take a trip
101 REM
110 PRINT "Hop to the subroutine"
120 GOSUB 200
130 PRINT "Back from the subroutine"
133 FOR T=1 TO 1000: NEXT T
134 PRINT
135 PRINT "Hop again"
140 GOSUB 200
150 PRINT "Home for good."
190 END
199 REM
200 REM subroutine
201 REM
210 PRINT "Got here ok."
215 FOR T=1 TO 1000: NEXT T
217 BEEP: PRINT "Pack your bags, back we go."
230 FOR T=1 TO 1000: NEXT T
290 RETURN
```

SAVE to disk

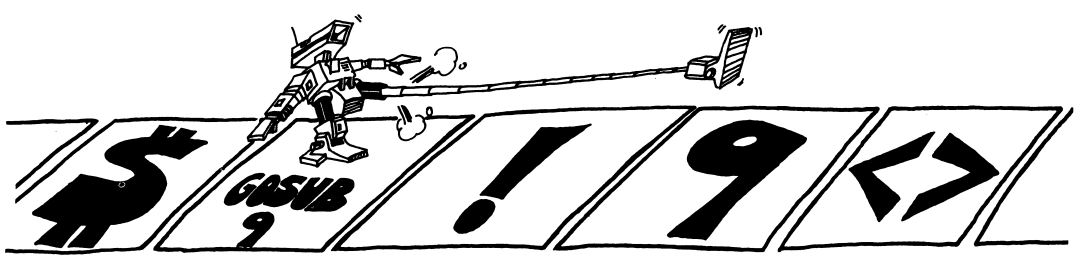
This is the skeleton of a long program. The main program starts at line 100 and ends at line 190.

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 line 140 "call the subroutine." This means the computer goes and does the commands in the subroutine, 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.

Assignment 24A:

1. The delay loop is written three times in the above program. Add another subroutine with a delay loop in it, and GOSUB every time you need a delay.

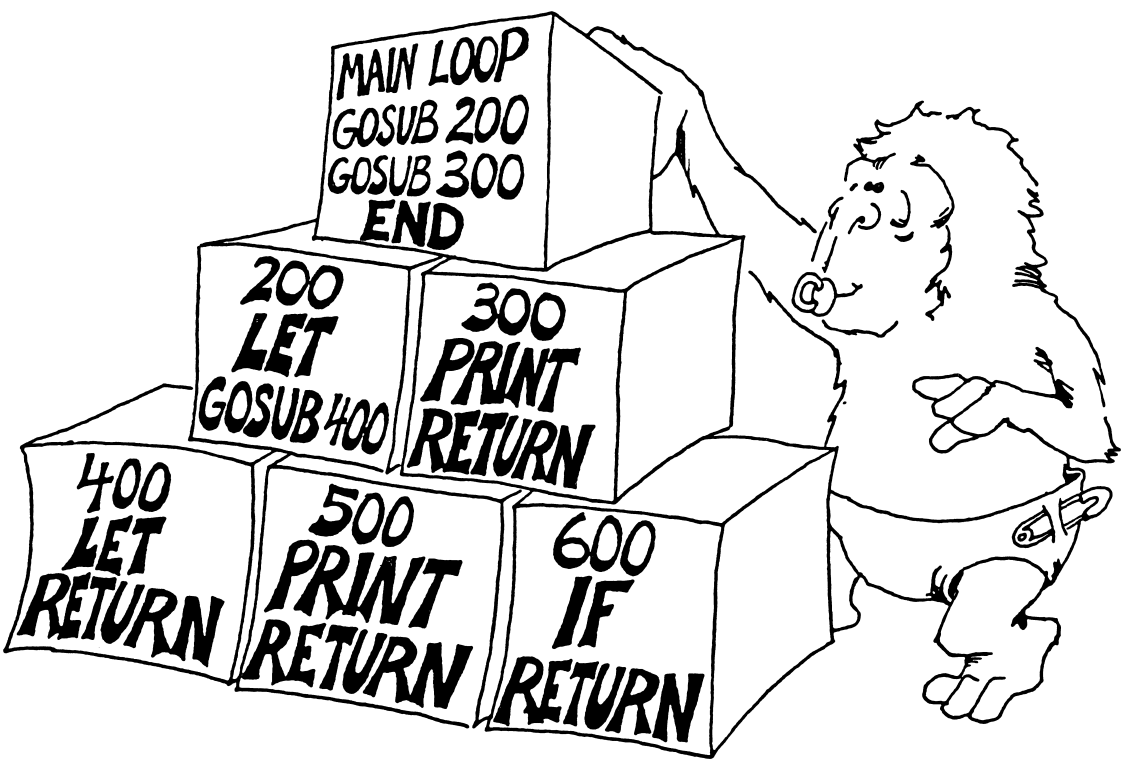
WHAT GOOD IS A SUBROUTINE?

In a short program, not much.

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

Run:

```
10 REM ??? Jumping J ???
20 CLS: SCREEN 0,1          ' text screen, enable color
22 X=13: Y=15: D=1
25 FOR J=1 TO 10
26 FOR I=1 TO 10
30 COLOR 2,0: GOSUB 110 ' draw
31 FOR T=1 TO 99: NEXT T
35 COLOR 0,0: GOSUB 110 ' erase
45 Y=Y-D
50 NEXT I
55 D=-D
60 NEXT J
90 COLOR 0,7
95 CLS
99 END
100 '
101 ' draw the J
102 '
110 LOCATE Y,X: PRINT "*****"
119 FOR K=1 TO 7
120 LOCATE Y+K,X+3: PRINT "*"
121 NEXT K
130 LOCATE Y+7,X: PRINT "***"
140 LOCATE Y+6,X: PRINT "*"
199 RETURN

SAVE to disk
```

The picture is the letter J. The subroutine starting in line 100 draws the J. Before you GOSUB 110, you determine 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 0, then the subroutine erases a J from that spot because the background is black.

The subroutine draws the J with its upper left corner at the spot X,Y on the screen. When you change X or Y (or both), the J will be drawn in a different spot. Line 22 says that the first J will be drawn near the middle of the screen.

The letter D tells how far the J will move from one drawing to the next. Line 22 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 24B:

1. Write a short program that uses subroutines. It doesn't have to do anything useful, just print some silly things. In it put three subroutines:

   Call one of them twice from the main program.
   Call one of them from another of the subroutines.
   Call one of them from an IF statement.

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

   Change the subroutine so 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 15 steps instead of 10.

   Change the size of each step from 1 to 2 (while total distance moved is 10 squares).

   Change the sliding so it slides uphill. Use

   ```
   X=X+D:Y=Y-D
   ```

   Change the program so the initial changes color from blue (color 1) through all the colors to brown (color 6) as it jumps.

3. Make a program that writes your three initials on the screen, each one in a different color. Then make them jump up and down one at a time!

INSTRUCTOR NOTES 25 ASCII CODE, ON...GOTO

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

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

Strictly speaking, there are only 128 ASCII characters, and some of these are supposed to signal mechanical actions on a teletype machine. It is more convenient to define a full byte's worth (256) of characters and each computer manufacturer uses the extra characters in a unique way. IBM assigns them to various graphics, math, and foreign language letters.

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).

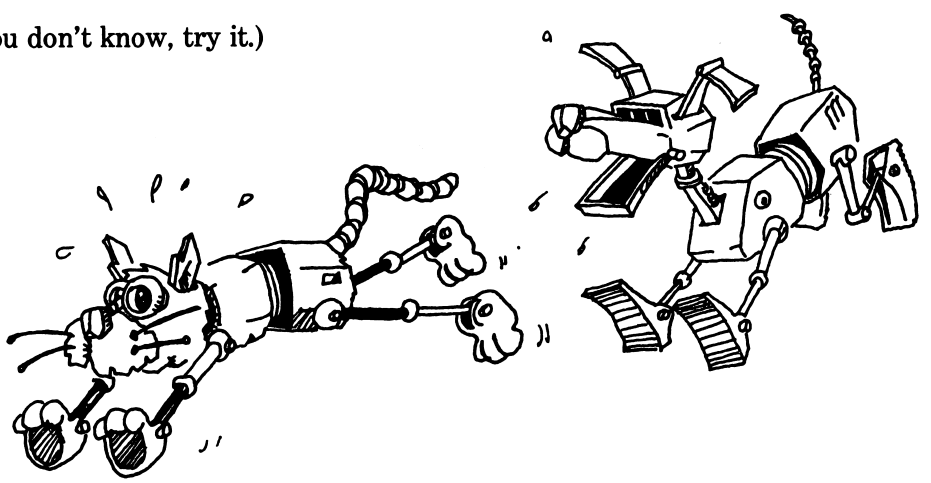
4. Which letter has the larger ASCII code number, B or W?

5. Do you know the ASCII code for the character "1"? Is it the number 1?

6. What will the computer do if you run this line:

```
10 PRINT CHR$(32); CHR$(65)
```

(If you don't know, try it.)



LESSON 25 ASCII CODE, ON . . . GOTO

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.

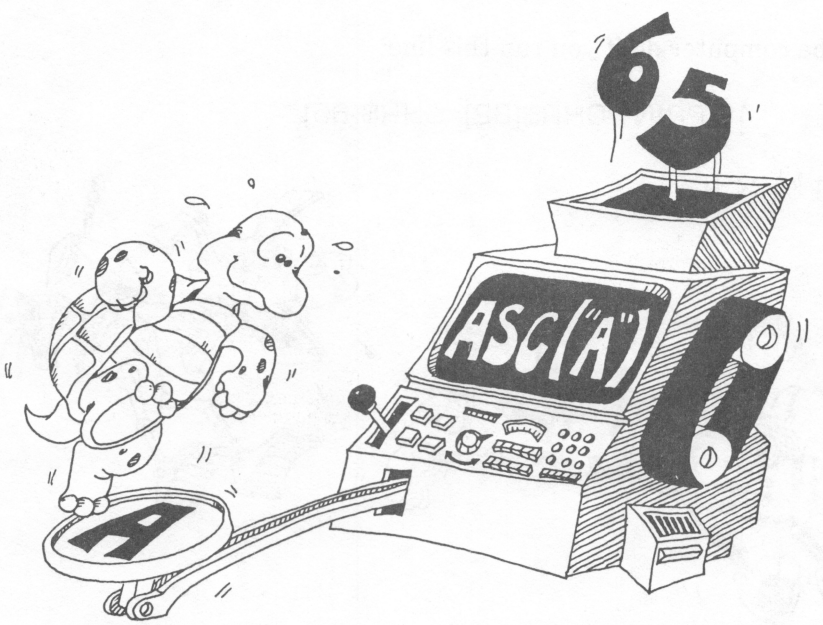
ASC( ) CHANGES CHARCTERS INTO NUMBERS

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

RUN:
```
10 REM *** What number is this key? ***
20 PRINT "Press keys to see ASCII number"
30 INPUT C$
40 PRINT C$;TAB(5);ASC(C$)
50 GOTO 30
```

Try out some letters, digits, and punctuation marks. Hold down the shift key and press some letters.

Press BREAK to end the program. Then SAVE it to disk.



CHR$( ) CHANGES NUMBERS INTO CHARACTERS

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

RUN:
```
10 REM /// DISPLAY ASCII ///
11 REM
20 CLS
30 FOR I=0 TO 255
40 PRINT I, CHR$(I)
45 PRINT
50 FOR T=1 TO 200: NEXT T
60 NEXT I
```

SAVE the program to disk.

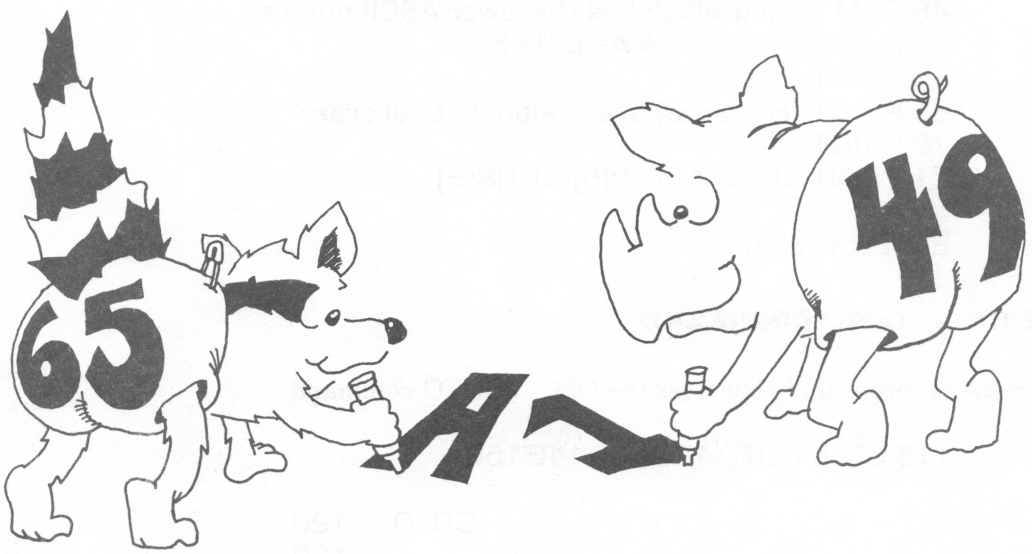
Use CHR$( ) to print the many ASCII characters which are not on the keyboard. They range from graphics like the "smiley face" and borders and blocks, to foreign letters like Greek and Japanese.

CHR$( ) IS THE REVERSE OF ASC( )

We showed 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.



THE ASCII NUMBERS FOR CHARACTERS

Here are the groups of characters and their ASCII numbers:

1 to 31	symbols
32 to 47	punctuation
48 to 57	number digits
58 to 64	punctuation
65 to 90	capital letters
91 to 96	punctuation
97 to 122	small letters
123 to 127	punctuation
128 to 168	foreign letters
169 to 223	graphics
224 to 238	greek letters
239 to 255	math symbols

ALPHABETICAL LIST

They can also help in making alphabetical lists.

RUN:

```
10 REM Alphabetize
20 PRINT
30 INPUT "Give me a letter: ",A$
35 PRINT
40 INPUT "Give me another: ",B$
45 A=ASC(A$)
46 B=ASC(B$)
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
60 PRINT
65 PRINT "Here they are in alphabetical order"
70 PRINT
71 PRINT CHR$(A);TAB(5);CHR$(B)
```

SAVE it to disk.

THE ON . . . GOTO COMMAND

The SNAKE program below uses the ON . . . GOTO command.

```
115 ON K GOTO 120,130,140,150
```

if	K	is	1	GOTO	120
			2		130
			3		140
			4		150
if	K	is	something		go on to
			else		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.

```
  10 REM > > > > > SNAKE > > > > >
  11 GOTO 1000
 100 REM
 101 REM ------------main loop
 102 REM
 105 D$=INKEY$
 110 IF D$="," THEN K=K+1:IF K=5 THEN K=1
 111 IF D$="." THEN K=K-1:IF K=0 THEN K=4
 115 ON K GOTO 120,130,140,150
 120 Y=Y-1:GOTO 160
 130 X=X-1: GOTO 160
 140 Y=Y+1:GOTO 160
 150 X=X+1
 160 COLOR 7,0:LOCATE Y,X:PRINT CHR$(219);
 161 COLOR 1,0:LOCATE L,A:PRINT " ";
 170 A=B:B=C:C=D:D=E:E=F:F=X
 171 L=M:M=N:N=O:O=P:P=Q:Q=Y
 199 GOTO 105
1000 REM
1001 REM > > > > > s n a k e > > > > >
1002 REM
1011 COLOR 7,0
2000 X=40:Y=12:K=1
2010 A=X:B=X:C=X:D=X:E=X:F=X
2011 L=Y:M=Y:N=Y:O=Y:P=Y:Q=Y
2050 PRINT:PRINT:PRINT
2100 PRINT" Use < and > keys"
2200 FOR T=1 TO 1000:NEXT T
3200 CLS
3999 GOTO 100
```

Assignment 25:

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 26 SNIPPING STRINGS: LEFT$, MID$, RIGHT$, LEN

In this lesson the functions:

LEFT$ MID$ RIGHT$ LEN

are demonstrated. The use of MID$( ) with three arguments is shown, but not that with the third argument omitted.

These functions, together with the concatenation operation "+", allow complete freedom to cut up strings and glue them back in any order.

As in earlier explanations, the main characteristics of the functions are shown, but not all the special cases, especially those that lead to error messages. It is better that extensive explanations not clutter up the text. If the student experiences difficulty, an experienced programmer or an adult consulting the computer reference manuals should clear up the problem.

QUESTIONS:

1. If you want to save the STAR from STARS AND STRIPES, what function will you use? What arguments?

2. If you want to save AND, what function and arguments?

3. If you want to count the number of characters in the string PQ$, what function do you use? What argument?

4. What is wrong with each of these lines?

```
10 A$=LEFT$(4,D$)
10 RIGHT$(R$,I)
10 F$=MID$(A,3)
10 J$=LEFT(R$,YT)
```

5. What two arguments does the RIGHT$( ) function need?

6. What command will snip the third and fourth letters out of a word.

7. Write a short program that takes the word computer and makes it into putercom.

LESSON 26 SNIPPING STRINGS: LEFT$, MID$, RIGHT$, LEN

GLUING STRINGS

You already know how to glue strings together:

```
55 A$="con" + "cat" + "en" + "ation"
60 PRINT A$
```

The real name for "gluing" is "concatenation."

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

SNIPPING STRINGS

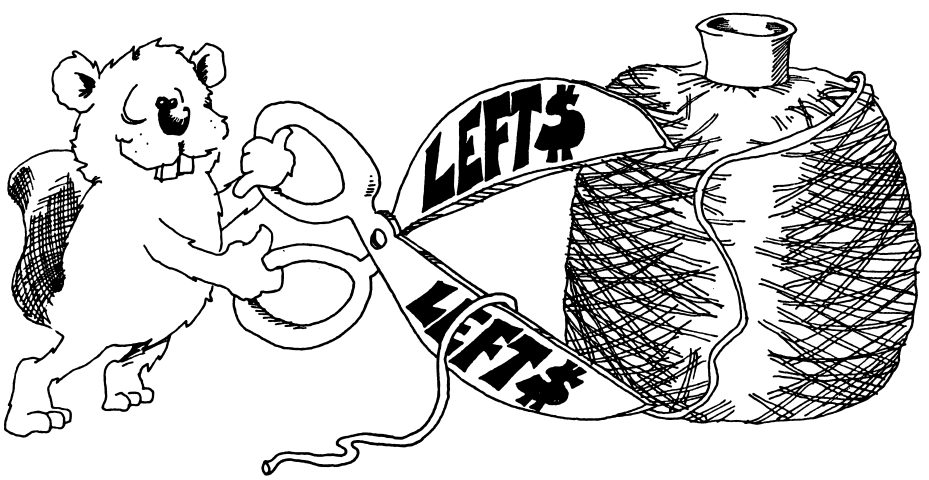
Let's cut a piece off a string. Enter and RUN:

```
10 REM > > > Scissors > > >
20 CLS
30 N$="123456789"
35 Q$=LEFT$(N$,4)
40 PRINT Q$, N$
```

The LEFT$ function snips off the left end of the string. The snipped off piece can be put in a box or printed or whatever.

RULE: The LEFT$( ) function needs two things inside the ( ) signs.

1. The string you want to snip.
2. The number of characters you want to keep.



Try another. Change line 40 to:

```
40 PRINT RIGHT$(N$,3)
```

and RUN the program again. This time the computer prints:

```
789
```

RIGHT$( ) is like LEFT$( ) except the characters are saved off the right end of the string. (They are still saved left to right.)

MORE SNIPPING AND GLUING

RUN:

```
10 REM ::: Scissors and Glue :::
20 CLS
30 N$="123456789"
35 FOR I=1 TO 9
40 L$=LEFT$(N$,I): R$=RIGHT$(N$,I)
50 PRINT I;TAB(5);L$;TAB(15);R$
60 NEXT I
```

The pieces of string you snip off can be glued back together in a different order. Add this line and run:

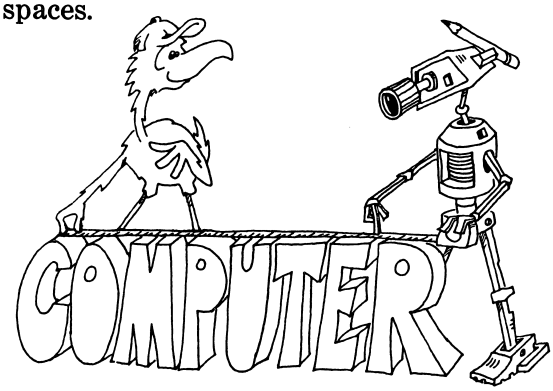
```
55 IF I=4 THEN PRINT: PRINT R$ + L$: PRINT
```

HOW LONG IS THE STRING?

RUN:

```
10 REM ::: Long rope :::
20 CLS
30 INPUT "Give me a string: ";N$
40 L=LEN(N$)
50 PRINT "The string: '";N$;"'"
55 PRINT: PRINT "is";L;"characters long"
```

The function LEN( ) tells the number of characters in the string; it counts everything in the string, even the spaces.



CUTTING A PIECE OUT OF THE MIDDLE

The MID$( ) function cuts a piece out of the middle of the string.

RUN:
```
10 REM ### Middle ###
20 CLS
30 N$="123456789"
40 P$=MID$(N$,3,4)
50 PRINT P$
```

The line: 40 P$= MID$(N$,3,4) means:

Get the string from box N$.
Count over three letters and start saving letters into box P$.
Save four letters.

LOOK MA, NO SPACES

Enter:
```
10 REM > > > no spaces > > >
11 REM
20 PRINT: PRINT
30 PRINT "Give me a long sentence": PRINT
35 INPUT S$
40 L=LEN(S$)
45 T$=""
50 FOR I=1 TO L: REM look at each letter
60 L$=MID$(S$,I,1) 70 IF L$< >" " THEN T$=T$+L$: REM
   don't save spaces
90 NEXT I
92 PRINT: PRINT T$
95 PRINT: PRINT: PRINT
```

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

ALPHABETICAL ORDER

Enter:

```
20 IF "A" < "B" THEN PRINT "A comes before B"
22 If "ca"<"cz" THEN PRINT "CA comes before CZ"
```

RUN each one and then change the < to a > and RUN again.
The "less than" and "greater than" signs can be used to see if two trings are in alphabetical order.

Assignment 26:

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

   THIS IS A DRAGON

   becomes:

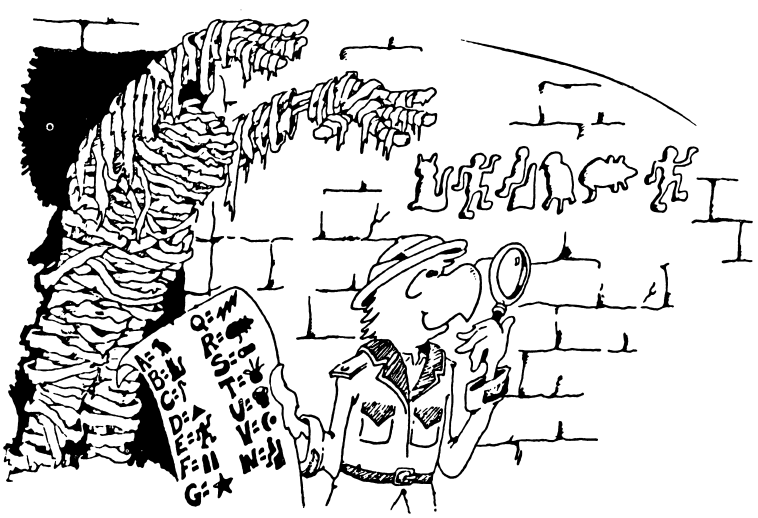
   HTSII S ARDGANO

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 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 AY. Examples:

BOX becomes OXBAY
APPLE becomes APPLEAY



INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS

This lesson treats two functions, STR$ and VAL. A general review of the concept of function is also made.

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

VAL does just the opposite, taking a string and making a numeric value from it. It accepts decimals and "scientific" notation (e.g.,"1.2E+13"). If the first character is not a decimal digit, or + or –, it returns the value Ø. Otherwise, it scans the number, terminating at the first non-numeric character (other than the E of the scientific notation).

This interconversion of the two main types of variables adds great flexibility to programs involving numbers.

You can slice up a number and rearrange its digits by first converting it to a string. This is demonstrated in the assignment that makes a number play "leap frog" by repeatedly putting its rear digit in the front.

Functions "return a value" to the expression they are in. One also says that functions are "called" just as one "calls" a subroutine. The reason is, of course, that functions are implemented as subroutines on the machine code level.

QUESTIONS:

1. If your number "marches" too quickly in the program from assignment 27, how do you slow it down?

2. If your program has the string 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? How many arguments does the RIGHT$( ) function have? How many for the CHR$( ) function?

5. Can you put a function at the start of a line?

6. Each line below has errors. Explain what is wrong.

```
10 INT(Q)=65
10 D$=LEFT(R$,1)
10 PW$=VAL(F$)
10 PRINT CHR$
```

LESSON 27 SWITCHING NUMBERS WITH STRINGS

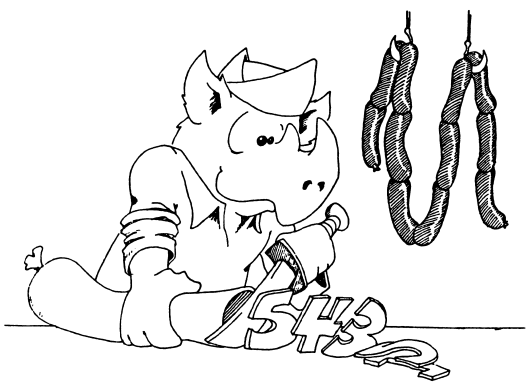
This lesson explains two functions: VAL( ) and STR$( ).

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
20 CLS
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
```

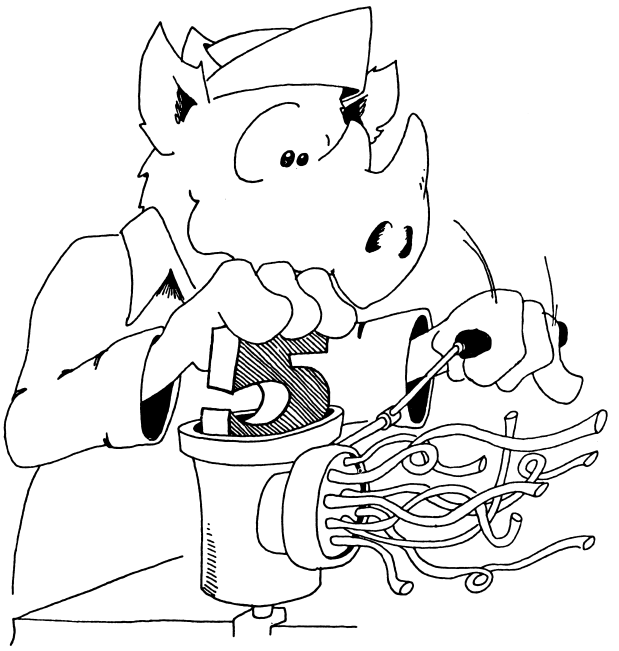
VAL stands for "value." It changes the string into a number, if it can.



MAKING NUMBERS INTO STRINGS

RUN:
```
10 REM Making numbers into strings
11 REM
20 PRINT
25 INPUT "Give me a number ",NB
30 N$=STR$(NB)
35 L=LEN(N$)
37 PRINT
40 FOR I=L TO 1 STEP -1
45 B$=B$+MID$(N$,I,1)
50 NEXT I
60 PRINT "Here it is backwards"
65 PRINT:PRINT B$
```

STR$ stands for "string." It changes a number into a string.



FUNCTIONS AGAIN

In this book we use these functions:

RND( )	INT( )		
LEFT$( )	RIGHT$( )	MID$( )	LEN( )
VAL( )	STR$( )		
ASC( )	CHR$( )		

RULES about functions:

Functions always have ( ) with one or more "arguments" in them. Example:

MID$(D$,5,J) has three arguments: D$, 5, and J.

The arguments may be numbers or strings or both.

A "function" is not a "statement." It cannot begin a line.

right: 10 LET D=LEN$(CS$)

wrong: 10 LEN (CS$)=5

A function acts just like a number or a string. We say the function "returns a value." The value can be put in a box or printed just like any other number or string. The function may even be an argument in another function.

The arguments tell which value is returned.

(Remember, string values go in string variable boxes, numeric values go in numeric boxes.)

PRACTICE WITH FUNCTIONS

For each function in the list below:

Is the value of the function a "string" or a "number"?

How many arguments does it have? What are their names?

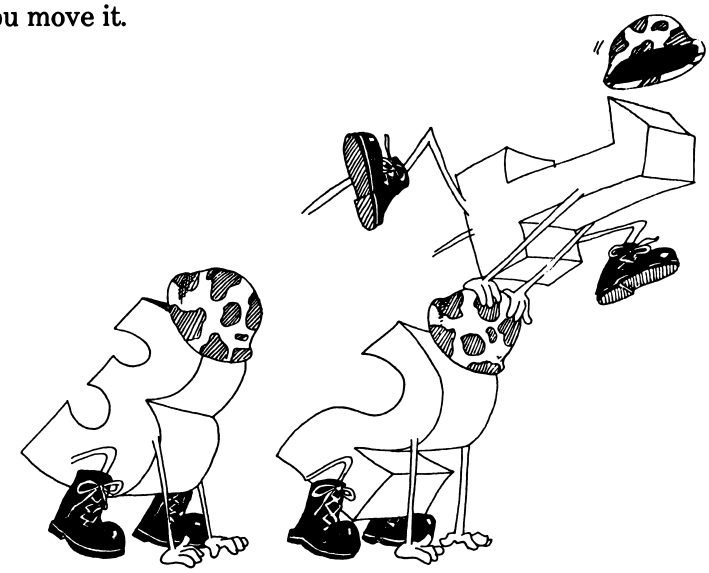
For each argument: Is it a variable, constant or a function?
Is it "string" or "numerical"?

RND(9) fn _____
arg _____
INT(Q) fn _____
arg _____
MID$(RI$,E,2) fn _____
arg _____
arg _____
arg _____
VAL(ER$) fn _____
arg _____
STR$(INT(RND(8))) fn _____
arg _____

Assignment 27:

1. Write a program that asks for a number. Then make another number that is backward from the first, and add them together. Print all three numbers like an addition problem (with "+" sign and a line under the numbers).

2. Make a number "leapfrog" slowly across the screen. First write it on the screen. Then take its left digit and move it to the front. Keep repeating. Don't forget to erase each digit when you move it.



INSTRUCTOR NOTES 28 LOGIC: AND, OR, NOT

This lesson treats the AND, OR and NOT relations and the numeric values for TRUE and FALSE.

There are two abstract ideas in this lesson which may give you some trouble. One is that TRUE and FALSE have numeric values of -1 and 0. Any expression that is in the form of an assertion (a "phrase A"), has a numeric value of 0 or -1. This number (the 0 or -1) is treated like any other number. It can be stored in a numeric variable, printed, or used in an expression. Most often it is used in an IF statement.

The other abstract idea compounds the confusion. The IF command doesn't really look to see if "phrase A" is present. Rather, it looks for a numeric value between IF and THEN. Any number that is non-zero is treated as TRUE. We call this a "little white lie."

You can use the logical values in equations that look ridiculous at first glance. For example:

```
10 INPUT A
20 B = 5 - 7*(A<3)
30 PRINT B
```

The value of B will be 12 or 5, depending on whether or not A is less than 3.

QUESTIONS:

1. For each IF statement, tell if it will print anything:

```
10 IF 3=3                THEN PRINT "hi"
10 IF 3=3 OR 0=2         THEN PRINT "hi"
10 IF NOT (3=3)          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 numbers 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 28 LOGIC: AND, OR, NOT

ANOTHER TEENAGER PROGRAM

Enter:

```
10 REM < < < AND, OR, NOT > > >
20 CLS: PRINT
30 INPUT "Your first name ";N$
35 PRINT
40 INPUT "Your age ";A
45 PRINT
50 TA=(A>12 AND A<20)
55 PRINT " ";N$;
60 IF TA THEN PRINT " is a teenager"
65 IF NOT TA THEN PRINT " is not a teenager"
70 IF A=16 THEN PRINT " and is sweet sixteen"
80 IF A=12 OR A=20 THEN PRINT " and just missed"
```

RUN and SAVE to disk.

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 lines 50 and 60.

IF (you are over 12) AND (you are less than 20) THEN (you are a teenager).

WHAT DOES "OR" MEAN?

In line 80 the OR is used. Two things are said: "Age is 12" and "age is 20."

Only one of them needs to be true for you to have "just missed" being a teenager.

IF (you are 12) OR (you are 20) THEN (you just missed being 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 Ø, meaning FALSE.

and this:

```
PRINT 3=3
```

The computer checks to see if 3=3. It is, so the computer prints "-1" meaning "TRUE."

PUTTING TRUE AND FALSE IN BOXES

The numbers for TRUE and FALSE are treated just like other numbers, and can be stored in boxes with numeric variable names on the front. RUN this:

```
10 N = (3=22)
20 PRINT N
```

The number 0 is stored in the box N because 3=22 is FALSE.

And this:

```
10 N = "B"="B"
20 PRINT N
```

The number -1 is stored in the box N because the two letters in the quotes are the same, so the statement "B"="B" is TRUE.

Whole strings are tested for equality:

RUN: 10 PRINT "ab"="ac"

The computer prints 0 for FALSE, because the second letter of each string is not the same.

THE IF COMMAND TELLS LITTLE WHITE LIES

The IF command looks like this:

10 IF (phrase 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"

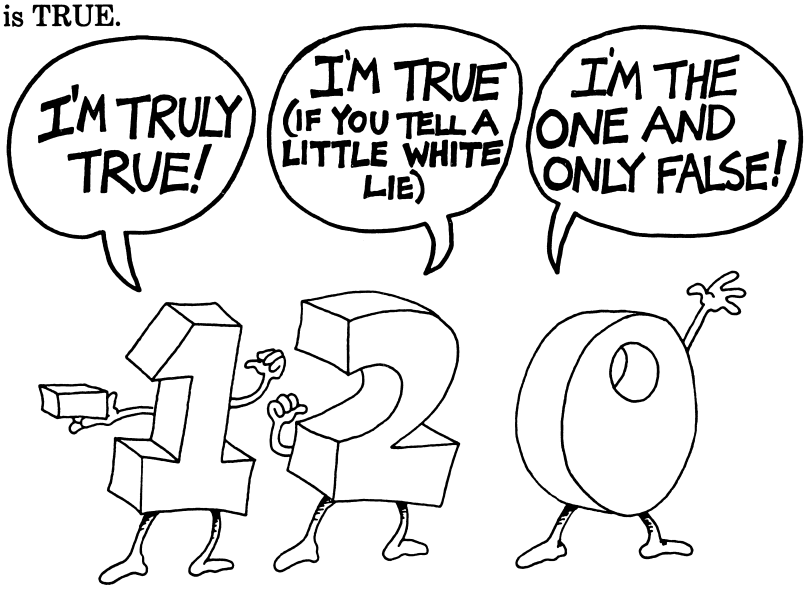
What does it print? _____

RULE: In an IF, the computer looks at "phrase 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 streches the truth to say "TRUE is anything that is not FALSE," that is, any number that is not zero is TRUE.



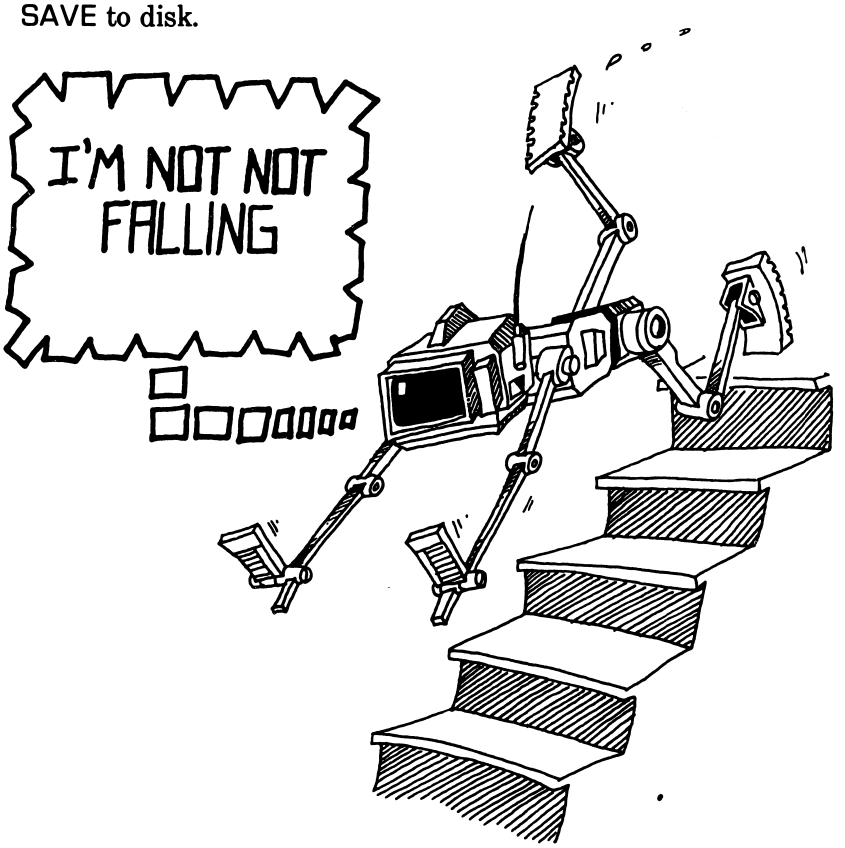
WHAT DOES "NOT" MEAN?

NOT changes FALSE to TRUE, and TRUE to FALSE. Try this:

```
10 REM ??? Double Negative ???
20 N=0
30 PRINT "N ";              TAB(15); N
40 PRINT "NOT N";           TAB(15); NOT N
50 PRINT "NOT NOT N ";TAB(15); NOT (NOT N)
60 REM The computer knows that "I don't have no . . . "
61 REM means "I do have . . . . "
```

Be sure to put a space after each NOT.

SAVE to disk.



The NOT makes sense only when used with 0 or -1. Try this. Change line 20 to:

```
20 N=-1
```

It still makes sense. But try:

```
20 N=3
```

And you do not get TRUE or FALSE printed as numbers. (You do not get -1 or 0.)

THE LOGICAL SIGNS

You can use these six symbols in "phrase A":

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

You have to press two keys to make the <> sign 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 "phrase A" phrases mean the same:

```
2<=Q        (2<Q) OR (2=Q)
```

Assignment 28:

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 Ø                THEN PRINT "JELLYBEAN"
IF 1                THEN PRINT "JELLYBEAN"
IF 9                THEN PRINT "JELLYBEAN"
IF 3< >Ø            THEN PRINT "JELLYBEAN"
IF 2 AND 4          THEN PRINT "JELLYBEAN"
IF Ø 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 (Ø) OR Ø     THEN PRINT "JELLYBEAN"
IF 4<=5             THEN PRINT "JELLYBEAN"
```

3. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing and count them. If there are 2 such words there is a double negative. Test the program on the sentence COMPUTERS AIN'T GOT NO BRAINS.

INSTRUCTOR NOTES 29 SECRET WRITING AND INKEY$

INKEY$ is a method of requesting a single character from the keyboard and putting it into the box of a specified string variable.

There is no screen display at all. No prompt or cursor is displayed and the keystroke is not echoed to the screen.

The utility of the INKEY$ command lies in the fact that everything it does is unseen. For example, a secret password may be received with a series of INKEY$s without displaying it to bystanders.

Another advantage over INPUT is that no Enter key pressing is required. This makes INKEY$ useful in "user friendly" programming.

The INKEY$ statement doesn't wait for a key to be pressed. This makes it useful in action games. If you need to have the program wait for a keystroke, you must do an IF and branch back until a keystroke is detected. This is demonstrated in the lesson.

If you want to get numeric values, get them as strings and convert them to numbers using the VAL( ) function discussed in a later lesson.

QUESTIONS:

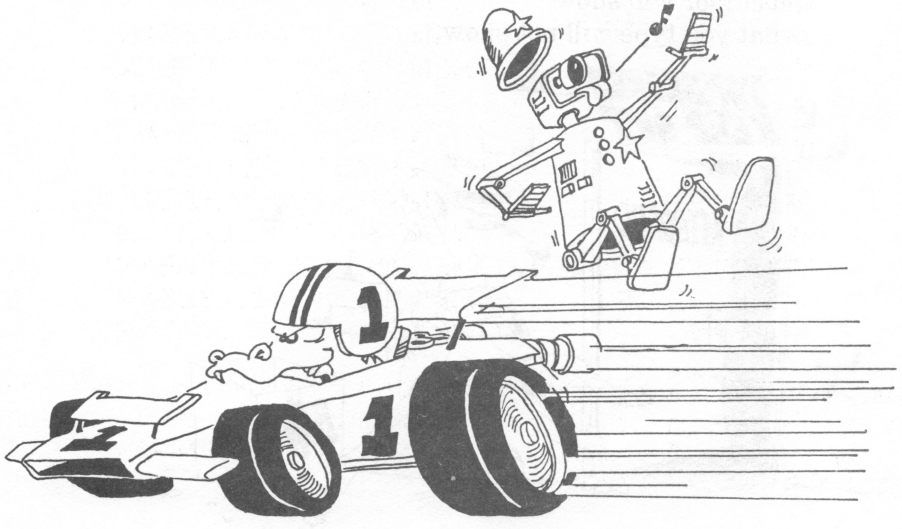
1. Compare INPUT and INKEY$. For each item below, which does which?

   One gets one letter at a time, the other gets whole words and sentences.

   One has a cursor, the other does not.

   One prints on the screen, the other does not.

   One needs the Enter key, the other does not.



LESSON 29 SECRET WRITING AND INKEY$

THE INPUT STATEMENT

There are two ways to use INPUT:

Without a message:

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

With a message:

```
10 INPUT "Name, age ";NA$,AG
```

Either way, the computer waits for you to type a word, sentence or number.

Then you press the Enter key to tell the computer that you are done entering.

THE INKEY$ STATEMENT

The INKEY$ statement is different from INPUT. It gets a single character from the keyboard.

It doesn't wait.

It looks to see if a key is being pressed. If so, it puts the character into the string variable box.

You do not have to press Enter.

INKEY$ FOR INVISIBLE TYPING

Nothing shows on the screen:

No question mark will show
no cursor will show
what you type will not show.

To see what happens, you have to PRINT the variable.

RUN:
```
10 K$=INKEY$
20 PRINT K$
25 REM box K$ holds the character
30 GOTO 10
```

Ctrl Break to end the RUN.

The computer prints a blank until you press a key. Then it prints the character.

Try this: Hold down the "a" key

See that the computer prints a which is run up the screen. Then the a starts to repeat and you see a string of letters up the screen.

Try holding down different keys.

Hold down two keys at once. INKEY$ will display the last one you press.

MAKING THE COMPUTER WAIT FOR YOU TO TYPE

Add to the above program:

```
15 IF K$="" THEN 10
20 PRINT K;TAB(7); CHR$(K)
```

Now the computer is more polite! It keeps looking until a key is pressed.

SECRET WRITING

Use INKEY$ in guessing games. You can enter the word or number to be guessed without the other player being able to see it.

RUN this program:

```
10 REM --- secret ---
20 CLS
30 PRINT "Press any key"
40 K$=INKEY$:IF K$ = "" THEN 40
45 BEEP
47 FOR T=1 TO 1000: NEXT T
50 PRINT "The key you pressed was ";CHR$(K)
99 GOTO 40
```

RUN this one too:

```
10 REM ### Backwards ###
20 CLS
30 PRINT "Type in a 5 letter word"
35 PRINT
40 FOR I = 1 TO 5
```

```
42 L$=INKEY$:IF L = "" THEN 42
44 W$=L$ + W$
48 NEXT I
50 PRINT "Now here it is backwards"
55 PRINT
60 PRINT W$
```

Line 42 will not let the program continue until you press a key.

MAKING WORDS OUT OF LETTERS

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

```
10 REM Get a word
20 CLS
30 PRINT "Type a word. End it with an 'Enter'."
35 W$=""
40 L$=INKEY$:IF L$ = "" THEN 40
50 IF ASC(L$) = 13 THEN 80
60 W$=W$ + L$
65 GOTO 40
80 REM word is finished
85 PRINT W$
```

How does the computer know when the word is all typed in? Line 50 checks to see if the Enter key was pressed. The ASCII number of the Enter key is 13. Line 50 branches to print the word if the Enter key was pressed.

SECRET NUMBERS

If you want to enter a secret number from the keyboard, you have to use INKEY$ to enter digit characters (0 to 9), glue them into a string, and then use the VAL function explained in Lesson 27.

Assignment 29:

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 PICK 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."). Use INKEY$ so no player can see the words of the others. You may expand the game by having adjectives before the nouns.

INSTRUCTOR NOTES 30 LONG PROGRAMS

This lesson demonstrates top down organization of a task.

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.

Here we outline the program right on the screen. The task is "chunked" into sections by using subroutines. This leads to clarity in the articulation of the program parts and allows testing and debugging of each part separately from the others.

After the outline is done, each subroutine is expanded by writing in ordinary English what needs be done. Only when the English description is itself sufficiently detailed, does the BASIC programming begin.

Of course, there is always some backing and filling to be done as the program is written. The number of subroutines may change and the tasks performed in each will also change, usually expand.

There are those who advocate performing all planning of the program on paper before starting to code. This may work for some programmers, but children especially are unlikely to adopt this style of work. Besides, if one advocates word processors so that writing text can be done interactively on the screen, it would seem equally appropriate to plan computer programs on the screen.

QUESTIONS:

1. Why is it good to outline the program on the screen?

2. If you have trouble deciding what steps go in a game program, how can you, a friend and a piece of paper help?

3. What is the next thing you do (in English) to the outline?

4. When do you test each subroutine that you have written?

LESSON 30 LONG PROGRAMS

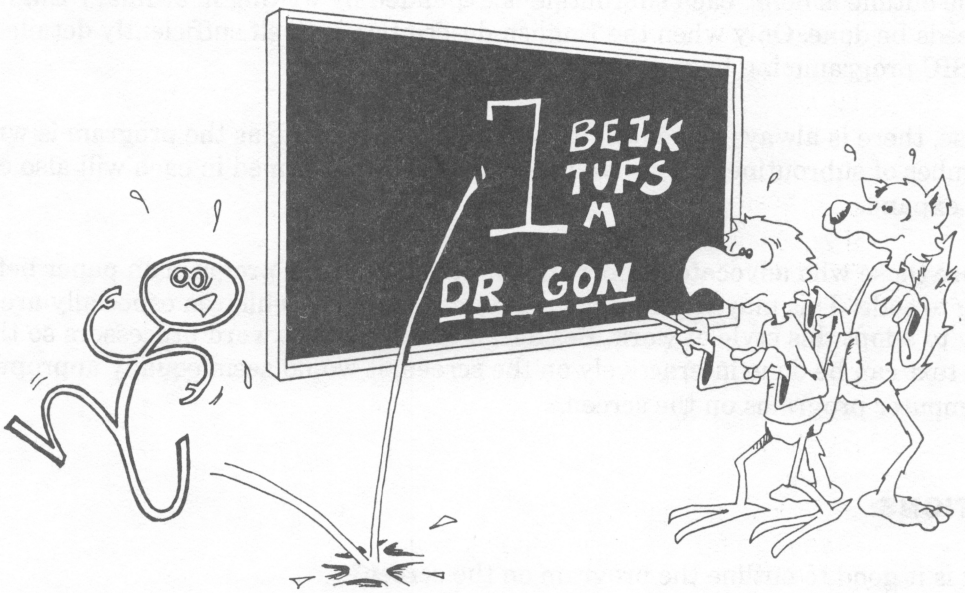
HOW TO WRITE A LONG PROGRAM

Let's write a Hangman game. This is a word guessing game where you draw another part of the hanging person each time you guess a letter that isn't in the word.

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 progam has to do the same things.

The outline 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, fill in more details.

```
  10 REM *** Hangman Game ***
  99 REM
 100 REM Main Loop
 101 REM
 120 INPUT" Need Instructions? <Y/N> ", 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 of 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
```

SAVE to disk.

Now is the time to start writing and testing the first part of the program. Put a STOP in line 132 so that only the first subroutine will be run. (After the first subroutine works correctly, take the STOP out. See Lesson 33.)

Start by writing the subroutine at 300, GET A WORD. The first step is to write more details, in English, of what the subroutine needs to do. Then start writing the BASIC lines.

VARIABLE NAMES

Variable names in IBM BASIC can be up to 40 characters long.

The name must start with a letter and can have letters, numbers, or periods in it.

The name must not be one of the reserved words like FOR, TO, IF, STOP, used as commands in BASIC. There is a list of reserved words in the appendix of this book.

But the name can have a reserved word inside it. Example:

FOR	cannot be a variable name
FORMULA	is OK
FOR.ME.TOO	is OK

Assignment 30:

1. Finish the Hangman game . This is a long project. Start by writing the GET A WORD subroutine. Then SAVE it to disk. You may want to write one subroutine each day until the program is done.

INSTRUCTOR NOTES 31 ARRAYS AND THE DIM STATEMENT

This lesson introduces arrays. The DIM( ) statement is 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 a member.

Two dimensional arrays are compared to the numbers on a calendar month page or the rectangular array of cells on the TV screen.

Arrays themselves are not too difficult a concept. The trick is to see how they help in programming. There are a large variety of uses for arrays, and many do not seem to fall into recognizable categories.

One can use them to store lists of information. Connected lists can also occur. The telephone number program uses two linear arrays: one for names, the other for numbers. They are indexed the same, so a single index number can retrive both the name and the number that goes with it.

Another general use of arrays is to store numbers which cannot neatly be obtained from an equation. An example would be the length in days of the 12 months.

Games often use arrays to store information about the playing board.

QUESTIONS:

1. What does the DIM BD(6) statement do?

2. Where do you put the DIM statement in the program?

3. What two kinds of array families are there?

4. What is the "index" or "subscript" of an array?

LESSON 31 ARRAYS AND THE DIM STATEMENT

MEET THE ARRAY FAMILY

```
22 F$(Ø)="dad"
24 F$(1)="mom"
26 F$(2)="Karen"
```

Each member of the family is a variable. The F$ family are string variables.

Here is a family of numerical variables:

```
35 N(Ø)= 43
37 N(1)= 13
39 N(2)= Ø
41 N(3)= Ø
```

The family has a "last name" like A( ) or B$( ). Each member has a number in ( ) for a "first name." The array always starts with the first name Ø.

Instead of "family" we should say "array."

Instead of "first name" we should say "index number" or "subscript."

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. 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(Ø), A(1), A(2) and A(3). These boxes are for numbers and contain the number Ø to start with. Another example:

```
30 DIM A(3),B$(4)
```

This time, DIM reserves four boxes for the A( ) array and five for the string array B$( ). The boxes named B$(Ø) through B$(4) are for strings and are empty to start with.

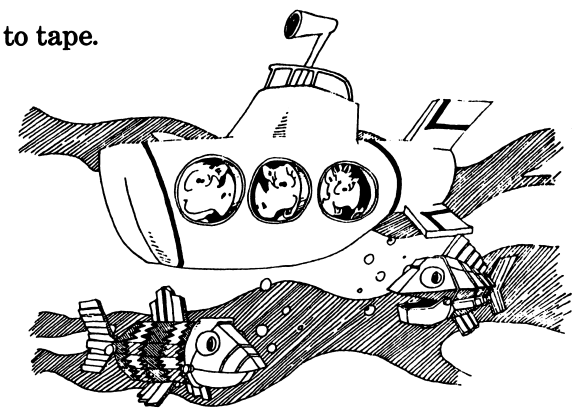
RULE: Put the DIM( ) statement early in the program, before the array is used in any other statement.

MAKING A LIST

Enter:

```
10 REM +++ In a row +++
20 cls: PRINT
30 DIM A$(5)
35 PRINT"Enter a word"
40 FOR N=Ø TO 5
45 IF N>Ø THEN PRINT"Another"
50 INPUT A$(N)
55 PRINT
60 NEXT N
70 PRINT
100 REM Put in a row
105 PRINT "Here they are in a row"
106 PRINT
110 FOR I=Ø TO 5
120 PRINT A$(I);" ";
130 NEXT I
```

RUN and SAVE to tape.



You can use a member of the array by itself, look at this line:

```
40 B$(2)="yellow submarine"
```

Or the array can be used in a loop where the index keeps changing. Lines 50 and 120 in the program "in a row" do this.

MAKING TWO LISTS

Enter:

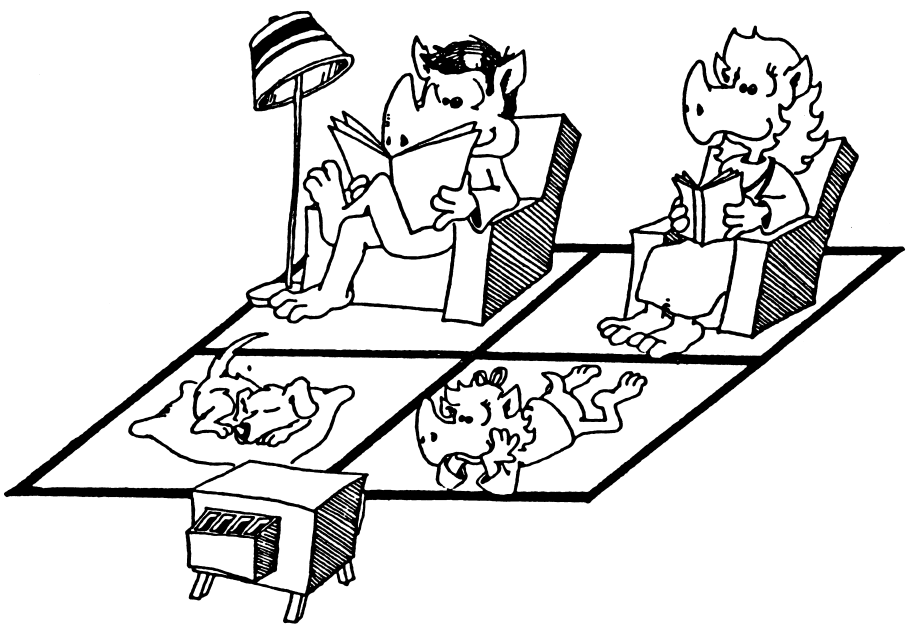
```
10 REM Phone list
20 cls: PRINT
30 DIM NA$(20), NU$(20)
35 I=0
40 PRINT "Enter names and numbers "
50 PRINT: INPUT"NAME? ";NA$(I)
60 INPUT"NUMBER? ";NU$
70 I=I+1: GOTO 50
```

RUN. Press BREAK key to stop the program. SAVE to disk.

ONE DIMENSION, TWO DIMENSION, . . .

The arrays that have one index are called one dimensional arrays.

But arrays can have two or more indices. Two dimensional arrays have their "family members" put into a rectangle, like the days in a month on a calander.



EIGHT QUEENS

The Eight Queens puzzle asks you to put eight chess queens on the chessboard in such a way that no queen is attacked by any other queen. If you are not familiar with chess, look up the moves of the queen in an encyclopedia. Obviously you can't have two queens on the same row or column. Queens attack along the diagonal as well. There are 92 patterns of queens on the board that solve this puzzle.

```
  1 REM 8 queens
  2 GOTO 1000
100 ' ---------- main loop
115 M=R(I)
120 M=M+1:IF M=9 THEN GOSUB 600: GOTO 115
130 IF B(I,M) =0 THEN GOTO 700
140 GOTO 120
200 ' ---------- update attacked squares
210 FOR L=1 TO 8
215 B(I,L)=B(I,L)+D
220 B(L,J)=B(L,J)+D
225 NEXT L
500 ' ---------- diagonal
510 FOR K=1 TO 8
515 X=I+K:IF X>8 THEN 530
522 Y=J+K:IF Y>8 THEN 525
523 B(X,Y)=B(X,Y)+D
525 Y=J-K:IF Y<1 THEN 530
526 B(X,Y)=B(X,Y)+D
530 X=I-K:IF X<1 THEN 590
540 Y=J+K:IF Y>8 THEN 550
545 B(X,Y)=B(X,Y)+D
550 Y=J-K:IF Y<1 THEN 590
555 B(X,Y)=B(X,Y)+D
590 NEXT K:B(I,J)=Q
599 RETURN
600 ' ---------- go back
610 R(I)=0
612 I=QN
615 IF I=0 THEN END
620 J=R(I):D=-1:Q=0:GOSUB 200
640 QN=QN-1
699 RETURN
700 ' ---------- go ahead
710 R(I)=M:J=M:QN=QN+1:D=1:Q=-1
730 GOSUB 200:NM=NM+1
740 IF QN=8 THEN GOTO 800
780 I=I+1:KB=1:IF KB< >0 THEN GOSUB 900
799 GOTO 115
```

```
 800 ' ---------- solution
 810 BEEP
 814 NS=NS+1:GOSUB 900:GOSUB 612
 899 GOTO 115
 900 ' ---------- display
 910 FOR X=1 TO 8:FOR Y=1 TO 8
 920 LOCATE 2+2*X,T-1+2*Y:PRINT "";
 925 BB=B(X,Y)
 930 IF BB=-1 THEN PRINT CHR$(1)
 940 IF BB> 0 THEN PRINT " "
 950 IF BB= 0 THEN PRINT "X"
 955 IF BB<-1 THEN PRINT X,Y:END
 960 NEXT Y:NEXT X
 980 PRINT:PRINT:PRINT TAB(T-3);"SOLUTIONS";NS;"      MOVES";NM
 999 RETURN
1000 DIM R(8),B(8,8)
1010 CLS: WIDTH 40:LOCATE,,0 ' turn cursor off
1020 I=1:QN=0:NS=0:T=13
1023 ' Graphics: hold down Alt key, type digits on
1024 ' NUMERICAL keypad, then let up on Alt key
1025 ' Line 1030: use 218,196,194,196,... 191
1026 ' Line 1045: use 179 and spaces
1027 ' Line 1046: use 195,196,197,196,... 180
1028 ' Line 1048: use 192,196,193,196,... 217
1029 ' See appendix on ASCII characters in IBM BASIC Manual
```

```
1030 LOCATE 3,3:PRINT TAB(T);"ZDBDBDBDBDBDBDBD?"
1040 FOR I=1 TO 8
1045 PRINT TAB(T);"3 3 3 3 3 3 3 3 3"
1046 PRINT TAB(T);"CDEDEDEDEDEDEDED4"
1047 NEXT:I=1
1048 LOCATE 19,3:PRINT TAB(T);"@DADADADADADADADY"
1049 LOCATE 1,1:PRINT TAB(T);"     EIGHT QUEENS"
1999 GOTO 100
2000 PRINT"
```

This program uses two arrays. R(I) tells in which row the queen on the ith column is. B(I,J) keeps track of the board. The value of element I,J is a number that tells about that square on the board. If the value is -1, there is a queen on that square. If the value is 0, then no queen is attacking that square. Values of 1, 2, 3, etc. tell how many queens are attacking that particular square. The program takes about two hours to find all the solutions.

Assignment 31:

1. Write a program that stores the number of days in each month in an array. Then when you ask the user to enter a number <1 to 12>, it prints out the number of days in that month.

2. Finish the PHONE LIST program so that it prints out the list of names with the telephone numbers beside them.

3. We wrote "Eight Queens" for a standard 8x8 chess board. Change the "Eight Queens" so the user can choose any size board.

4. Change "Eight Queens" into super queens. Each can move like a Queen or like a Knight. Are there any solutions?

INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS

This lesson shows how to write 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 "Enter 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? (Avoids using the ENTER 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:

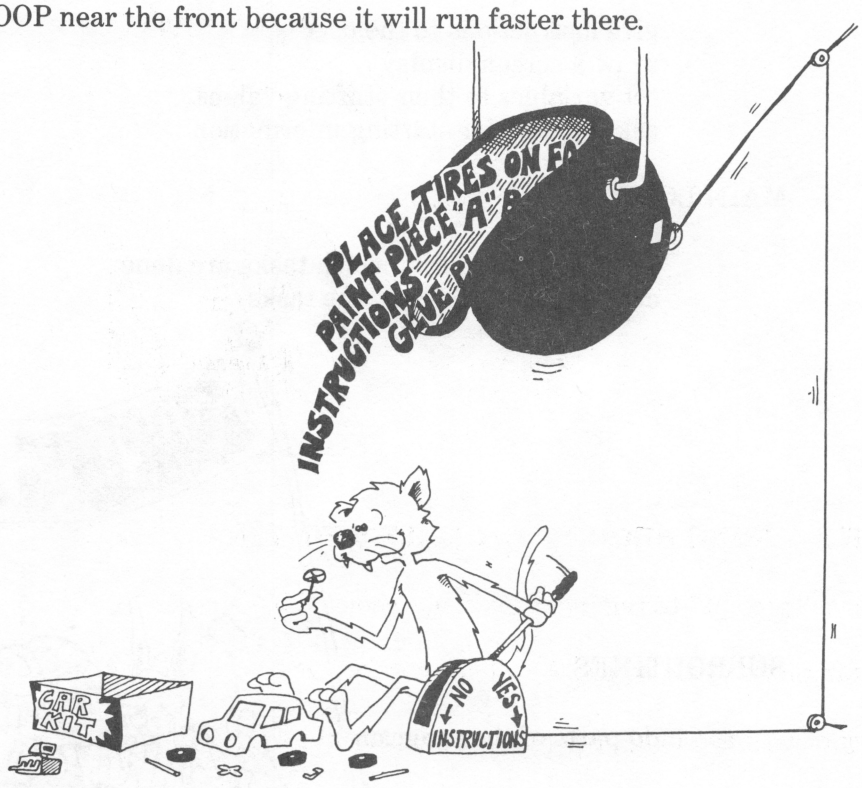
do parts of the program

PROGRAM OUTLINE

```
1 GOTO 1000:REM *** program name ***
---
100 REM MAIN LOOP
---
---       calls subroutines
---
199 END
1000 REM
1001 REM *** program name ***
1002 REM
---       REMs that give a description of the
---       program, variable names, etc.
---
1999 REM
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 STUFFF 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

1. Between line 2 and line 99 for subroutines that must run fast.
2. After line 2999 for starting stuff subroutines.
3. 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

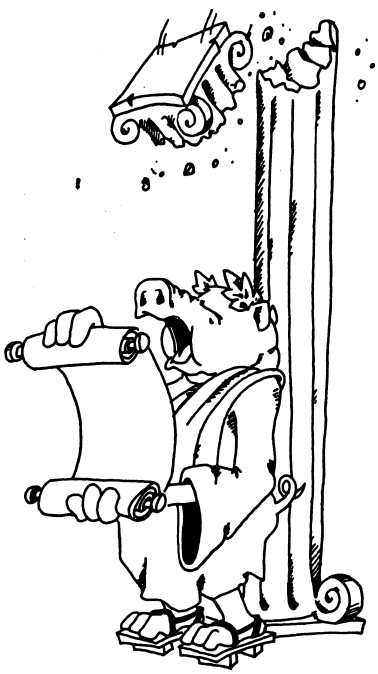
SCROLLING gives headaches!

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 LOCATE to print exactly 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 INKEY$ command to enter single letters. This saves having to press Enter.

```
380 PRINT "Do you need instructions? <y/n> "
382 R$=INKEY$:IF R$="" THEN 382
384 IF R$="y" THEN 600
```

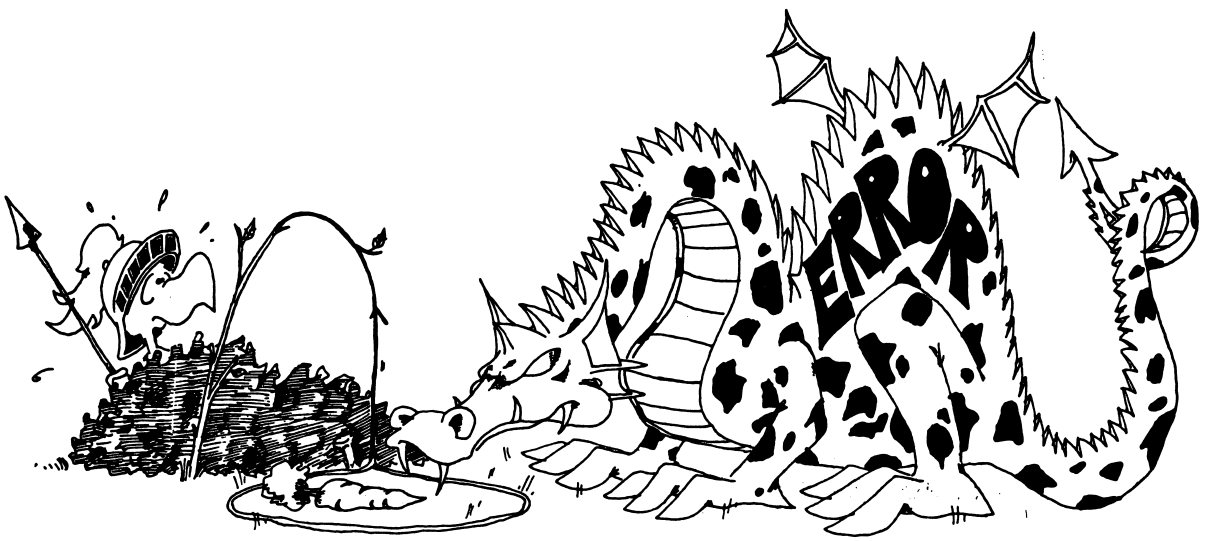
SET TRAPS FOR ERRORS

Example: Add this line to the above lines:

```
386 IF R$< >"n" THEN GOTO 380
```

Line 380 asked for only two choices, "y" or "n." If the user presses some other key, line 386 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. Make a program to write a very large number, 50 digits. Pick the digits at random. Put a comma between each set of three digits.

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 to get DRAGONET.
   Put it at the front of the alphabet and the rest of
   the letters after it in normal order

   DRAGONETBCFHIJKLMPQSUVWXYZ cipher alphabet
   ABCDEFGHIJKLMNOPQRSTUVWXYZ normal alphabet

The user chooses to code or decode from a menu.

INSTRUCTOR NOTES 33 DEBUGGING, STOP, CONT

The "sigh and moan" technique being a loser, our students need a bag of tricks that help isolate program bugs, and should practice on programs that they are writing as they go through this book.

The inexperienced debugger feels hopeless inertia when "it doesn't work right." Rather than sit and stare, it is more useful to try some changes. Any changes are better than none, but random changes are very inefficient. The best changes are those that eliminate sections of the program from the list of possible hiding places for the bug.

As programs grow in complexity, more of the bugs result from unforseen interactions between separated parts of the program. The bag of tricks we offer helps find these also. Delay loops, PRINT commands, and STOP statements help the student see how the program is functioning.

Don't overlook those techniques you can use after the program is stopped with a Ctrl Break, STOP, or an END. You can PRINT out any variable values you like, so as to see what the program has done. You can also do arithmetic in the PRINT command, to check what the program should be doing.

QUESTIONS:

1. How can you make the computer print

   Break in line 55

   by adding a command to the program?

2. How are the STOP and the END commands different?

3. How are the STOP statement and Ctrl-Break keys different?

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 Ctrl-Break keys 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 cls
25 N=INT(RND(8)*200)
30 FOR I=0 TO 200
40 IF I=N THEN STOP
50 NEXT I
```

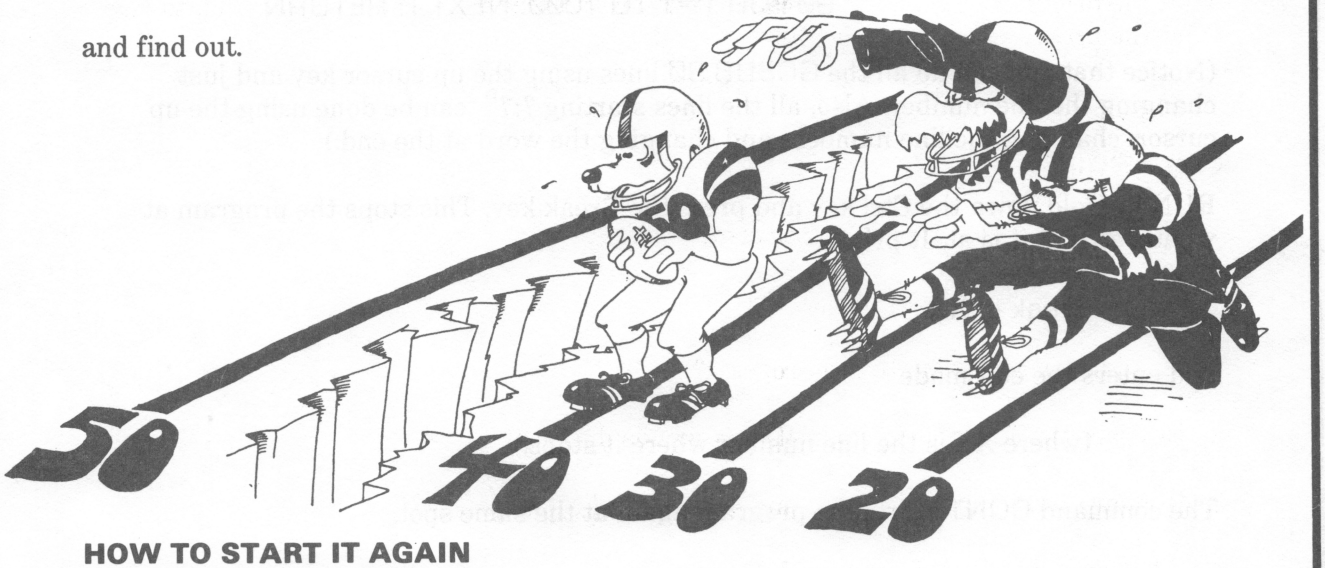
The program will stop, and the computer will print a message:

```
Break in 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 with the CTRL and C keys.

Try it:

```
10 REM Go forever
15 cls: ?: ?: ?
20 ?: ? "Mud"
21 GOSUB 90
22 ?: ? "     Turtles"
23 GOSUB 90
24 ?: ? "         Of"
25 GOSUB 90
26 ?: ? "             The"
27 GOSUB 90
28 ?: ? "                 World"
29 GOSUB 90
30 ?: ? "                     Unite!"
31 GOSUB 90
40 GOTO 10
90 FOR T=1 TO 1000: NEXT T: RETURN
```

(Notice that you can do all the GOSUB 90 lines using the up cursor key and just changing the line number. Also, all the lines starting ?:?" can be done using the up cursor, changing the line numbers and changing the word at the end.)

RUN it. Hold down the Ctrl key and press the Break key. This stops the program at whatever spot it is at. It prints:

```
Break in 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.

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 CTRL and Break keys 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:

1. List parts of the program and study them.

2. Use the PRINT command to look at variables. Do they have the values you expected?

3. Do little calculations on the computer in the "calculator mode" (another name for "edit mode") to check what the computer is doing.

4. Use the LET command to change the values of variables.

If you find the trouble, you may add a line, change a line, or delete a line.

STARTING THE PROGRAM AGAIN

There are four ways to start a program. They are:

CONT	if you have not changed the program
GOTO XX	where XX is a line number
RUN XX	where XX is a line number
RUN	your old friend

You may use the CONT command if you have not:

added a line
deleted a line
or changed a line by editing it

Or you may start RUNning the program at a different spot by entering (without line number) the command:

GOTO XX

where XX is the line number where you want to restart the program.

If you have changed the program, your only choice is to start at the beginning or at some other line number XX with RUN.

What is the difference between these four ways?

CONT GOTO XX

These two ways use the values in the variable boxes left over from the last time you ran.

CONT	starts at the line where the break occurred.
GOTO XX	starts at line XX

RUN RUN XX

These two ways throw away all the variable boxes made the last time, then execute the program.

RUN	starts at the first line of the program
RUN XX	starts at line XX

CONT can only restart a program that was stopped with a break from a STOP or Ctrl Break. But RUN, RUN XX, and GOTO XX can also start a new program.

DEBUGGING

Little errors in your program are called "bugs."

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

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

2. If the computer just keeps running but doesn't do the right thing, stop it and 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 STOPs 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.

2. Go back and fix up some other program that you have written.

DISK USAGE

DISK CARE

The student should be instructed on the care of diskettes at this time. *Especially:*

1. Do not touch the brown or grey magnetic disk through the oblong holes.

2. Do not bend the diskette.

3. Insert and remove the disk carefully from the drive.

4. Always put the disk back in its protective cover after use.

5. Do not spill food or drink on the diskette. In fact, it is better not to snack while at the computer.

6. Keep the disk away from magnetic fields. All speakers have magnets in them and they have the capability to erase information from a disk whether the speaker is *on* or *off*. The monitor or T.V. also has magnetic fields, so does the telephone and most motors.

PREPARING A DISK FOR THE STUDENT

The computer will be much easier for your student if you make special disk for her. The disk will automatically start up the computer, set the screen, list the files on her disk and leave her in BASICA ready to write programs or load files.

Place the DOS diskette in the disk drive and turn on the IBM-PC computer. After a moment, the disk drive will start, then stop.

The computer is asking for date and time. Just press the Enter key twice.

(The message may be unreadable if you are using a color monitor. If so, after pressing the Enter key twice enter:

```
mode 40
```

and you will see the A> prompt clearly.)

With the DOS diskette still in the drive, enter:

```
format a:/s
```

The computer will reply:

```
Insert new diskette for drive A:
and strike any key when ready
```

Put a brand new disk in the drive and press a letter key. The computer will say:

```
Formatting . . .
```

and you will hear the disk drive run for quite a while. Then the computer will say:

```
Formatting . . . Format complete
System transferred

     362560 bytes total disk space
      14336 bytes used by system
     308224 bytes available on disk

Format another (Y/N)?
```

Answer by typing n.

Now we need to put the BASICA file on the disk. Take out the student's disk and put the DOS disk back in. Then enter:

```
copy basica.com b:
```

The disk will run, then the computer will print:

```
Insert diskette for drive B: and strike any key when ready
```

Take out the DOS disk and put the student's disk back into the drive. (Unless you have two drives, in which case put the student's disk into drive B.) Press a letter key.

The drive will run for a moment, then the computer prints:

```
1 File(s) copied
```

In a similar way, copy the COLORBAR.BAS file to the student's disk.

We want the student's disk to automatically prepare the computer for her use each time she turns the computer on. So we will put an AUTOEXEC.BAT file on the disk which will automatically load and run a BASIC program named STUDENT. Enter:

```
copy con:autoexec.bat
basica student
```

Now press the F6 key, then press the Enter key. You will see the prompt:

```
^Z
```

```
Insert diskette for drive A: and strike any key when ready
```

The student's disk is already in the drive, so just press a letter key. The drive will run for a moment and the computer will say:

```
1 File(s) copied
```

Now it is time to write the BASIC program. Enter:

```
basica
```

and the computer will load it from the disk. Then enter this program, putting your own student's name in line 70.

```
10 rem --- student ---
12 rem
14 rem greeting program for
16 rem
18 rem KIDS AND THE IBM-PC
20 rem
50 width 40: rem omit if you have a green screen monitor
60 cls: print: print: print
70 print tab(13);"MINDA'S DISK"
75 print: files
80 print "Entering EDIT MODE"
90 new
```

DO NOT RUN THIS PROGRAM YET! Because of line 90, it will erase itself after it runs!

SAVE it to disk. Enter:

```
save "student"
```

This disk is the student's own. She will boot the system with it when she starts work, and will keep all the programs she writes on it.

RESERVED WORDS IN BASIC

ABS	AND	ASC	ATN	AUTO	
BEEP	BLOAD	BSAVE			
CALL	CDBL	CHAIN	CHR$	CINT	CIRCLE
	CLEAR	CLOSE	CLS	COLOR	COM
	COMMON	CONT	COS	CSNG	CSRLIN
	CVD	CVI	CVS		
DATA	DATE$	DEF	DEFDBL	DEFINT	DEFSNG
	DEFSTR	DELETE	DIM	DRAW	
EDIT	ELSE	END	EOF	EQV	ERASE
	ERL	ERR	ERROR	EXP	
FIELD	FILES	FIX	FNxxxxxx	FOR	FRE
GET	GOSUB	GOTO			
HEX$					
IF	IMP	INKEY$	INP	INPUT	INPUT#
	INPUT$	INSTR	INT		
KEY	KILL				
LEFT$	LEN	LET	LINE	LIST	LLIST
	LOAD	LOC	LOCATE	LOF	LOG
	LPOS	LPRINT	LSET		
MERGE	MID$	MKD$	MKI$	MKS$	MOD
	MOTOR				
NAME	NEW	NEXT	NOT		
OCT$	OFF	ON	OPEN	OPTION	OR
	OUT				
PAINT	PEEK	PEN	PLAY	POINT	POKE
	POS	PRESET	PRINT	PRINT#	PSET
	PUT				
RANDOMIZE	READ	REM	RENUM	RESET	RESTORE
	RESUME	RETURN	RIGHT$	RND	RSET
	RUN				

SAVE	SCREEN	SGN	SIN	SOUND	SPACE$
	SPC	SQR	STEP	STICK	STOP
	STR$	STRIG	STRING$	SWAP	SYSTEM
TAB	TAN	THEN	TIME$	TO	TROFF
	TRON				
USING	USR				
VAL	VARPTR	VARPTR$			
WAIT	WEND	WHILE	WIDTH	WRITE	WRITE#
XOR					

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(Ø)	is the first member of the array A
B$(7)	is the eighth member of the array B$
CD(3,M+1)	is a member of the array CD

arrow keys
Four keys on the computer that have arrows on them. They move the input cursor to the left and right, up and down. ASCII Stands for American Standard Code For Information Interchange. Each character has an ASCII number.

assertion
The name of a phrase that can be TRUE or FALSE. The "phrase A" in an IF statement is an assertion. An assertion has a numeric value of 0 or 1. See expression, TRUE, FALSE, logic, phrase A. Example:

the assertion	"A"< >"B"	is TRUE
the assertion	3 = 4	is FALSE

background
The part of the screen that is blank, not having characters on it.

BASIC
Beginners's All-purpose Symbolic Instruction Code. A computer 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 IBM computer makes a "beep" 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.

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.

buffer
A storage area in memory for temporary storage of information being inputed or outputed from the computer.

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 and then returns to the calling spot.

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."

column
Things arranged vertically. See row.

command
In BASIC a command makes the computer do some action, such as erase the screen and memory by the NEW command. See statement, expression. Some commands need expressions to be complete. Example:

```
SAVE "doggie"
```

concatenation
Means sticking/gluing 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 IBM computer:

INPUT cursor	a flashing line on the screen
PRINT cursor	invisible, "shows" where next character will be printed

data
BASIC has two kinds of data: numeric and string. Logical data (TRUE, FALSE) are types of numeric data.

debug
Means to run a program, find the errors and fix them. You fix the errors by editing the program. See edit.

delay loop
A part of the program that just uses up time and does nothing else. Example:

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

duration
A number in a SOUND statement that tells how long the sound will last.

edit
There are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

edit mode
When ready and the flashing cursor show, you are in the edit mode. The computer awaits commands or the entering or program lines.

enter
To put information into the computer by typing, then pushing the Enter key. The information goes into the input buffer as it is typed. When Enter 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 are within reasonable bounds.

execute
To run a program or to perform a single command or statement.

expression
A portion of a statement that has a single value, either a number or a string. See value. Examples:

```
7*X+1
"DOPE "< > N$
A$ + "HAT"
```

FALSE
The number 0. See logic, TRUE, 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 (numeric or string) determined by its arguments. See value, argument. The functions treated in this book are:

ASC, CHR$, INT, LEN, RND, LEFT$, MID$, RIGHT$, STR$, VAL, TAB

garbage
A random mess of characters in memory. Usually due to human or machine error.

graphics
Means picture drawing.

index
An array name is followed by one or more numbers or numeric variables in parentheses. Each number is an index. Another word for index is "subscript."

```
Q(7,J)        7 and J are indices
```

integers
The whole numbers, positive, negative and zero.

I/O
Input/Output. Input from keyboard, disk, etc. Output to screen, printer, disk, etc.

joystick
A device used in games. It is like the control stick used in early airplanes. It can detect eight different directions, as well as "centered."

jump
The GOTO command makes the computer jump to another line in the program, rather than execute the next line.

line
Lines start with a number followed by a statement which may contain expressions, etc.

Parts of a line:

```
16 IF 7<=INT(Z) THEN PRINT LEN(Q$+"R")+2;"RAT": GOTO 40
```

16	line number
IF 7<=INT(Z) THEN PRINT . . . "RAT"	statement
GOTO 40	statement
7<=INT(Z)	an assertion
7<=INT(Z)	an expression
LEN (Q$+"R")+2;"RAT"	an expression
INT(Z)	a function
LEN(Q$)	a function
Z	argument
Q$+"R"	argument
7, "R", 2, "RAT"	constants
<=, +	operations
IF, INT, THEN, PRINT, LEN, GOTO	reserved words

line buffer
The storage space that receives the characters you type in. See buffer.

line edit
Retyping parts of a line to correct it.

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.

listing
A list of all the lines in a program.

load
To transfer the information in a file on tape to the memory of the computer by using the LOAD command.

logic
The part of a program that compares numbers or strings. The relations =, < >, <, >, <=, and >= are used. See assertion, phrase A, AND, OR, NOT.

loop
A part of the program that is done over and over again. There are many kinds of loops: FOR . . . NEXT loops, "home made" loops that use IF . . . commands with a loop variable and DO WHILE... and DO UNTIL . . . loops.

loop variable
Is the number that changes as the loop is repeated. For example:

```
40 FOR I=1 TO 5
50 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 labels on the front and information inside.

menu
A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

message
A statement that tells what is expected in an INPUT statement. Example:

```
61 INPUT "AGE";A
```

monitor
Has two meanings. We use it to mean a box with a TV type screen which 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.

```
L=INT(LEN(P$)+3)          nested functions
X=5*(6+(7*(8+K)))         nested parentheses
```

number
Is one type of information in BASIC. The other is "string." The numbers are generally decimal numbers. See integer, strings.

operation
In arithmatic: addition, subtraction, multiplication and division, with symbols +, –, *, and /. The only operation for strings is concatenation.

phrase A
Is a phrase in this book that stands for an assertion in an IF statement. See assertion. Example:

```
IF A>4 THEN 500        A>4 is "phrase A"
```

pitch
The number in a SOUND statement that tells the musical pitch of the sound. Pitch is the same as "note" and can be high or low.

pixel
Picture element. The smallest dot that is placed on the screen in a graphics mode.

pointer
A number in memory that tells where in a list of DATA you are at the present moment.

program
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.

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 number
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 user of the computer.

punctuation
The characters like period, comma, /, ?, !, $, etc.

random number
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 graphics setup subroutine
```

reserved words
A list of words and abbreviations that BASIC recognizes as commands, statements, or functions. The reserved words cannot be used as variable names.

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 horizonally (across).

save
To put the program that is in the computer's memory on disk.

screen
The monitor screen (similar to a TV screen) that is hooked up to the computer. See monitor.

scrolling
The usual way the computer writes to the full screen is to put the new line at the bottom of the screen and push all the old lines up. This is called "scrolling."

simple variable
A variable that is not an array variable.

stack
Is a data type used in machine language programming. The data are arranged in a column and the last one put on is the first one taken off.

starting stuff
Is the name given in this book to initialization material in a program. It includes REMs for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things that need to be done just once at the beginning of a program run.

statement
The smallest complete section of a program. It starts with a command. The command may have expressions in it.

store
To put information in memory or to save it on 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
Another name for "index." 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.

timing loop
A loop that does nothing except use up a certain amount of 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 computer. 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, takes a copy of what is in the box back to the expression, puts it where the variable name was, and then continues to evaluate the expression. See variable name.

variable name
A variable is either a string variable or a numeric variable. The name tells which it is. String variables have names ending in a "$" sign. Numeric variables do not. The variable name has one or two characters. The first character must be a letter, the second a letter or a number.

variable, array
See array.

variable, simple
See simple variable.

ERROR MESSAGES

1. NEXT without FOR

The computer reached a NEXT command and cannot find the FOR that belongs to it. You may have an extra NEXT. You may have missed the FOR because a GOTO jumped you over it. Maybe the NEXT has the wrong loop variable. Example:

```
10 FOR I=1 TO 7
20 PRINT I
30 NEXT X        (It should be NEXT I)
```

2. Syntax error

You made a "spelling" error in a line you tried to enter. Examples:

(a) `10 A)=8`

(b) `10 DEM A(2,2,2)` (It should be DIM)

Maybe the data in a DATA statement did not match the variable that called it. Example:

```
10 READ A
20 DATA "HI"
```

3. RETURN without GOSUB

The computer found a RETURN statement before it went through the GOSUB statement. Perhaps you "fell through" to a subroutine at the end of the program. Example:

(a)
```
 10 GOSUB 100
 20 REM The main program goes here.
 80 REM After running the main program,
 81 REM you fall through to the subroutine.
100 REM SUBROUTINE
110 REM The subroutine is here.
199 RETURN
```

You can fix this program by adding line:

```
99 END
```

4. Out of data

You used READ so many times that it used up all the data in your DATA statements. Then you used READ once more. Example:

(a)
```
10 FOR I=1 TO 4
20 READ N$
30 NEXT I
99 DATA TOM, DICK, HARRY
```

5. Illegal function call

There are a number of different things that may give this message.

Maybe you tried to use an array that has a bad index. Example:

```
10 DIM A(-5)
```

Maybe you used a bad argument in a function.

Maybe you tried to LIST a BASIC program that the programmer had "protected."

Maybe you tried to delete line numbers that do not exist.

6. Overflow

You tried to make too big a number. Example:

```
10 X=2
20 FOR I=1 TO 50
30 X=X * X
40 PRINT X
50 NEXT I
```

7. Out of memory

You tried to use more memory than the machine has. Your program may have too many lines. Or it may have variables that use up too much room. Example:

```
10 DIM A(99,99)
```

Maybe you have too many loops, or too many subroutines. Your graphics may use painting that is too complex. Or your expressions may be too complex (have too many parentheses).

8. Undefined line-number

You used a GOTO or GOSUB or RESTORE to a line number that is not in the program.

Or you used RUN XX and there is no line XX in the program.

9. Subscript out of range

You used an array with a subscript (index) that was negative or too large. Examples:

(a)
```
10 A(-3)=5
```

(b)
```
10 DIM B(8)
20 B(99)=4
```

10. Duplicate Definition

You tried to dimension an array that already had a dimension. Examples:

(a)
```
10 DIM A(3), DIM A(5)
```

(b)
```
10 FOR I=1 TO 3
20 DIM A(7)
30 NEXT I
```

(c)
```
10 A(2)=6
20 DIM A(20)
```

In example (c), line 10 is OK. If you forget to dimension an array, the computer gives it a dimension of 10. But you cannot later execute a DIM statement for that array.

11. Division by zero

The computer tried to divide a number by zero, or by a variable that equals zero. Example:

```
10 A=0
20 B=7/A
```

12. Illegal direct

You tried to enter a command without a line number. Most commands are allowed in direct mode. Some are not. Example:

```
DEF FNA(X)=X*X
```

This DEF FN statement can only be used in a numbered line.

13. Type mismatch

You tried to use a string where a number was needed, or a number where a string was needed. Examples:

(a)
```
10 A="HI"
```

(b)
```
10 B$=D
```

14 Out of string space

You used so many long strings that the computer ran out of memory.

15 String too long

Each string must be 255 or less characters long. Example:

```
10 A$="HI"
20 A$=A$+A$
25 PRINT A$
30 GOTO 20
```

16. String formula too complex

Your statement with strings is too complicated. Break it up into several statements.

17. Can't continue

You tried to CONTinue a program. But you can only continue a program that stopped because it executed a STOP statement or you pressed Ctrl-Break.

You cannot CONTinue a program if you have added, deleted, or changed any lines after it stopped, or if you erased it with NEW.

18. Undefined user function

We did not teach about user defined functions in this book. This error comes when you use a variable whose name starts with the letters FN. Example:

```
10 Y=FNA(8)
```

(Unless you did a DEF FNA . . . first.)

19. No RESUME

We did not teach about ERROR statements in this book. The ERROR statement must have a RESUME statement after it.

20. RESUME without ERROR

Likewise, the RESUME statement must have an ERROR statement before it.

21. Missing operand

You left out part of the arithmetic statement. Example:

```
10 A = 3 *
```

or

```
20 B$ = "HI" +
```

22. Line buffer overflow

You tried to enter a line that had too many characters.

23. Device Timeout

The computer waited for a printer or other device to say it was ready to go. The computer got tired of waiting.

24. Device Fault

The printer or other device is not turned on or is not working correctly.

25. FOR without NEXT

The computer entered a FOR loop but reached the end of a program before it found the NEXT that belongs to it.

26. Out of paper

Your printer is not on or is out of paper. You didn't want to use the printer? Did you use a LLIST command? It calls the printer.

27. WHILE without WEND

We did not teach about WHILE or WEND statements in this book.

28. WEND without WHILE

Same as above.

29. FIELD overflow

This happens when you OPEN a file and write to it. We did not teach about using files in your programs. The files we talked about were BASIC programs on disk.

30. Internal error

The computer found something wrong as it tried to run your program. It may be the program on disk is bad, or your computer hardware may be sick.

31. Bad file number

32. File not found

The computer could not find the file you want when you did a LOAD or a KILL. Do FILES to see if the file is on the disk. Try again, using the file name with extension.

Or maybe the computer is looking on the wrong drive.

33. Bad file mode

Has to do with OPEN which we did not teach in this book.

34. File already open

Has to do with OPEN. You cannot KILL a file that is open.

35. Device I/O Error

BAD NEWS! The computer could not read the disk and the error may be due to a bad disk, or the computer may be working poorly.

36. File already exists

Each file name on a disk must be different than all the others. Try to SAVE again, using a different file name.

37. Disk full

The disk cannot hold any more files, either because the files already there are too long, or there are too many of them.

38. Input past end

This should only happen on files you have OPENed.

39. Bad record number

Should only happen on files you have OPENed and you are using a PUT or a GET. We have not taught about these things in this book.

40. Bad file name

You used a bad file name. Look in Lesson 14 to see how to make good file names.

41. Direct statement in file

This happens in programs on disk in the ASCII mode. We did not teach about this in the book.

42. Too many files

You are doing a SAVE. Either the disk is full or you have an incorrect file name.

43. Device Unavailable

You tried to use a device (disk, printer, etc.) that is not connected to your computer, or is not turned on.

44. Communication buffer overflow

This happens when you are doing complicated I/O. We do not teach about this in the book.

45. Disk Write Protect

The disk you are trying to SAVE to has a tab stuck over its "write protect" notch. Take the tab off, or use another disk.

46. Disk not Ready

You forgot to put a disk in the drive, or you forgot to close the door.

47. Disk Media Error

The disk drive hardware detected an error on the disk. The disk may be bad. Copy its good files to another disk. Then try to FORMAT the bad disk. If you can't, the disk must be thrown away.

48. Advanced Feature

You are using cassette BASIC or Disk BASIC and the command you used only works in Advanced BASIC.

-- Unprintable error

There is no error number for the error the computer found. It probably has to do with your using the ERROR statement. We do not teach about the ERROR statement in this book.

ANSWERS TO ASSIGNMENTS

1.3

```
10 REM Greeting
20 PRINT" Hi There,"
30 PRINT" Computer"
```

```
10 REM Names
15 COLOR 0,4
20 PRINT" Minda"
30 PRINT" Anne"
40 PRINT" Carlson"
50 COLOR 7,0
```

2.1

```
10 REM Names
15 COLOR 3,0
16 BEEP
20 PRINT" Minda"
25 BEEP
30 PRINT" Anne"
35 BEEP
40 PRINT" Carlson"
50 COLOR 7,0
```

3.5

```
10 REM Birds
15 CLS
20 PRINT
22 BEEP
25 PRINT "---O---"
30 PRINT
40 PRINT
42 BEEP
50 PRINT "    ---O---"
60 PRINT
70 PRINT
75 BEEP
80 PRINT " ---O---"
```

4.3

```
10 REM Smile
12 CLS
20 PRINT
30 PRINT
40 PRINT
50 PRINT "    OO      OO        "
60 PRINT
61 PRINT
62 PRINT
63 PRINT " *                 * "
64 PRINT "  *               * "
65 PRINT "   *             *"
66 PRINT "    ********* "
```

5.1

```
10 REM Talking
15 CLS
20 PRINT
22 PRINT
24 PRINT
30 PRINT "Hello. What is your name?"
32 PRINT
34 INPUT N$
36 PRINT
40 PRINT "Well,"
42 PRINT
44 PRINT N$
46 PRINT
50 PRINT "It's silly to talk to computers!"
```

5.2

```
10 REM The string box
12 CLS
20 PRINT "What is your favorite color?"
25 INPUT C$
27 PRINT
30 PRINT "I put that in box C$."
32 PRINT
35 PRINT "Now, your favorite animal?"
40 INPUT C$
42 PRINT
45 PRINT "I put that in box C$ too."
```

```
47 PRINT
50 PRINT "Now let's print what is in box C$"
52 PRINT
55 PRINT "It is:"
57 PRINT
60 PRINT C$
```

6.1

```
10 REM Music
12 CLS
20 PRINT "What is your favorite musical group?"
25 INPUT G$
27 CLS
30 PRINT "What tune do they play?"
35 INPUT T$
40 CLS
50 PRINT
55 PRINT G$;" plays ";T$
```

6.2

```
10 REM Same as above except:
55 PRINT G$;
56 PRINT " plays ";
57 PRINT T$
```

7.2

```
10 REM Feelings
15 CLS
20 PRINT
22 PRINT
24 PRINT "How is the weather?"
26 PRINT
28 INPUT W$
30 PRINT "And how do you feel?"
32 PRINT
34 INPUT F$
36 PRINT
38 PRINT "You mean:"
40 PRINT
45 S$=W$ + " and " + F$
50 PRINT S$
```

8.3

```
10 REM Teen Times
11 REM
15 COLOR 5,0
20 PRINT "Teen Power"
21 PRINT
22 PRINT
23 PRINT
30 GOTO 20
```

8.5

```
10 REM Friends
15 CLS
16 COLOR 1,0
20 PRINT "Minda"
25 PRINT
28 COLOR 3,0
30 PRINT "Nell"
35 PRINT
95 REM press Ctrl-Break to stop program
99 GOTO 20
```

9.A1

```
10 REM Boys and Girls
15 CLS
20 PRINT
25 PRINT "Are you a boy or a girl?"
26 PRINT
28 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"
```

9B.1

```
 2 REM PIZZA
 3 REM by Chris Clark, Jr. Age 14 going on (you figure it out)
 4 CLS
 5 PRINT "Hallo, Ay am Mario, your pizza man."
 6 PRINT
 7 PRINT "Just tell me ze gory details and I'll do the rest"
```

```
  9 PRINT
 10 PRINT "What size should zis pizza be? (s-m-l)"
 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$="l" THEN PRINT "You must have a big bunch at home!"
 40 PRINT
 41 PRINT "Now, you want double chees on zis (y-n)?"
 42 INPUT CH$
 45 REM etc.
 50 REM mushrooms, etc.
 60 REM anchovies, etc.
 80 REM peppers, etc.
 90 REM meat,etc.
150 PRINT "Hokay, here is your pizza!"
154 IF S$="s" THEN PRINT "Wan small pizza with ";
156 REM etc.
160 IF BASE$="p" THEN 165
161 PRINT "pepperoni"
165 REM etc., etc.
238 PRINT
240 FOR J=1 TO 1000
242 NEXT J
```

9B.2

```
10 REM === Color Guessing Game ===
20 CLS
23 PRINT
24 PRINT
25 PRINT "Player 2 Turn your back"
27 PRINT
30 PRINT "Player 1 Enter a color"
35 INPUT C$
42 PRINT
43 PRINT
50 PRINT "Player 2 Turn around and guess"
52 PRINT
54 PRINT
55 INPUT G$
56 PRINT
60 IF G$=C$ THEN 80
61 PRINT "wrong."
67 PRINT
70 GOTO 55
80 PRINT "RIGHT!!!"
```

10.1

```
10 REM Birth Year
15 CLS
30 PRINT "How old are you?"
32 PRINT
34 INPUT A
36 PRINT
40 PRINT "And what year is it now?"
42 PRINT
45 INPUT Y
50 LET B=Y-A
52 PRINT
55 PRINT "Has your birthday come yet this year?"
58 PRINT "<y-n>"
59 PRINT
60 INPUT Y$
65 IF Y$="y" THEN 70
67 LET B=B-1
70 PRINT
75 PRINT "You were born in";B;"."
```

10.2

```
10 REM Multiplication
15 CLS
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
```

10.2A

```
10 REM Multiply
12 CLS
20 PRINT "Give me two numbers"
21 PRINT
25 INPUT A,B
26 PRINT
30 PRINT "Here is their product:";
35 PRINT A*B
```

11A.1

```
10 REM nicknames
15 CLS
20 PRINT "What is your last name?"
22 PRINT
24 INPUT L$
28 CLS
30 PRINT "Someone type the nickname"
32 PRINT
34 INPUT N$
36 CLS
38 PRINT TAB(5);N$;TAB(15);L$
40 FOR T=1 TO 2000:NEXT T
50 GOTO 10
```

11A.2

```
10 REM !"$#! Insults !#$"!
15 CLS
16 PRINT
17 PRINT
20 PRINT "Hey you!! What is your name?"
22 PRINT
25 INPUT N$
30 CLS
31 PRINT
32 PRINT
35 PRINT N$
36 PRINT
37 PRINT
38 FOR T=1 TO 2000:NEXT T
40 PRINT "Bah!!"
41 PRINT
42 PRINT
45 BEEP
50 PRINT "Your father eats leeks!!!"
```

11B.2

```
10 REM s l o w  p o k e
20 CLS
22 PRINT
24 PRINT
28 FOR T=1 TO 1000:NEXT T
29 PRINT" I'm"
30 BEEP
32 FOR T=1 TO 1000:NEXT T
34 PRINT" so"
40 BEEP
42 FOR T=1 TO 1000:NEXT T
44 PRINT" tired!!!"
46 BEEP
```

12.3

```
10 REM I got your number!
20 CLS
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.
61 IF N>10 THEN END
64 FOR T=1 TO 2000:NEXT T
66 CLS
68 GOTO 30
70 PRINT "That's all, folks"
```

13.1

```
10 REM ** A pair of dice **
15 CLS
16 RANDOMIZE
18 CLS
```

```
20 LET D1=1+INT(RND*6)
22 LET D2=1+INT(RND*6)
25 D=D1+D2
30 PRINT "The roll gave:"
32 PRINT
33 PRINT " The first die ";D1
34 PRINT " The second die";D2
35 PRINT " The dice       ";D
47 PRINT
48 PRINT
50 PRINT "Again?"
51 PRINT
55 INPUT Y$
60 IF Y$="y" THEN 18
```

13.2

```
10 REM Paper, Scissors, Rock
11 RANDOMIZE
12 CLS
13 PRINT
14 PRINT
16 PRINT TAB(12);" Play the "
18 PRINT
19 PRINT TAB(12);" P a p e r ":PRINT
20 PRINT TAB(12);"    S c i s s o r s ": PRINT
21 PRINT TAB(12);"         R o c k "
22 PRINT
23 PRINT TAB(12);" Game against the computer"
24 PRINT:PRINT:PRINT:PRINT
25 PRINT "Enter your choice <p,s,r>"
29 REM------ computer chooses its move
30 C=INT(RND*3)+1
31 IF C=1 THEN C$="p"
34 IF C=2 THEN C$="s"
36 IF C=3 THEN C$="r"
37 REM------ C$ is the computer's choice
38 INPUT Y$
39 REM------ Y$ is your choice
40 REM------ Is there a tie?
50 IF C$< >Y$ THEN GOTO 60
52 REM------ if C$=Y$, there is a tie
55 PRINT "       tie"
57 GOTO 30
60 REM------ no tie, who wins?
61 REM
62 IF C$="p" THEN IF Y$="s" THEN GOTO 70
```

```
63 IF C$="s" THEN IF Y$="r" THEN GOTO 70
64 IF C$="r" THEN IF Y$="p" THEN GOTO 70
65 REM------ computer wins
66 PRINT "        computer wins"
69 GOTO 30
70 REM------ you win
72 PRINT "        you win"
79 GOTO 30
90 REM and the game by pressing
91 REM        'Ctrol-Break'
```

15.2

```
10 REM !!! Vacation !!!
13 CLS
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 can 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 IF D<.5 THEN PRINT "Flip pennies with your kid brother":GOTO 90
41 IF D<1 THEN PRINT "Spend the afternoon in beautiful Hog Wallow, Mich."
   :GOTO 90
42 IF D<5 THEN PRINT "Enter a pickle eating contest in Scratchy Back, Tenn."
   :GOTO 90
47 REM etc.
58 IF D>1000000! THEN PRINT "Buy a cozy yacht and cruise the Caribbean Sea"
   :GOTO 90
73 REM etc.
86 PRINT "treat your whole school to a 'round the world trip!"
90 REM ending of program
```

15.3

```
10 REM Crazy
12 RANDOMIZE
15 CLS
20 PRINT "What is your name?"
21 PRINT
22 INPUT N$
30 CLS
40 PRINT
41 PRINT N$
45 PRINT
50 Z=INT(RND*3)+1
60 ON Z GOTO 70,80,90
70 PRINT "Has one brick short of a full load"
71 END
80 PRINT "Has bats in the attic"
81 END
90 PRINT "Hasn't got both oars in the water"
91 END
```

16.1

```
10 REM Jumping name
12 CLS
13 RANDOMIZE
15 INPUT "name";N$
16 CLS
20 FOR S=1 TO 50
30 X=INT(RND*30)+1
31 Y=INT(RND*22)+1
32 C=INT(RND*7)
33 COLOR C,0
35 LOCATE Y,X:PRINT N$
45 FOR T=1 TO 500:NEXT T
50 NEXT S
```

17A1

```
10 REM Counting by fives
12 CLS
20 FOR I=5 TO 100 STEP 5
30 PRINT I
35 FOR T=1 TO 500:NEXT T
40 NEXT I
```

17B2

```
10 REM Your name is falling
20 CLS
25 PRINT "Your name"
27 PRINT
30 INPUT N$
33 CLS
35 FOR I=1 TO 22
40 PRINT TAB(I);N$
42 FOR T=1 TO 200:NEXT T
45 NEXT I
```

17B4

```
10 REM Friends
15 CLS
20 PRINT "Give me your names"
25 INPUT N$
26 INPUT F$
30 FOR I=1 TO 5
31 BEEP
35 PRINT N$;" and ";
36 PRINT F$
38 PRINT
40 FOR T=1 TO 300:NEXT T
50 NEXT I
```

17 EXTRA

```
10 REM Chirping
12 COLOR 2,0:CLS
14 FLAG=0
15 FOR X=3 TO 35 STEP 2
19 LOCATE 15,X-1:PRINT " ";:REM erase
29 LOCATE 15,X:PRINT "  ":REM bird, wings down
35 FOR T=1 TO 200:NEXT T
40 LOCATE 15,X:PRINT " ";:REM erase
50 LOCATE 15,X+1:PRINT "  ":REM bird, wings down
55 FOR T=1 TO 200:NEXT T
56 FLAG=FLAG+1:IF FLAG=5 THEN BEEP:FLAG=1
60 NEXT X
99 COLOR 7,0
```

18.1

```
 10 REM Relatives
 12 CLS
 20 PRINT "Relation?"
 21 PRINT
 22 INPUT W$
 23 PRINT
 24 FL=0
 29 RESTORE
 30 READ R$
 32 READ N$
 34 IF R$="end" THEN 300
 36 IF R$=W$ THEN 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 PRINT R$;" ";N$
202 REM
220 FL=1
299 GOTO 30
300 REM
301 REM no relation
302 REM
310 IF FL=1 THEN 320
315 PRINT "You do not have a ";W$
320 FOR T=1 TO 3000:NEXT T
399 GOTO 12
```

19.2

```
10 REM loopy tunes
20 FOR I=1 TO 3
25 PRINT "sing"
26 PRINT
30 FOR J=1 TO 3
```

```
33 PRINT "tra ";
34 SOUND 200*I,2
40 FOR K=1 TO 3
43 PRINT "la ";
44 SOUND 300*I,2
50 NEXT K
51 PRINT
55 NEXT J
56 PRINT
57 PRINT
60 NEXT I
```

20B.3

```
 10 REM Sinbad's Magic Carpet
 11 RANDOMIZE
 12 COLOR 0,0,0:CLS
100 FOR I=1 TO 15
110 FOR J=1 TO 11
115 COLOR INT(RND*6)+1,0
120 LOCATE J,          I:PRINT "2"
121 LOCATE 22-J,       I:PRINT "2"
122 LOCATE 22-J,31-I:PRINT "2"
123 LOCATE J,      31-I:PRINT "2"
190 NEXT J:NEXT I
195 FOR I=1 TO 9999:NEXT I
199 COLOR 7,0,0
```

20B.4

```
10 REM cross my heart
25 FOR X=2 TO 23
26 CLS
27 COLOR 4,7,3
30 LOCATE 10,X
31 PRINT "M          M"
40 LOCATE 12,X
41 PRINT " i       i "
50 LOCATE 14,X
51 PRINT " n "
60 LOCATE 16,X
61 PRINT " d       d"
70 LOCATE 18,X
71 PRINT "a          a"
30 FOR T=1 TO 300:NEXT T
81 CLS:COLOR 2,7,3
```

```
82 LOCATE 14,X+5
83 PRINT CHR$(3)
84 FOR T=1 TO 300:NEXT T
90 NEXT X
99 COLOR 7,0
```

```
10 REM measles
11 COLOR 7,0,3
12 CLS
20 FOR I=1 TO 100
25 COLOR INT(RND(9)*8),0,0
30 X=INT(RND(9)*40)+1
31 Y=INT(RND(9)*24)+1
33 LOCATE Y,X:PRINT"O";
35 FOR T=1 TO 200:NEXT T
40 NEXT I
50 COLOR 7,0,3
```

```
10 REM snow flakes
12 COLOR 7,7,3:CLS
20 REM main loop
21 X=INT(RND(9)*39)+1                    'new spot
22 C=INT(RND(9)*7)+1                     'new color
25 FOR D=1 TO 23
27 COLOR 7,7,5
30 LOCATE D,X                            'erase
31 PRINT" ";
40 COLOR C,7
50 LOCATE D+1,X                          'flake
51 PRINT"O";
80 NEXT D
90 GOTO 20
```

24B.1

```
 10 REM -------- gosub and return
 12 CLS
 20 REM -------- the first one
 21 GOSUB 200
 30 REM ------- the next one
 31 GOSUB 300
```

```
 40 REM ------- the last one
 41 GOSUB 400
 51 GOSUB 200
 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
340 COLOR 7,2:CLS
350 PRINT"red smoke"
355 PRINT
360 GOSUB 900
380 COLOR 0,7:CLS
399 RETURN
400 REM
401 REM ------ last one
402 REM
450 PRINT"is pouring from your computer!"
455 PRINT
460 GOSUB 900
499 RETURN
900 REM
901 REM ------ timer
902 REM
930 BEEP
950 FOR T=1 TO 400:NEXT T
999 RETURN
```

25.1

```
10 REM ALPHABETICAL
12 CLS
14 PRINT:PRINT:PRINT:PRINT
20 PRINT"This program arranges the letters"
21 PRINT"of a word in alphabetical order."
25 PRINT
30 PRINT"Give me a word."
31 PRINT
32 INPUT W$
33 PRINT
35 L=LEN(W$)
```

```
39 K=1
40 FOR I=97 TO 97+26
41 REM test letters in alphabet
42 REM to see if in word
45 FOR J=1 TO L
50 G=ASC(MID$(W$,J,1))
55 IF G=I THEN H$=H$+CHR$(G):K=K+1
60 NEXT J,I
70 PRINT"Here it is in alphabetical order:"
75 PRINT:PRINT" ";H$
```

25.2

```
10 REM !@#$% double dutch %$#@!
12 CLS
25 PRINT"Give me a sentence:"
26 PRINT
27 INPUT S$
28 PRINT
30 L=LEN(S$)
50 FOR I=1 TO L
51 L$=MID$(S$,I,1)
52 IF L$="a" THEN 72
53 IF L$="e" THEN 72
54 IF L$="i" THEN 72
55 IF L$="o" THEN 72
56 IF L$="u" THEN 72
69 SS$=SS$+L$
72 NEXT I
76 PRINT "Here it is in double dutch"
80 PRINT SS$
```

25 EXTRA

```
10 REM ON . . . GOTO sample
20 REM make a menu
22 CLS
25 PRINT "make your choice:"
27 PRINT
28 PRINT" <a> Take a nap"
30 PRINT" <b> Eat an apple"
32 PRINT" <c> Call a friend"
40 PRINT
42 X$=INKEY$:IF X$="" THEN 42
46 PRINT
48 X=ASC(X$)-96
```

```
50 ON X GOTO 60,70,80
60 PRINT"Your bed is not made!":END
70 PRINT"Your sister ate the last one":END
80 PRINT"Your father is on the phone!":END
332 " < c > Call a friend"
```

26.1

```
10 REM cipher maker
12 CLS
20 PRINT "Code Making Program"
21 PRINT
25 PRINT"Enter a sentence for coding:"
30 INPUT S$
35 L=LEN(S$)
36 S$=S$+" "
40 FOR I=1 TO L STEP 2
45 P$=MID$(S$,I,2)
50 Q$=MID$(P$,2,1)+MID$(P$,1,1)
55 L$=L$+Q$
60 NEXT I
64 PRINT
65 PRINT"Here is the code sentence:"
66 PRINT
70 PRINT " ";L$
```

26.2

```
10 REM Question Answerer
12 CLS
20 PRINT"Enter a question"
22 PRINT
25 INPUT Q$
27 L=LEN(Q$)
28 PRINT
30 REM take off the question mark
32 Q$=MID$(Q$,1,L-1)+" "
36 REM look for the end of the first word
40 FOR I=1 TO L
41 C$=MID$(Q$,I,1)
43 IF C$< >" " THEN 46
44 S1=I:I=L
46 NEXT I
48 REM look for the end of the second word
50 FOR I=S1 + TO L
52 C$=MID$(Q$,I,1)
```

```
53 IF C$< >" " THEN 56
54 S2=I:I=L
56 NEXT I
58 REM turn the words around
60 S$=MID$(Q$,S1+1,S2-S1)
62 V$=MID$(Q$,1,S1)
65 PRINT S$;V$;MID$(Q$,S2+1,L-S2)
```

26.3

```
10 REM Pig Latin
15 CLS
20 PRINT"Pig Latin Program"
25 PRINT
30 PRINT"Give me a word"
31 PRINT
33 INPUT W$
34 L=LEN(W$)
35 PRINT
40 REM Find the first vowel
41 FOR I=1 TO L
42 V$=MID$(W$,I,1)
43 IF V$="a" THEN 50
44 IF V$="e" THEN 50
45 IF V$="i" THEN 50
46 IF V$="o" THEN 50
47 IF V$="u" THEN 50
49 NEXT I
50 IF I=1 THEN L$=W$+"lay":GOTO 80
60 REM found it
68 L$=MID$(W$,I,L-I+1)
70 L$=L$+MID$(W$,1,I-1)
72 L$=L$+"lay"
80 PRINT " ";L$
90 FOR T=1 TO 1000:NEXT T
99 GOTO 15
```

27.1

```
10 REM Backward Added To Forward
15 CLS
20 INPUT"Give me a number";N
30 N$=STR$(N):L=LEN(N$)
40 FOR I=1 TO L
42 B=L-I+1
45 B$=B$+MID$(N$,B,1)
```

```
50 NEXT I
55 B=VAL(B$)
60 PRINT:PRINT" ";N$
61 PRINT" +";      B$
62 L$="-----------"
65 PRINT" ";MID$(L$,1,L+1)
70 A=N+B
72 A$=STR$(A)
75 IF LEN(A$)=L THEN PRINT" ";A:END
80 PRINT"";A
```

27.2

```
10 REM Marching Numbers
12 CLS
20 INPUT " Give me a number";N
22 B$=""
23 CLS
25 N$=STR$(N):L=LEN(N$)
28 N$=MID$(N$,2,L):L=LEN(N$)
40 FOR I=1 TO 40-L
42 LOCATE 12,I:PRINT" "
46 LOCATE 12,I+1:PRINT N$
65 FOR T=1 TO 500:NEXT T
66 N$=MID$(N$,2,L-1)+MID$(N$,1,1)
70 NEXT I
```

28.3

```
 10 REM === ain't got no . . . ===
 12 CLS:PRINT:PRINT:PRINT
 19 REM ----------------get a sentence
 20 PRINT" Enter a sentence":PRINT
 22 PRINT" No punctuation":PRINT
 24 PRINT" Except apostrophy":PRINT
 32 INPUT S$:S$=S$+" "
 35 L=LEN(S$)
 40 REM nn is number of negative words
 41 REM s1 is start of a word
 42 REM s2 is end of a word
 43 NN=0:S1=1:S2=1
 44 REM ---------------- Test words
 45 FOR I=1 TO L
 50 L$=MID$(S$,I,1):A=ASC(L$)
 53 IF A>65 AND A<91 THEN A=A+32
 54 L$=CHR$(A)
```

```
 55 REM ---------------- Is it a space?
 56 IF L$=" " THEN S1=S2:S2=I+1:GOSUB 200
 60 NEXT I
 65 REM ---------------- Print result
 66 PRINT:PRINT
 70 IF NN=0 THEN PRINT" No negative words."
 71 IF NN=1 THEN PRINT" A negative sentence"
 72 IF NN=2 THEN PRINT" Double negative"
 73 IF NN>2 THEN PRINT" Hard to understand"
 80 FOR T=1 TO 2000:NEXT T
 99 GOTO 12
100 REM
101 REM ---------- Test sentences
102 REM
111 REM I don't eat junk food.
112 REM I never eat no junk food
113 REM I don't never eat no junk food.
200 REM
201 REM ---------- Word negative?
202 REM
205 LW=S2-S1-1
210 W$=MID$(S$,S1,LW)
220 READ NW$
222 IF NW$="end" THEN RESTORE:GOTO 299
224 IF W$=NW$ THEN NN=NN+1
230 GOTO 220
299 RETURN
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
914 DATA haven't,hasn't,hadn't
915 DATA wouldn't,couldn't,shouldn't
916 DATA end
```

29.1

```
10 REM menu maker
12 CLS
20 PRINT"which color do you like?"
21 PRINT
22 PRINT"  <y> yellow"
23 PRINT"  <r> red"
24 PRINT"  <b> blue"
26 PRINT
```

```
30 X$=INKEY$:IF X$="" THEN 30
32 IF X$="y" THEN COLOR 6,0:CLS
36 IF X$="r" THEN COLOR 0,2:CLS
37 IF X$="b" THEN COLOR 0,1:CLS
40 GOTO 12
```

29.2

```
10 REM silly sentences
12 CLS
13 PRINT"Silly sentences"
14 PRINT
15 PRINT"Want instructions <y/n>"
16 PRINT
18 GOSUB 200
20 IF Y$="y" THEN GOSUB 100
21 PRINT"The subject: (end with a period)"
22 PRINT
23 GOSUB 300
33 PRINT"The verb: (end with a period)"
34 PRINT
40 GOSUB 300
50 PRINT"The object: (end with a period)"
51 PRINT
52 GOSUB 300
85 PRINT S$
99 END
100 CLS
110 PRINT"Three players enter parts of a sentence":PRINT
115 PRINT"no player can see what the other enters":PRINT
120 PRINT"The first enters the subject":PRINT
121 PRINT" (The person doing something)":PRINT
125 PRINT"The second enters the verb":PRINT
126 PRINT" (The action word)":PRINT
130 PRINT"The third enters the object":PRINT
131 PRINT" (The person or thing to whom":PRINT
132 PRINT"  the action is done)":PRINT
133 PRINT
150 FOR T=1 TO 2000:NEXT T
199 RETURN
200 REM look at keyboard
210 Y$=INKEY$:IF Y$="" THEN 210
299 RETURN
300 REM get a word
310 GOSUB 200
320 IF Y$="." THEN 390
330 S$=S$+Y$
```

```
340 GOTO 310
390 S$=S$+" "
399 RETURN
```

31.1

```
10 REM Arrays
12 DIM D(12)
15 CLS:PRINT:PRINT:PRINT
20 FOR I=1 TO 12
25 READ D:D(I)=D:NEXT I
30 INPUT" Month number <1-12>";M
33 PRINT:PRINT"Month number";M;"has";D(M);"days."
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

32.1

```
10 REM a jillion
12 CLS
15 PRINT:PRINT:PRINT
20 PRINT" Here is a big, big number"
25 PRINT:PRINT
30 FOR I=1 TO 50
31 S=S*1.05
32 REM SOUND S,20
33 FOR J=1 TO 3
35 D=INT(RND(9)*10)+48
36 D$=CHR$(D)
38 FOR T=1 TO 50:NEXT T
40 PRINT D$;
50 NEXT J
55 IF I=50 THEN END
60 PRINT",";
70 NEXT I
```

32.2

```
  1 REM ----- Code--Decode -----
  2 GOTO 1000
100 REM
101 REM ----- Main Loop
102 REM
110 GOSUB 400'        get password
```

```
115 PRINT:PRINT" code or decode <c/d>?"
116 Y$=INKEY$:IF Y$="" THEN 116
121 IF Y$="c" THEN 500' code
130 IF Y$="d" THEN 600' decode
140 GOTO 115
199 END
400 REM
401 REM ----- get password
402 REM
403 REM ----- form cipher alphabet
404 PRINT:PRINT:PRINT
405 INPUT " input password";PW$
406 REM ----- remove repeated letters
408 F$=LEFT$(PW$,1)
410 FOR I=2 TO LEN(PW$):L1$=MID$(PW$,I,1)
412 FOR J=1 TO LEN(F$):L2$=MID$(F$,J,1)
420 IF L1$=L2$ THEN 430
421 NEXT J:F$=F$+L1$
430 NEXT I:PW$=F$
433 PRINT:PRINT" the shortened password is "
435 PRINT:PRINT" ";PW$
439 REM remove password letters from alphabet
440 FOR J=1 TO LEN(PW$):L2$=MID$(PW$,J,1)
441 IF L2$=LEFT$(A$,1) THEN A$=MID$(A$,2):GOTO 460
442 FOR I=1 TO LEN(A$):L1$=MID$(A$,I,1)
445 IF L1$=L2$ THEN A$=LEFT$(A$,I-1)+MID$(A$,I+1)
455 NEXT I
460 NEXT J
461 REM
462 REM ----- form cipher alphabet
463 REM
465 A$=PW$+A$
470 PRINT:PRINT" alphabets:"TAB(23);"plain"
471 PRINT:PRINT" ";B$
475 PRINT " ";A$
480 PRINT:PRINT TAB(21);" cipher"
499 RETURN
500 REM
505 PRINT:PRINT" input message":PRINT
506 L$=INKEY$:IF L$="" THEN 506
510 L=ASC(L$):IF L =13 THEN 590
520 IF L<96 OR L>123 THEN P$=P$+L$:GOTO 540
530 P$=P$+MID$(A$,L-96,1)
540 PRINT L$;
589 GOTO 506
590 PRINT:PRINT P$
599 END
600 REM
```

```
610 PRINT:PRINT" type in the coded message":PRINT
615 L$=INKEY$:IF L$="" THEN 615
616 L=ASC(L$):IF L=13 THEN 699
620 FOR I=1 TO 26
625 IF L$=MID$(A$,I,1) THEN PRINT MID$(B$,I,1);:GOTO 615
630 NEXT I
635 PRINT L$;
640 GOTO 615
699 END
1000 REM
1001 REM ----- starting stuff
1002 REM
1010 CLS
2010 A$="abcdefghijklmnopqrstuvwxyz"
2020 B$=A$
2999 GOTO 100
```

INDEX OF COMMANDS AND FUNCTIONS EXPLAINED IN THIS BOOK

INDEX OF KEYS USED IN THIS BOOK

INDEX OF TOPICS

```
CLS
COLOR 0,1,2
PRINT "RICHARD"
BEEP
FOR I=1 TO 2000: NEXT I
COLOR 0,1,3
PRINT "FRANK"
BEEP
FOR I=1 TO 2000: NEXT I
COLOR 0,1,4
PRINT "CLOTHIER"
BEEP
```

KIDS AND THE IBM-PC/PCjr

A KID'S BOOK FOR ADULTS TOO? YOU BET!

The title of this book may be KIDS & THE IBM-PC/PC*jr*, but don't let it fool you. Designed for children ages 10 to 14, it also provides valuable information for the adult purchasing his or her first IBM Personal Computer. KIDS & THE IBM-PC/PC*jr* was created to lead you gently, yet quickly, into the world of IBM BASIC.

You'll learn how to write action games, board games and word games. Guidance, explanation, exercises, reviews and quizzes are given in an easy going, non-technical style. Study guides precede each chapter, and every new concept is tied to simple everyday experiences. KIDS & THE IBM-PC/PC*jr* is illustrated with dozens of clever cartoons — so you'll laugh as you learn!

Computers are here to stay. KIDS & THE IBM-PC/PC*jr* prepares the computer generation by solving the mysteries of the computer in entertaining and enjoyable ways.

ABOUT THE AUTHOR: Edward H. Carlson has been a Professor of Physics and Astronomy at Michigan State University since 1965. His interest in computers began in 1960, and he has since been involved in numerous University computer projects. He recently helped establish a Computer Camp for children.

OTHER POPULAR COMPUTER BOOKS BY DATAMOST:

Kids & the Apple
Kids & the Atari
Kids & the Commodore 64
Kids & the IBM-PC/PC*jr*
Kids & the Panasonic
Kids & the TI-99/4A
Kids & the VIC-20
by Ed Carlson

How to Write an Apple Program
How to Write an IBM-PC Program
How to Write a TRS-80 Program
How to Write a Program Vol. II
A Computer in Your Pocket
by Ed Faulk

Games Apples Play
Games Ataris Play
by Mike Weinstock & Mark Capella

The Elementary Apple
The Elementary Atari
The Elementary Commodore 64
The Elementary IBM-PC
The Elementary Timex/Sinclair
The Elementary VIC-20
The Elementary TI-99/4A
by William Sanders

Computer Playground Apple II, II+, //e
Computer Playground Atari 400/800/1200
Computer Playground Commodore 64/VIC-20
Computer Playground TI-99/4A
by M.J. Winter

Using 6502 Assembly Language
p-Source
by Randy Hyde

ISBN 0-88190-265-9

DATAMOST INC™

8943 Fullbright Avenue, Chatsworth, CA 91311-2750
(818) 709-1202