

Revised

COMPUTE!'s Kids and the IBM PC & PCjr



Edward H. Carlson

Illustrated by Paul D. Trap

An entertaining, easy-to-use guide to learning
BASIC on the IBM PC or PCjr. For kids, both
young and old.

A **COMPUTE!** Books Publication

\$12.95

COMPUTE!'s Kids and the IBM



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

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

Copyright 1985, COMPUTE! Publications, Inc. All rights reserved

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

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-942386-93-0

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. PC and PCjr are trademarks of International Business Machines, Inc.

Contents

Before We Begin

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

Introduction

1. NEW, PRINT, REM, and RUN	1
2. Beeps and Strings	9
3. LIST and REM	14
4. The Keyboard and Editing	22
5. The INPUT Statement	29
6. Tricks with PRINT	35
7. The LET Statement	41
8. GOTO and the Break Key	46
9. The IF Statement	52
10. Introducing Numbers	58
11. TAB and Delay Loops	65
12. The IF Statement with Numbers	71
13. Random Numbers and the INT Function	78
14. Saving to Disk	86

Graphics, Games, and All That

15. Some Shortcuts	93
16. Moving Graphics and LOCATE	101
17. FOR-NEXT Loops	107
18. DATA, READ, and RESTORE	114
19. SOUND	121
20. COLOR	125
21. Drawing Pictures	131
22. Color Graphics	137
23. Music	143
24. Pretty Programs, GOSUB, and RETURN	149

Advanced Programming

25. ASCII Code and ON-GOTO	156
26. Snipping Strings: LEFT\$, MID\$, RIGHT\$, and LEN	162

27. Switching Numbers with Strings	168
28. Logic: AND, OR, and NOT	174
29. Secret Writing and INKEY\$	182
30. Long Programs	188
31. Arrays and the DIM Statement	192
32. User-Friendly Programs	200
33. Debugging: STOP and CONT	208

Appendices

A. Disk Use	215
B. IBM BASIC Reserved Words	219
C. Answers to Selected Assignments	221
D. Glossary	241
Topical Index	253
Command and Function Index	255

Acknowledgments

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

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

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

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 solutions to assignments to run on this machine.

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

COMPUTE! Publications started publishing *COMPUTE!* magazine at about the same time that I started writing articles on home computing. I am grateful that COMPUTE! encouraged my writing career by accepting some of my early articles. This encouragement continues with the publishing of the books from the Kids and ... series. I greatly appreciate the skill and energy of the COMPUTE! staff in editing and assembling this revision.

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

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

My final and heartfelt thanks go to my wife, Louise. As absorbed in professional duties as I, she nevertheless took on an increased share of family duties as the book demanded 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. It's for you.

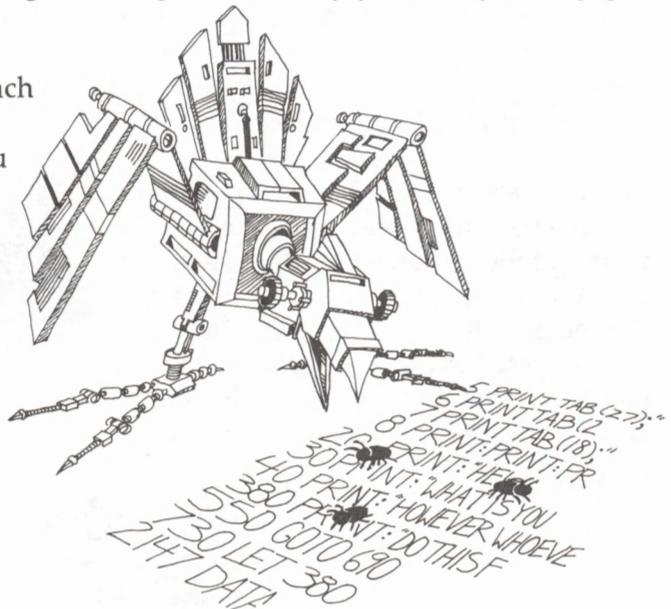
You will learn how to make your own action games, board games, and word games. You may entertain your friends with challenging games and provide some silly moments at your parties with the short games you invent.

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

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

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

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



To the Parents

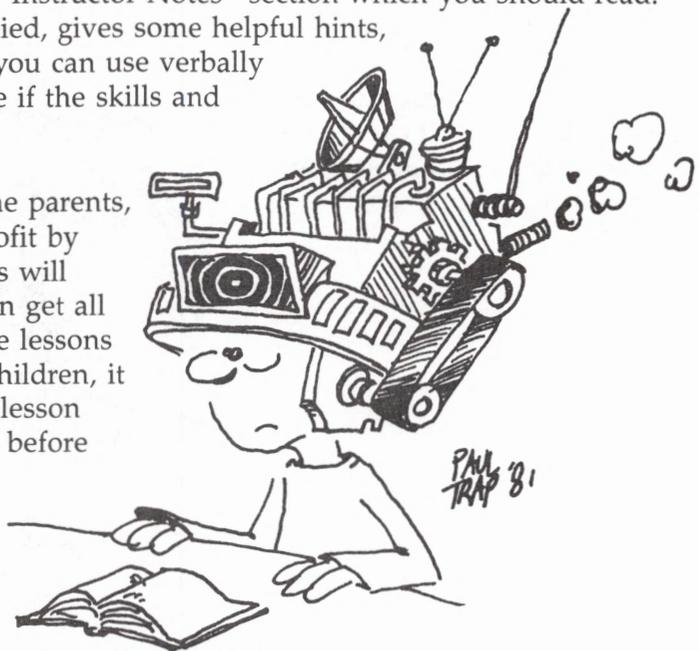
This book is designed to teach BASIC on the IBM to youngsters from 10 to 14 years old. It gives guidance, explanations, exercises, reviews, and quizzes. Some exercises have room for the student to write in answers that you can check later. Answers for program assignments are given in the back of the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

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

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

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

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



To the Teacher

This book is designed for students in about the fifth grade through the ninth 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 give good review questions.

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

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

Each concept is tied to the student's everyday experiences—through the language and examples used to express the idea and through the cartoons. Thus, metaphor makes the new material familiar to the student.



About Programming

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

Like a block set that has many copies of a few types of blocks, BASIC uses a relatively small number of standard commands. Yet the blocks can be formed into 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 thoughts on several scales, from the overall topic to lead and summary paragraphs and sentences, on down to grammar and punctuation in sentences and spelling of individual words.

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

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

The analytic and synthetic skills learned in programming can be transferred to more general situations and can help a child develop a more mature style of thinking and working.

About the Book

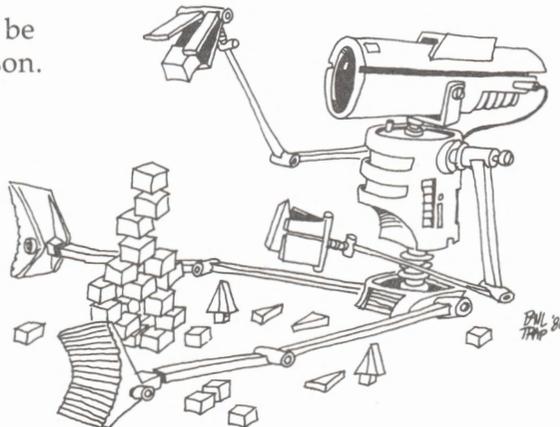
This book is written for the IBM PC and PCjr computers. Most of the book describes features present in Cassette BASIC, which is functionally equivalent in the two machines. Some commands, such as those for music and graphics, belong to the Advanced BASIC of the PC and the Cartridge BASIC of the PCjr.

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

The book starts with a bare-bones introduction to programming, leading quickly to the point where interesting programs can be written. See the notes for lesson 5, "The INPUT Statement," for an explanation. The central part of the book emphasizes more advanced and powerful commands. The final part of the book builds on this foundation, but also deals with such broader aspects of the art of programming 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 suggested solutions for some of the 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

This lesson is an introduction to the computer. Pull up a chair and help your student get started. Help the student understand that you cannot damage the computer by what you type in. If something goes wrong and all else fails, turn the computer off, then turn it on and start over again.

If your screen is hard to read (with a TV or some monitors in 80-column mode), enter

```
mode 40 to the A> prompt, or
screen 0,1 if you are in BASIC
```

Prepare a disk for the student's exclusive use (see Appendix A "Disk Use"). Lesson 1 gives start-up instructions for Advanced BASIC (Cartridge BASIC on the PCjr). If you want different start-up instructions, provide them for the student.

Help your student find the Ctrl, Enter, Home, Shift, and quotes keys. (The PCjr keyboard is color coded. If you need to use any function that is printed in green, first press the Fn key.)

Here are the contents of the first lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines; the Enter key.
3. 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.
7. Memory can be cleared with NEW.
8. What is on the screen and what is in memory are different. This may be a hard concept for the student to understand at first.
9. RUN makes the computer go to memory, look at the commands and statements in the program, and execute them.
10. You can skip numbers in choosing line numbers, and why you may want to.

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 the program.
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 drive(s) and take out any disks, put them in their envelopes, and put them safely away. If you have a PCjr, make sure Cartridge BASIC is in place and turn the computer on. For either computer, put your student disk in the disk drive (drive A if you have two drives). Find the switch on 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 part is Ok. Below the message 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.

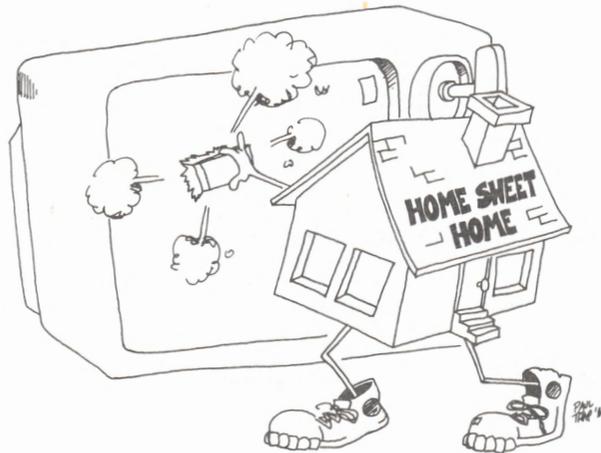
Cursor means "runner." The little line runs along the screen showing where the next letter you type will appear.

Typing

Type some things. What you type shows on the TV screen. Press the Home key. (On the PCjr, press the Fn key and then press the key which has Home printed on it in green.) The cursor jumps to its home in the top 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.



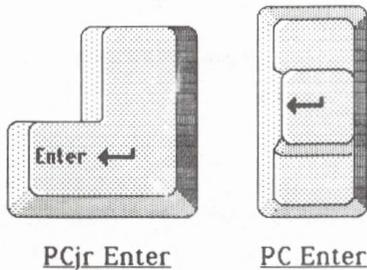
Find the Ctrl key. It is beside the A key on the left side of the keyboard. Hold down the Ctrl key and press the Home key. (PCjr: Don't forget to first press the Fn key.) That's how easy it is to erase the screen.

Command the Computer

Try entering this.

Type: give me candy

and press the Enter key. The Enter key is on the right side of the keyboard. It has a bent arrow on it like this:



If you make a mistake, clear the screen as described above and start over.

The computer will print

```
Syntax error
Ok
```

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

Surprisingly, the computer understands only about 178 words in BASIC. You need to learn which words the computer understands.

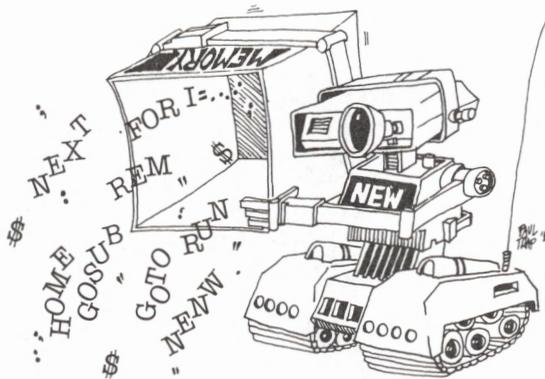
Here are the first four words to learn:

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

The NEW Command

Type: `new`

and press the Enter key. NEW empties the computer's memory so that you can put your program there.



How to Enter a Line

When this book tells you to *enter* something, it means 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 quotation marks, find one of the two Shift keys. Each has a fat open arrow pointing upward on it. Hold down a Shift key and press the key that has the " on it. It's two keys to the right of the L key.

Don't forget to press the Enter key at the end of the line.

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

What Is a Program?

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



How to Run a Program

A moment ago you put this program in memory:

```
10 PRINT "hi"
```

Enter: run

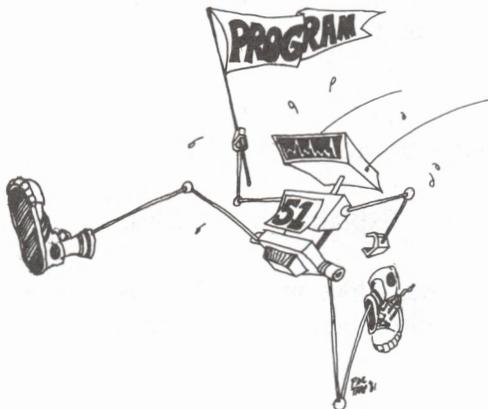
(Did you remember to press the Enter key?)

Make sure you do not confuse the number 0 with the letter O. Most computers display zero with a line drawn through it and the letter without the line.

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

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

```
hi  
Ok
```



How to Number the Lines in a Program

Clear the screen. Use 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 an instruction. You have already learned the PRINT statement. You will learn about the REM statement later.

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 statements in the lines. It starts with the lowest line number and goes down the list in order.

The REM Statement

The REM statement is for writing little notes to yourself. The computer ignores the notes. Use REM for putting the name of your program in the top line of the program. See lesson 3 for more details.

Assignment 1

1. Use two keys to erase the screen.
2. Use the NEW command. Explain what it does.
3. Write a program that uses REM once and PRINT twice. Then use the command RUN to make the program obey the commands.

Instructor Notes 2. Beeps and Strings

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

The CLS statement used in a program clears the screen and homes the cursor, just as the Ctrl-Home keys do from the keyboard (Ctrl-Fn-Home on the PCjr).

The COLOR statement changes both foreground and background colors of the characters printed on the screen. The third number in the COLOR statement changes the border color in the screen 0 mode.

If you are using a TV or certain monitors, you should enter

```
screen 0,1
```

before using COLOR. You can do this from the keyboard or you can put it in a line in your programs.

The idea of a *string constant* 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 Ctrl-Home (hold down Ctrl and press the Home key). On the PCjr, hold down Ctrl, press Fn, and then press the Home key.

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

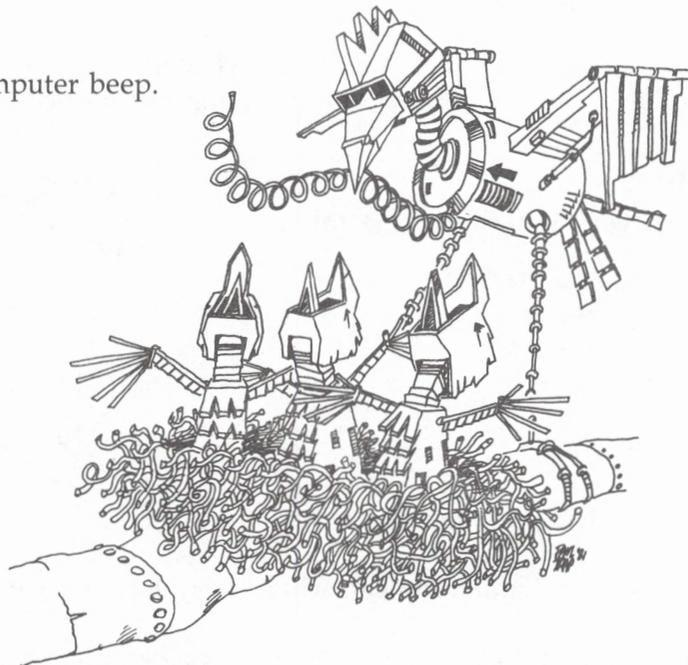
The Computer Beeps Like a Bird

Enter this program:

```
10 REM bird
20 CLS
25 PRINT "----0----"
30 BEEP
```

Enter: run

Line 30 makes the computer beep.



Clearing the Screen in a Program

The CLS statement clears the screen.

Run this:

```
10 REM wipe that smile off
20 PRINT "Smile"
30 BEEP
40 CLS
50 PRINT "That is better"
```

The above program prints Smile, then clears the screen quickly.

Printing an Empty Line

Run this:

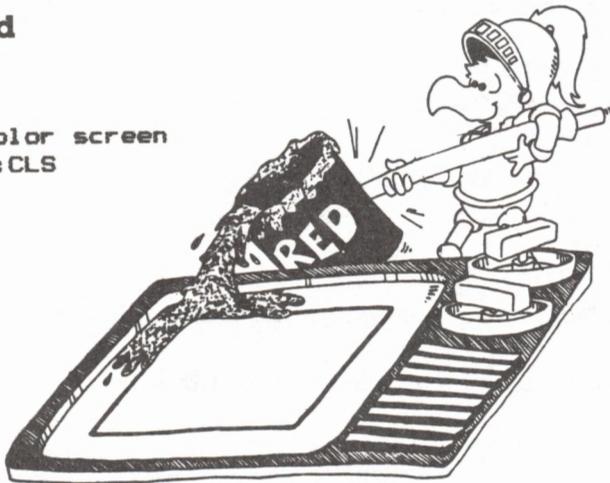
```
10 REM skipping
15 CLS
20 PRINT "here is the first line"
30 PRINT
40 PRINT "one line was skipped"
```

Line 30 just prints a blank line.

Color the Screen Red

Run this:

```
10 REM red color screen
20 COLOR 0,4:CLS
```



If your colors are smeared, try adding this line:

```
15 SCREEN 0,1
```

To get other colors, try other numbers from 1 to 7 in place of 4 and 0 in the COLOR statement.

Put a Frame Around the Screen

Enter line 20 again to look like this:

```
10 REM border  
20 COLOR 0,4,3:CLS
```

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

Back to Black and White

```
Enter: color 7,0  
      cls
```

(PCjr: Enter: screen 0)

String Constants

Look at these PRINT statements:

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

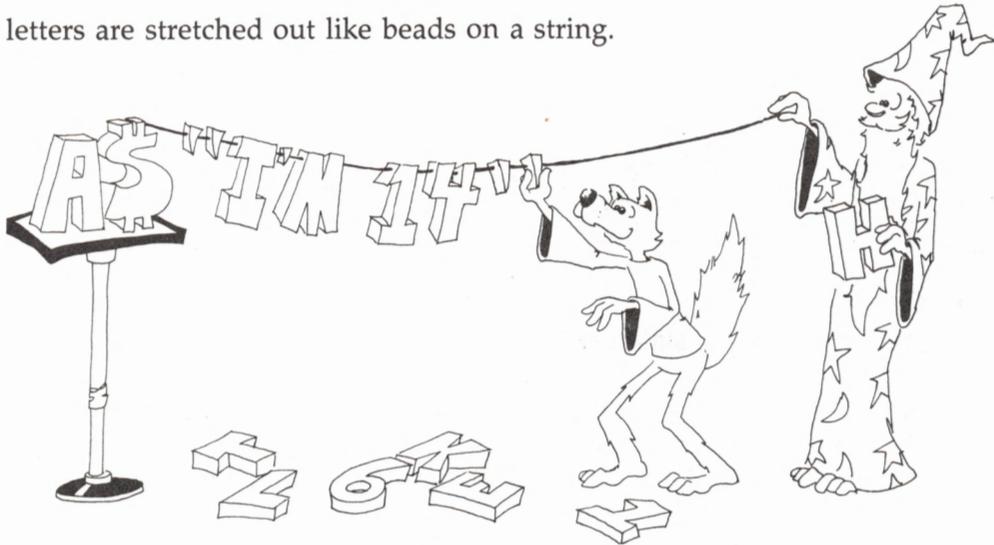
Letters, numbers, and punctuation marks are called *characters*.

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 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 with black letters printed on blue squares.
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 REM

In this lesson we will cover:

1. LIST, LIST 30.
2. Memory boxes holding lines.
3. Erasing one line from memory.
4. Adding a line between old lines.
5. Replacing a line.
6. REM for titles, remarks.
7. Drawings using PRINT statements.

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 prints the program to the screen. The special uses like LIST 100–300 and LIST –300 will be taken up later.

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 statement can be a little confusing to new students. It needs to be distinguished both from PRINT and from just typing to the screen. The use of PRINT to draw pictures is demonstrated. It will be better for the student to draw some at the end of each lesson than to do a lot now. After lesson 4, drawing helps develop line-editing skills.

Questions

1. How do you erase a line you no longer want?
2. 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 REM

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

Now enter these lines:

```
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.

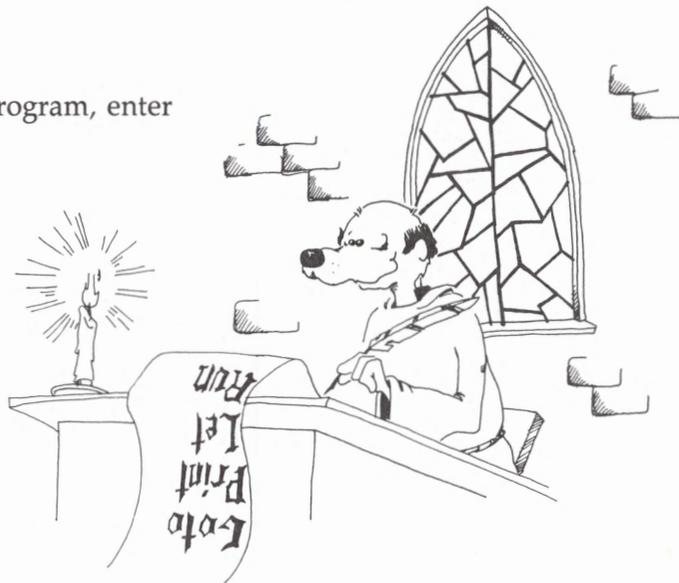
Listing the Program

To show the whole stored program, enter

```
list
```

To show line 30 of the program, enter

```
list 30
```

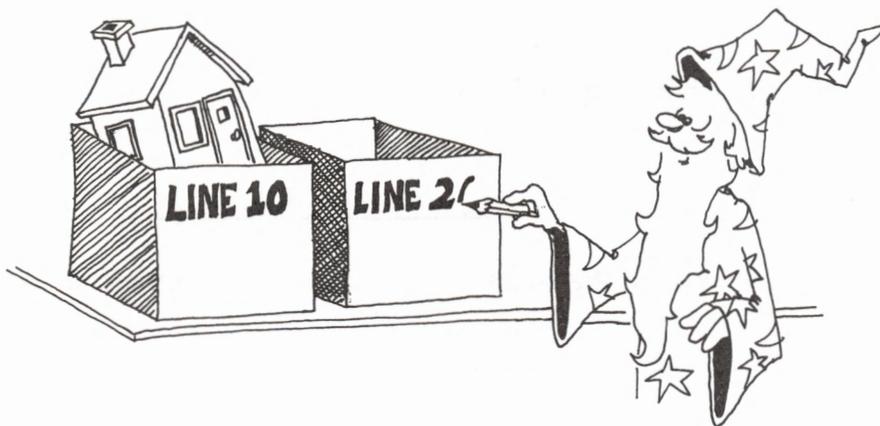


Capital Letters

The computer lists commands and statements in capital letters, even when you type in lowercase letters. In this book, we will often use capital letters for commands and statements (like LIST, PRINT, REM, RUN, and NEW), but you can type in lowercase letters if you want.

Memory

The computer's memory is like a shelf of boxes. On the front of each box is its name. At the beginning, all the imaginary 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 in 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" in the box and put that box in its place on the imaginary 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

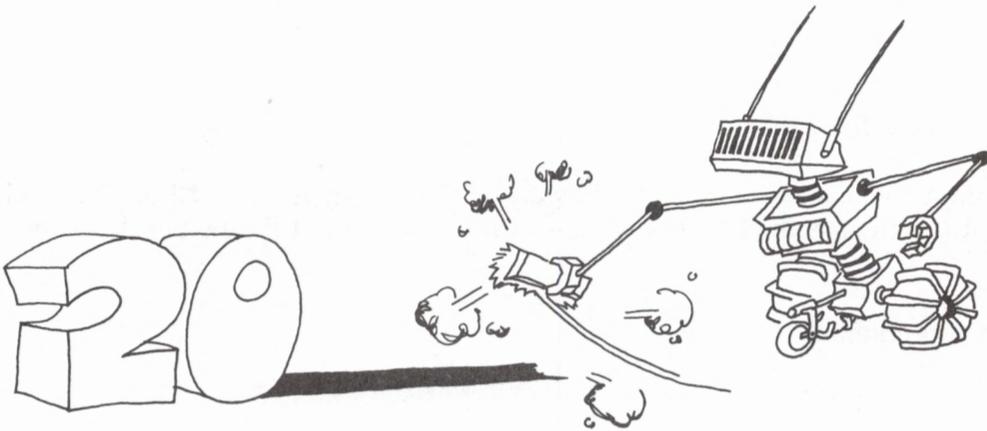
20

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

When you enter just a line number with nothing after it, the computer finds the box with that line number on 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 the numbers of the old lines and type your line in. The computer puts it in the correct place.

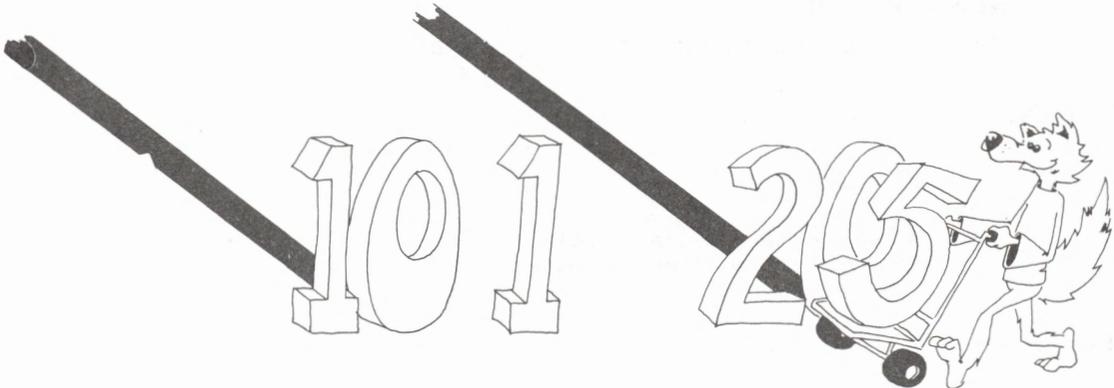
Enter NEW and this:

```
10 REM 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.

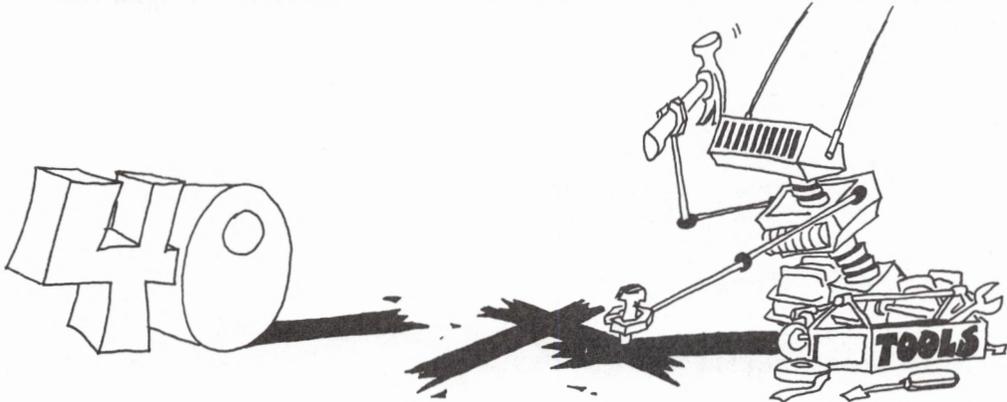


Fixing a Line

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

```
40 PRINT "needs fixing"
```

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



The REM Statement

Use a REM statement 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 it.

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 you or anyone else understand the program.

Picture Drawing

You can use the PRINT statement 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 "xxxxxxxxxxxxx"
50 PRINT "  o       o  "
```

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

Assignment 3

1. What command will list line 10 of the program?
2. How do you tell the computer to list the whole program on the screen?
3. What does the computer do (if anything) when it sees the REM statement? What is the REM statement used for?
4. Add a COLOR statement to the car picture program so that a blue car is drawn on a red background.
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 Backspace key, and the Delete key.



PCjr Backspace PC Backspace

The arrow keys are used to move 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 other characters or deleted.

There are two deletion modes: Backspace 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'll 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 down any key for a short time starts the auto-repeat feature of the keyboard. This is 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 techniques:

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

Using the Backspace key.

Using the Delete key. Explain how the Backspace key and the Delete key differ.

Using the repeat feature of the keyboard.

Lesson 4. The Keyboard and Editing

The Cursor Is a Flashing Line

As you remember, 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.

These keys move the cursor. Press one of the arrows. 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.)

Now do this: Use the cursor arrow keys to move the cursor to the middle of the screen and 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'll see

hhhhhhhhhhhhhhhhhhhhhhhhhhhhhh



Now try this:

Clear the screen.

Move the cursor to the middle of the screen.

Type your name there.

Put a box of "stars" around your name.

Fixing Messed-Up Lines

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

Enter:

```
10 REM zragon
```

Oops! We want dragon.

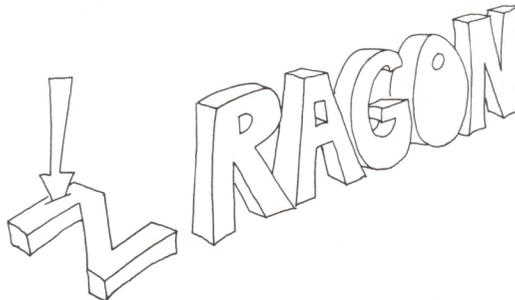
Use the arrow keys to move the cursor onto the z.

Type a *d* instead.

Now the line is correct:

```
10 REM dragon
```

Press Enter to store the correct line in the memory.



Erasing Letters

There are two erase keys. One at the top right of the keyboard has an arrow on it which points to the left.

We call it the Backspace key. It erases a mistake by causing the cursor to go to the left (backward) on the screen and erase whatever was there before.

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

Enter:

```
10 REM aaaaaaaaaaaaaaEiiiiiiiiiiiiii
```



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!

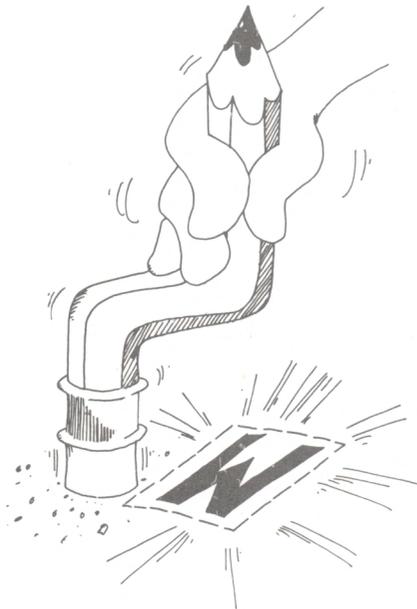
Enter:

```
10 REM aaaaaaaaaaaaaaWiiiiiiiiiiiiii
```

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

Now hold the Backspace key down. The cursor goes whizzing off to the left, erasing all the *a* letters and dragging the rest of the line with it.

Note: The Backspace 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 eats up letters from the right.

Backspace erases what is next to it on the left and then goes whizzing along to the left, erasing as it goes.

The CLS Statement in a Program Line

Run:

```
10 REM vanishing junk
12 CLS
20 PRINT
22 PRINT "Nice clean screen!"
```

The CLS statement clears the screen and puts the cursor in the home spot on the screen.

Assignment 4

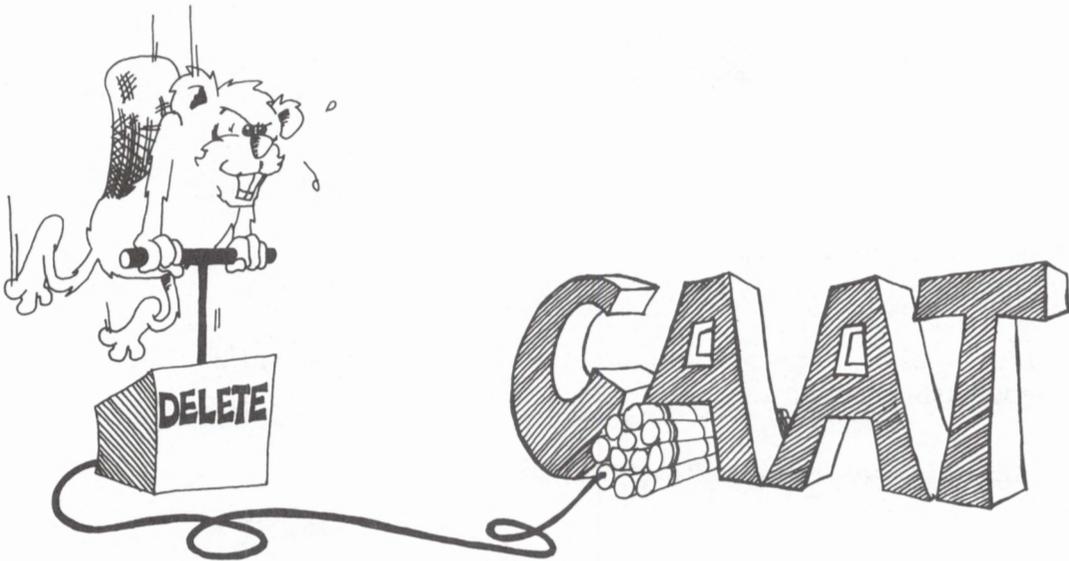
1. Type a line and use the arrow keys to move around in the line. Change letters in the line. When you are done, press the Enter key to enter the line into memory.

2. Fix 10 REM caaat to read 10 REM cat.

First, fix it by using the Del key. Then write it again and fix it using the Back-space 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 statement and string variables.

We introduce the input statement in its simplest form without a message in quotes in front of the string identifier:

```
INPUT A$
```

This allows the student to concentrate on the central feature of INPUT.

Similarly, we will give only the essential feature of each statement throughout the introduction to the book (through lesson 14). We want the student to see the forest before worrying about the trees. These are the statements and functions required for interesting programs:

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

String variables are introduced by 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 we discuss the RND function.

The two hats of the student, those of computer programmer and computer user, cause much confusion as the student works the assignments. PRINT is the programmer speaking, while the user can speak only when invited by an INPUT statement put there by the programmer.

Questions

1. What two different things does the computer put in boxes (one at programming time and one from an INPUT)?
2. How does the program ask 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 "?"
```

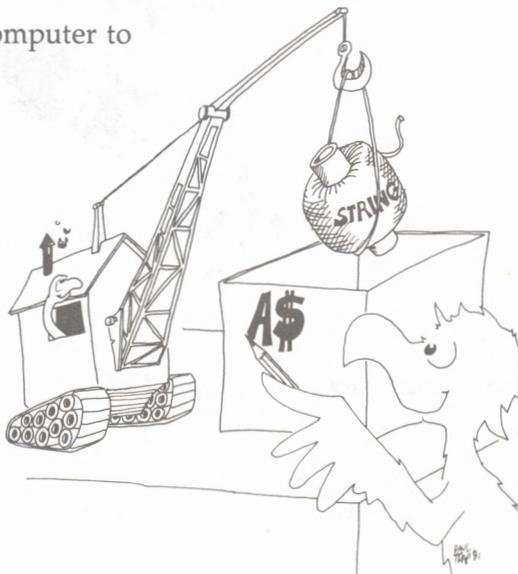
Run it. When you see a message followed by a question mark, type hi and press the Enter key.

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

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 in. The name is written on the front of the box and the string is put inside the box.

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

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

The box can hold only one string at a time.

Putting a new string in a box 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 in the string you type in answer to the computer.

If you accidentally do put a comma in, the computer will give this answer and then will wait:

?Redo from start

This means that the computer wants you to try again, but don't put any commas in 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 both programmer and user.



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

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

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

Assignment 5

1. Write a program that asks for the user's name and then says something silly to the user by name.
2. Write a program that asks you to INPUT your favorite color and put it in a box (a string) called C\$. Next, have the program ask your favorite animal and put this in 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 we will look at:

1. PRINT with a semicolon at the end.
2. PRINT with semicolons between items.
3. The *invisible* print cursor.

We won'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 statement. (The input cursor is the flashing line. It is familiar from the edit mode and the INPUT statement.)

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 statement.

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

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

The series of printed items will have their characters in contact. If spaces are desired, as in the "toast and jam" example, the spaces have to be put in the strings explicitly.

Questions

1. Which cursor is a little flashing line? What instruction puts it on the screen?
 2. Which cursor is invisible? What instruction 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* in *dg* to make *dog*.

Lesson 6. Tricks with PRINT

One Line or Many?

Enter this program and run it:

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

Each PRINT statement prints a separate line.

Now enter:

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

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

Run it.

What was different from the first time?

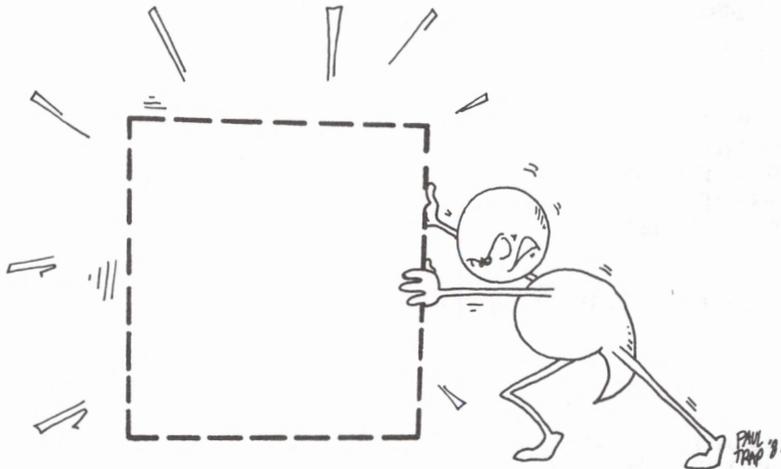
The Hidden Cursor

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

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



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



Famous Pairs

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 " in line 80.

Squashed Together or Spread Out?

Enter NEW and then try this:

```
10 PRINT "rock";"and";"roll"
```

After you have run it, try this 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: using the Backspace key and the Del (Delete) key.

The opposite of erase is *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 fixing 10 REM dn to be 10 REM dragon.

Rule: You have to press Ins only once. Then you can type as many letters as you want.

To stop the Ins thing, just move the cursor with any of the arrow keys. Try fixing 10 REM dn to be 10 REM dragon smoke.

How? First, put the cursor on the *n*, press Ins, and add *rago*. Then use the right arrow key to move the cursor to the space after *dragon* and type *smoke* (with a space before the *s*).

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 statement, print the group name and the tune name with the word *plays* in between.
2. Do the same, but use three PRINT statements that will print it all on one line.
3. Type these lines and then fix them by 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, again using the concept of memory boxes. Concatenation using the plus sign (+) is called "gluing the strings."

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

The box idea nicely separates the concepts *name* of the variable and *value* of the variable. *The name is on the label of the box, while the value is inside.* The contents of the box may be removed for use, and new contents may be 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.

So far we have used:

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

These are the special keys discussed so far:

Enter, Shift, Ctrl, Backspace, Del, Ins, Num Lock, and the four cursor keys.

Questions

1. LET puts things in boxes. So does INPUT. How are they different?
2. In this program, what is "Mom" called?

```
10 Q$="Mom"
```

What is the name of the string variable? What is the value of the string variable after the program runs?

3. If you run this little program, what is in each box after the program runs?

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

Lesson 7. The LET Statement

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

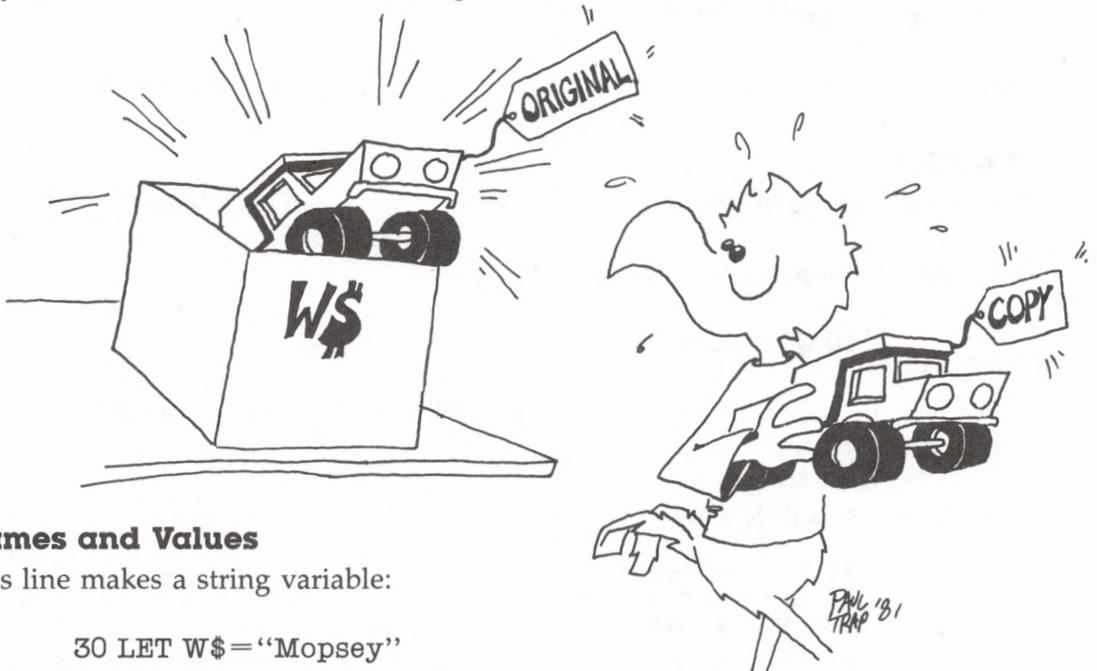
```
10 CLS
20 LET Q$="truck"
40 PRINT Q$
```

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 and writes Q\$ on the front and then puts the string "truck" in 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

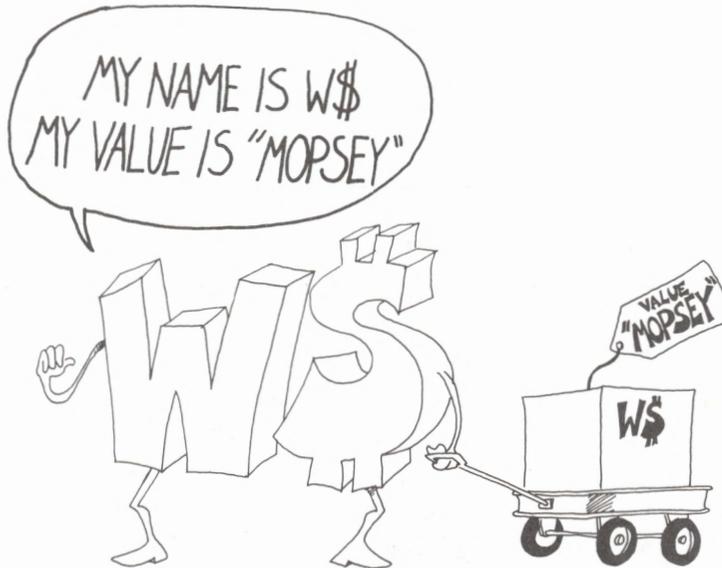
This line makes a string variable:

```
30 LET W$="Mopsey"
```

The name of the variable is W\$.

The value of the variable is put in the box.

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



Another Example

Enter and run:

```
10 LET D$="pickles"  
20 LET A$=" and "  
30 PRINT "what goes with pickles?"  
35 INPUT Z$  
40 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 plus sign (+) sticks two strings together like glue.

Assignment 7

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

Instructor Notes 8. GOTO and the Break Key

The GOTO statement allows a *dumb loop* that goes on forever. It also helps control the flow of command in later programs, after 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 for a reasonable length of time. In each loop through, something can be modified.

The problem is how to stop it. The Ctrl-Break (PC) or Fn-Break (PCjr) keys do this nicely.

GOTO is tolerant of *spaghetti* programming. Examples of spaghetti are shown to the students and although they will have some fun with them, the idea is to make students 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. Still lacking is IF, which will change the computer from a mere record player into a machine that can evaluate situations and make decisions accordingly.

Questions

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

```
10 PRINT "hi"  
20 GOTO 40  
30 PRINT "big"  
40 PRINT "daddy"
```

2. What about 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 your favorite movie star's name, and then does it over and over again.

Lesson 8. GOTO and the 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 the program. It never stops by itself! To stop your name from whizzing past your eyes, try this:

On the PC, hold down the Ctrl key and press the Break key.

On the PCjr, press the Fn key, then the Break key.

From now on, we will just say *press the Break key* when we mean to press the two keys.

Line 40 uses the GOTO statement. It is like "Go to Jail" in the game 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 many programs.



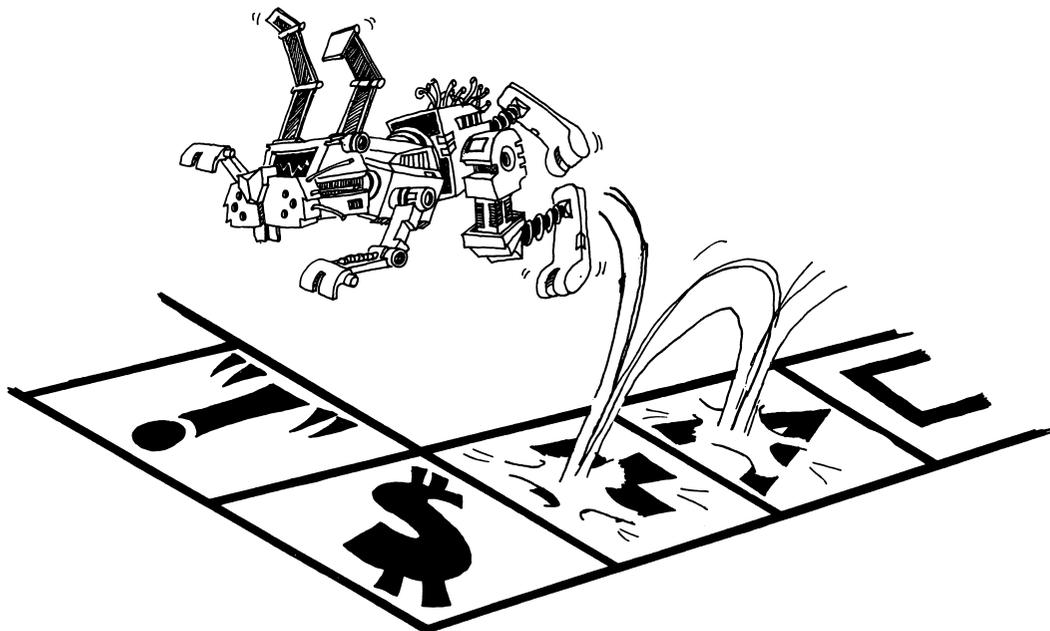
More Jumping

Enter:

```
10 REM quiet
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 question mark (?) and the flashing cursor. Press the Break key to end the program.

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



Kinds of Jumps

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

Jumping back gives a loop.

```
10 PRINT "Hi"
20 GOTO 10
```

The path through the program is like this:



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

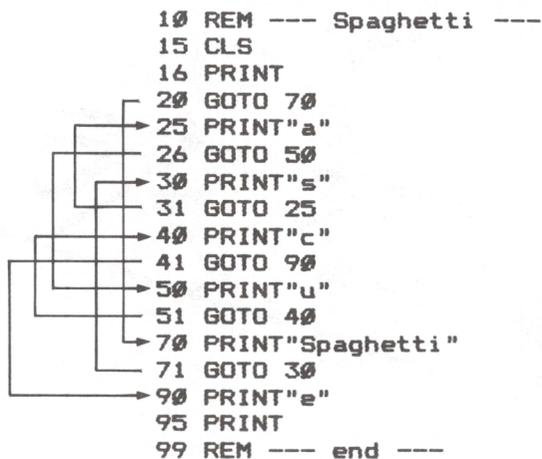
Jumping ahead lets you skip part of the program. It is not useful yet, but we will use it later in the IF statement.

The Break Key

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

A Can of Spaghetti

Look at this:



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 statement, draw the road map for this spaghetti program:

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

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 Break key.
6. Write a program that uses each of these statements:

CLS, BEEP, PRINT, INPUT, LET, GOTO

It also should glue two strings together.

Instructor Notes 9. The IF Statement

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

The GOTO statement 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 the assertion being tested for truth. *Statement C* is the statement to be performed if the assertion is true.

Two levels of abstraction occur in the assertions. One level is the comparison of two values to determine if they are equal. The other level is the judgment of whether an assertion is true or false.

Some care is needed to separate and clarify these notions.

When you see $A = B$, it may not be true that A equals B, because the assertion may 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 answer, print Yummy! For the V answer, print Mmmmmm!
 3. What is meant by phrase A. By statement 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 do statement C
```

The computer looks at phrase A.

If it is true, the computer does statement C.

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

```
10 IF phrase A is false THEN skip statement C and go on to the next line.
```

The IF in English and BASIC

In English:

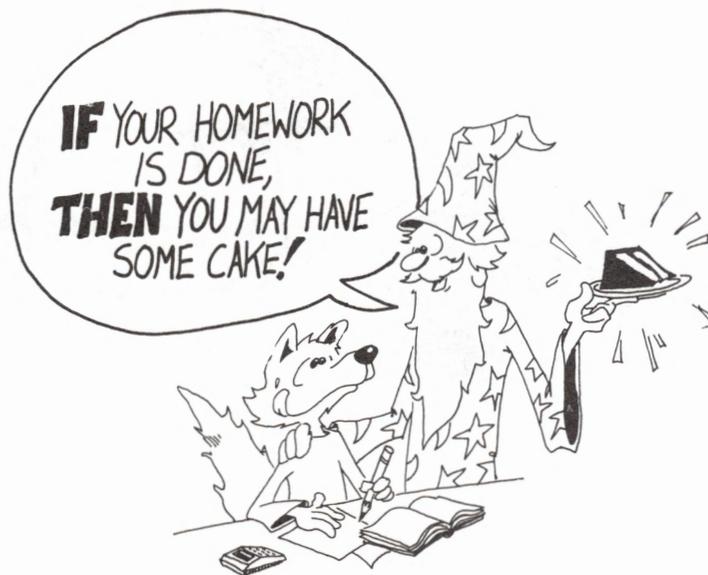
If your home work is done, then you may have some cake.

In BASIC:

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

Assignment 9A

Clear memory and write a program that asks if you like football or baseball better. If the answer is baseball, the program should print Play Ball! If the answer is football, the computer should beep and print Kickoff!



A Fork in the Road

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

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

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

Put it in an IF statement:

```
40 IF B$<>"fire" THEN PRINT "no smoke"
```

If the B\$ box contains anything but fire, then B\$ is not equal to fire, so the expression B\$<>"fire" is true. The computer prints no smoke.

But if the B\$ box contains fire, then the phrase B\$<>"fire" is false, so the computer will not print anything. Here is how it looks in a program:

```
10 PRINT "fire or ice?"
20 PRINT "enter 'fire' or 'ice'"
30 INPUT B$
40 IF B$<>"fire" THEN PRINT "no smoke"
50 IF B$="fire" THEN PRINT "hot"
```



Assignment 9B

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

```
G$<>C$
```

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

Instructor Notes 10. Introducing Numbers

This lesson introduces numeric variables and operations and revisits LET, INPUT, and PRINT statements.

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

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

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

A mixture of string and numeric values can be printed with PRINT.

The nonstandard use of = in BASIC—that it means *replace* and not *equal*—shows up clearly in this statement:

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

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 statement. 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. Do the same for string variables.

Lesson 10. Introducing Numbers

INPUT, LET, and PRINT

So far we have used only 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

The plus sign (+) indicates addition. Hold down the Shift key and press the = key to get +.

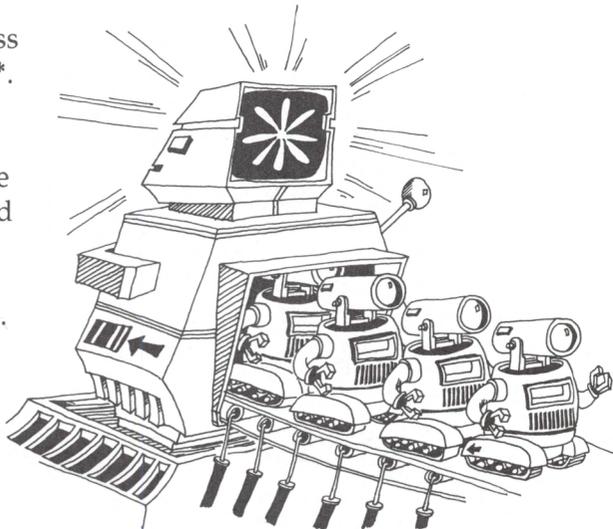
The minus sign (−) is for subtraction.

The / sign means division.

The * sign is for multiplication. Press Shift and the 8 key together to get *.

Computers use * instead of \times for a multiplication sign. Try this: Change line 40 above so that N is multiplied by 5.

Computers use / for a division sign. Answers are printed with decimals. Try this: Change line 40 so that N is divided by 5. What do you say in line 50?

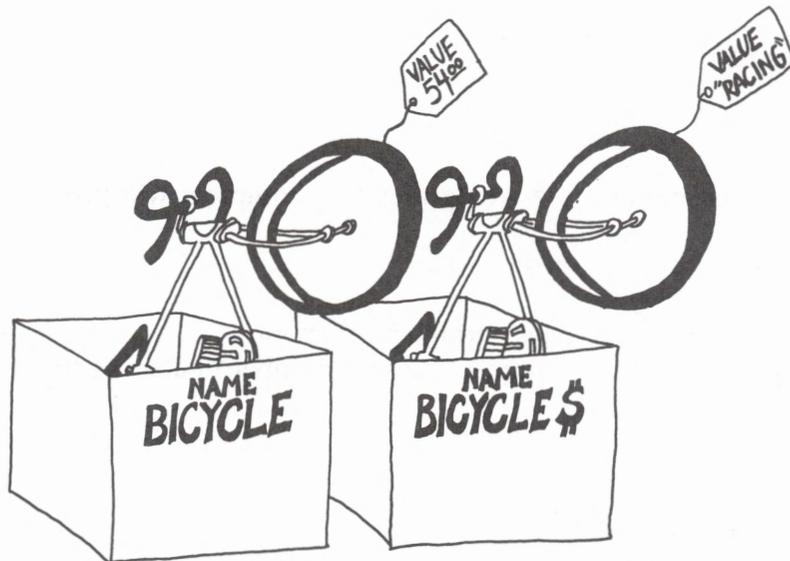


Variables

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 in the box is called the *value* of the variable.



Arithmetic in the LET Statement

Enter this:

```
10 LET A=2001
20 LET B=1985
30 LET C=A-B
40 PRINT "How much longer, Hal?"
50 PRINT C; "years."
```

Careful!

Numbers and strings are different. For example, the string "1985" is not a number. It is a string constant because it is in quotes.

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

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

Some string constants are "hi", "7", "two", "3.14".

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

Correct: 10 LET A = 3 + 7

Wrong: 10 LET A\$ = 3 + 7

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

If you try to enter any 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 okay, but 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 statement. (Just remember that you cannot do arithmetic with the mixture.)

Correct: print a;"seven";"7"

 print a;b\$

Run this: 10 PRINT 5/2;"is equal to 5/2"

A Funny Thing About the Equal Sign

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

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

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

$$7=7+1$$

which is not correct.

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

The computer goes to the box with N written on the front. It takes the number 7 from the box. It adds 1 to the 7 to get 8. Then it puts the 8 in the box.

Here is another way to say the same thing:

Let the *new N* equal the *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 with LET, you cannot.

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

In BASIC, LET $N = 3$ is correct, while LET $3 = N$ is wrong.

In BASIC, LET $N = B$ is not the same as LET $B = N$. Why not? What is in each box?

LET $N = B$ means _____

LET $B = N$ means _____

Assignment 10

1. Write a program that asks for the user's age and the current year. Then it subtracts and prints out the user's year of 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

TAB mimics the familiar tab function of a typewriter. This lesson discusses *delay loops*, which are useful in themselves, and are really just empty FOR-NEXT loops.

TAB must be used in a PRINT statement. Several TABs 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 discuss LOCATE 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 it is recommended.

This lesson introduces the use of delay loops to slow a 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 statement. The amount of delay is determined by the size of the loop variable. A value of 1000 gives a delay of about one second.

When the student understands 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 to handle loops in which things are going on inside.

Questions

1. Show how to write a delay loop that lasts 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. phone home"
```

Lesson 11. TAB and Delay loops

The TAB Function

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

(Remember, the print 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 in 10. You have to use TAB in a PRINT statement. You cannot use TAB by itself.

You Cannot TAB Backward

Try this:

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

TAB can move the printing to the right only. 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 GOTO 40
```

Press the Break key to stop the program.

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.

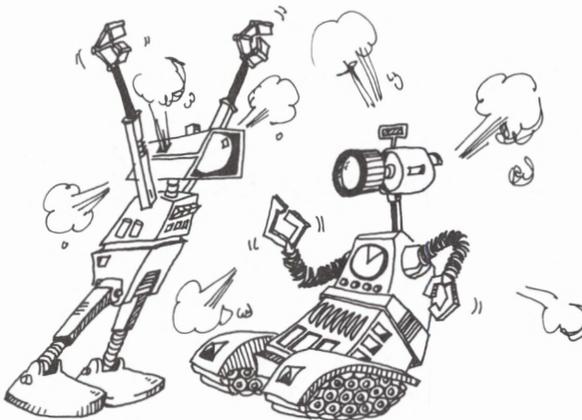
Try this:

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

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

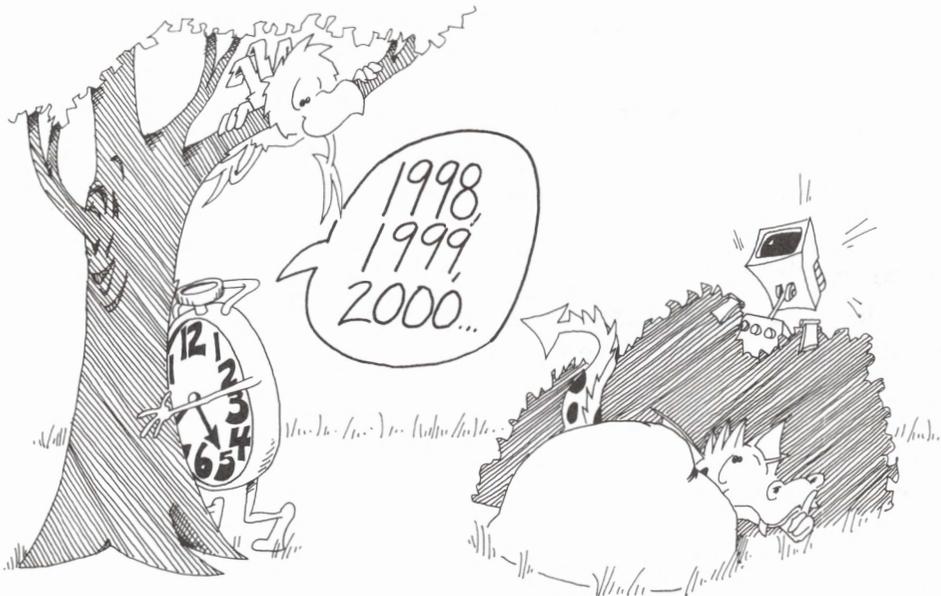
Functions Don't Fight But They Have Arguments

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



Assignment 11A

1. Write a program that asks for last names and nicknames. Then it prints the last name starting at column 5 and the nickname at column 15. Use a GOTO so the program will be ready for another name/age pair.
2. Write a greeting program. It asks your name. Then it beeps and writes your name. Then it TABs over in the line and prints a greeting.



Delay Loops

Here is a way to slow down parts of a program. It is called 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*650
40 FOR I=1 TO T:NEXT I
45 PRINT
46 BEEP
50 PRINT S;"seconds are up"
```

Assignment 11B

Write a slowpoke program that prints out a three-word message with several seconds between each word. Have the computer beep before each word.

Instructor Notes 12. The IF Statement with Numbers

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

= > < <>

It is a good idea to get the student to pronounce these expressions out loud. $A < B$ makes a lot more sense when pronounced *A is less than B* than when it's just allowed to flow over the eyeballs. Of course, the point (the little end) of the $<$ or $>$ symbol is at the side of the smallest of the two numbers.

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

A loop is demonstrated in the "Guessing Game" program, but it 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 statement can be true or false?
2. What follows the THEN in an IF statement?
3. After this little program runs, what will be in box D?

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

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

Lesson 12. The IF Statement with Numbers

Try this:

```
10 REM *** Teenager ***
15 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 statement is like the one that you used before with strings. Again we have

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

Phrase A can have these arithmetic symbols:

- = Equal to
- > Greater than
- < Less than
- <> Not equal to

Each phrase A is written 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 the following examples, LET A=7 and LET B=5 and LET C=5.

Say each phrase A aloud 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 line, break it into two parts:

```
60 IF A>12 THEN statement C  
   where statement C is  
   IF A<20 THEN PRINT "Teenager!"
```

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

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

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

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

Assignment 12A

Draw the fork in the road diagram for line 60 above. There will be two forks on the diagram. (See lesson 9.)

Guessing Game

```
10 REM ----GUESSING GAME----
15 CLS
20 PRINT "Two player game"
25 PRINT
30 PRINT "First player enter a number from 1 t
   o 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<N THEN PRINT "Too small"
65 IF G>N THEN PRINT "Too big"
70 IF G=N THEN GOTO 90
80 GOTO 50
90 REM The game is over
92 PRINT
95 PRINT "That's it!"
```

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

Line 80 usually 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?

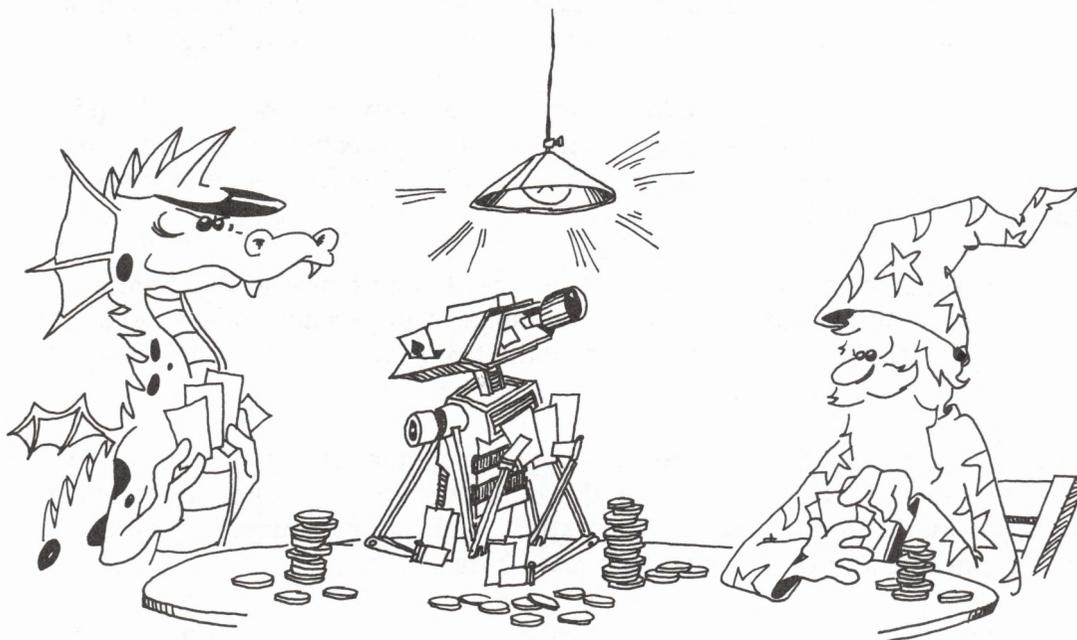
```
10 LET N=1
20 IF N=13 THEN PRINT "unlucky!"
30 LET N=N+2
40 IF N>30 THEN GOTO 99
50 GOTO 20
99 PRINT "done"
```

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, and so on.
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, the player guesses the suit, then the program asks the value. Keep score.



Instructor Notes 13. Random Numbers and the INT Function

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

The RND function produces pseudorandom decimal numbers larger than 0 and smaller than 1.0. Such numbers are directly usable as probabilities, but integers over some range, such as 1 to 6 for a die or 1 to 13 for the face value of cards, are often more directly usable.

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

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

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

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 as output for this line:

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

when the box G contains 2, 2.1, 2.95, 3.001, 67, 0, or 0.2.

-
-
2. Tell how the INT function is different from *rounding off* numbers. Which is easier for you to do?
 3. Tell how to change a number so that the INT function will round it off.
 4. What does 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 them. 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 bigger than 0. I chose 9 because it is near the parentheses on the keyboard which makes it easy to type (9).

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

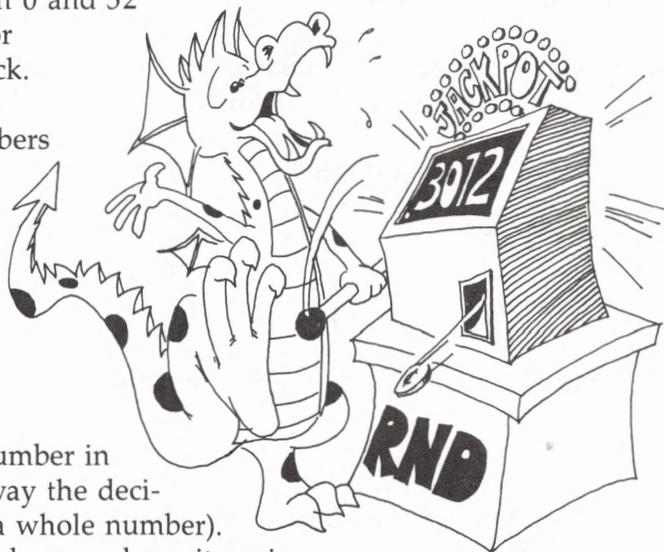
Change the program above to

```
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 (a whole number). Add this line to the program above and run it again:

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

How It Works

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

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

Run it many times and try these numbers in the parentheses: 0.3, 0.5, 0.9, 1.0, 1.1, 1.49, 1.51, 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 is below 0.5, you round down.

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

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

Run:

```
10 REM *** rounding off ***
20 CLS
22 PRINT "Give me a decimal number"
25 INPUT N
30 PRINT "rounded to the nearest integer"
40 PRINT INT(N+.5)
45 FOR T=1 TO 2000:NEXT T
50 GOTO 20
```

Press the Break key to stop the program.

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

Rolling the Bones

Ordinarily, 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
```

Use just

```
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)*3)+6 : PRINT D

Here's how it works:

Expression	Makes numbers from	
	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! Use the RANDOMIZE statement early in the program to make the computer choose different random numbers from those on the last run.

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

Instructor Notes 14. Saving to Disk

This lesson shows how to save programs to the disk and how to load them again. It can be used anytime after lesson 3.

It introduces the commands SAVE, LOAD, FILES, and KILL.

Other commands and statements used in this chapter are NEW, REM, LIST, PRINT, and CLS.

We delayed this subject until now because most programs so far were relatively short and uninteresting—not worth saving. The process of programming was being emphasized, not the end result of useful programs.

However, your own judgment should prevail, and you can insert this lesson at an earlier point if your student wants to save some programs.

Questions

1. What is a *file*?
2. How long can a filename be?
3. Can punctuation marks be in a filename? Can the filename have spaces in it?
4. How can you check to be sure the program got on the disk without problems?
5. What happens to the program already in memory if you load another program?
6. Does the filename have to be the same as the program name?
7. If a program is put into a file, is it still in memory?

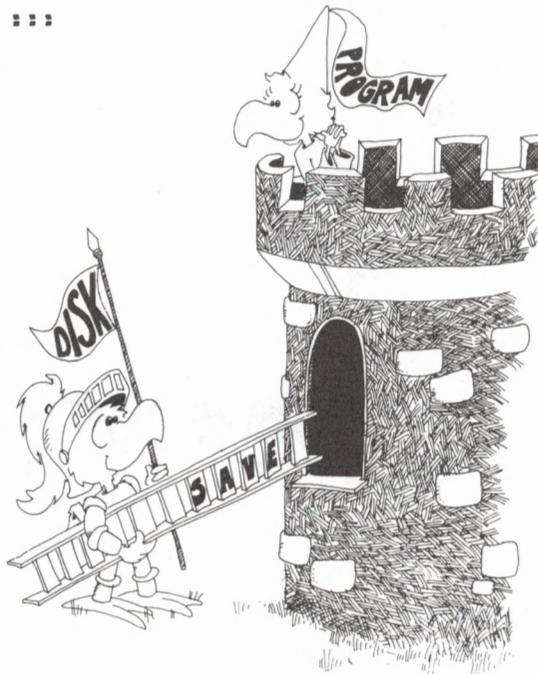
Lesson 14. Saving to Disk

Entering a Program

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

If not, enter `new`, followed by pressing the Enter key. Then enter these lines:

```
10 REM ::: hi :::  
20 CLS  
30 PRINT "hi"
```



Saving a Program

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

Enter: `save "hi"`

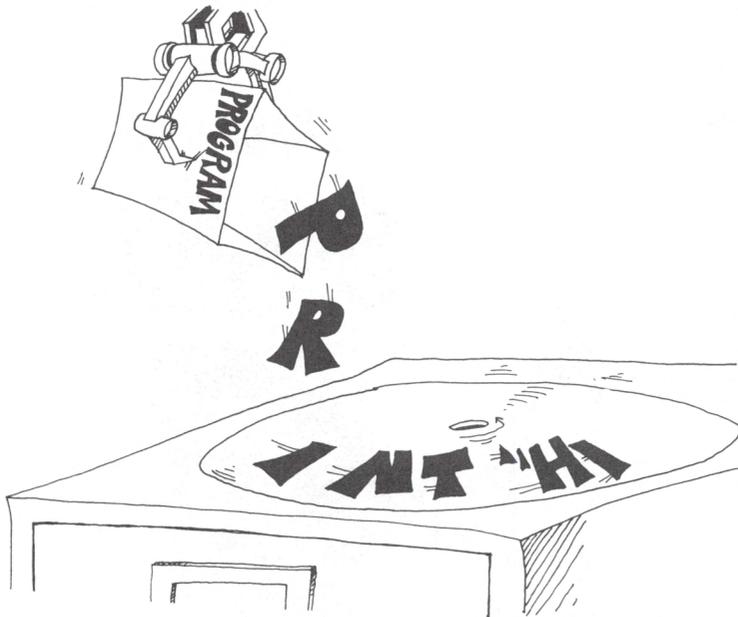
You will hear a whirring sound and see the red light on disk drive A. When the red light goes off and the whirring stops, your program is stored on the disk.

The filename 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.

If you mess things up, press the Break key 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, if you are in 80-column display mode, you will see a list in three columns of all the files on the disk. Your file is probably the last one in the list. It will say

```
HI      .BAS
```

Filenames

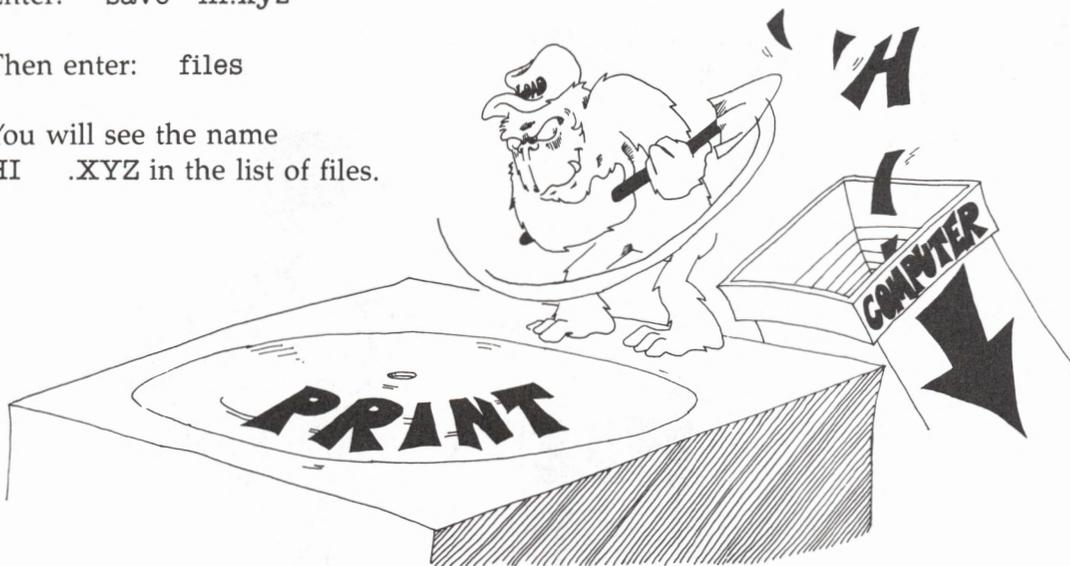
Each file has a name that is from one to eight characters long. The filename can be followed by an *extension*. The extension starts with a period (.) and has one, two, or three characters in it. If you have not given your filename an extension, and the file is a BASIC program, the computer automatically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

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 and press Ctrl-Home (Ctrl-Fn-Home on the PCjr). 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"  
then  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 KILL will give you an error message:

```
File not found
```



Legal Filenames

A filename can have

- up to eight characters plus an extension.
- numbers in it.
- only one period in it (the start of the extension).
- spaces in it.
- punctuation in it, *but*
it cannot have a comma in it.

An 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.bbcb  
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 Filenames

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

Commands

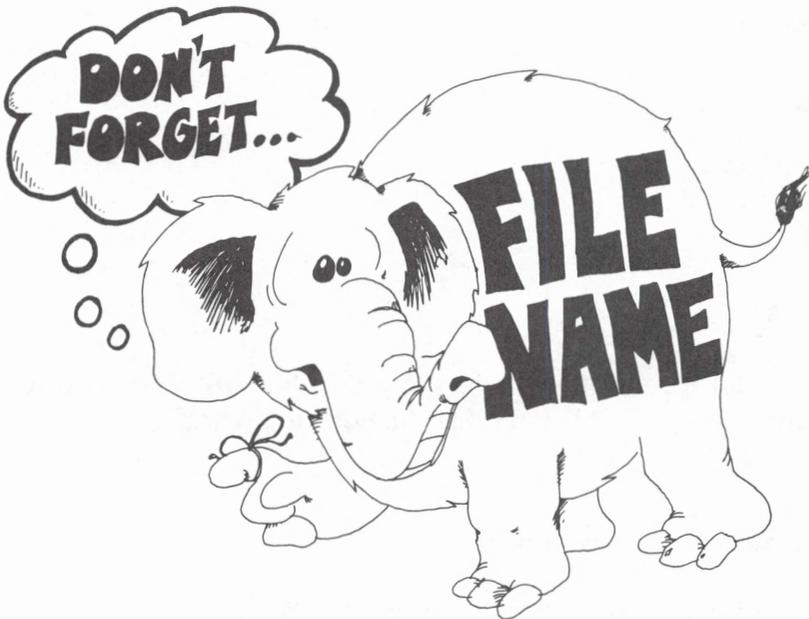
These four commands are used with files:

```
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, too.
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

These shortcuts are discussed:

?	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

Now that the student knows how to use RND and can save programs on disk, all the elements are in place for writing substantial programs.

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

The colon allows you to put a 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 less cluttered program results. The colon thus becomes a powerful and nontrivial means of improving the clarity of the program.

The colon can, however, cause problems in a program. Be careful about adding other statements onto a GOTO, a REM, or an IF line.

Since a question mark (?) is always printed on the screen by INPUT, an INPUT message should not end with a question mark.

Questions

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

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

5. What is wrong with each of these lines?

```
10 REM Beginning:GOTO 1000  
10 GOTO 50:S$="Fast"
```

6. If the computer prints ?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

These two are the same as well: 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 this:

```
10 INPUT "Enter your name"; N$
```

Put a semicolon between the message Enter your name and the variable.

Another INPUT Shortcut

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

Run:

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

You see Location? on the screen. Enter two numbers with a comma between them.

```
Location? 5,6
```

Another example:

```
30 INPUT "Month, Day, Year";M$,D,Y
```

After the question mark, type: April,29,1985

Error Message in INPUT

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

```
?Redo from start  
?
```

This also 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  
RUN  
?May,1  
?Redo from start  
?May,1,1985
```

Another Way to Get an Error Message

Run 10 INPUT N, A\$ and try these pairs of answers:

l, b
b, l
l, l
b, b

The error message ?Redo from start is put on the screen whenever the user answers with a string for a number.

(It is okay to answer a "number" for a string, because the computer says, "Okay, 1985 is a string," even if you meant it to be a number.)

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 the 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
```

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

When you LIST the line, you will 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.

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:

```
40 H=X+Y/66 : REM H is the height
```

The Colon After an IF Statement

You can make neater IF statements by using colons.

Without colons:

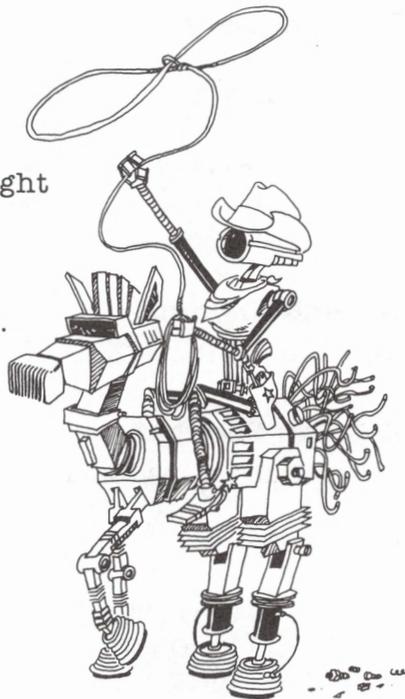
```
50 IF A=0 THEN GOTO 80  
60 B=Q  
62 C=B*D  
66 PRINT "Wrong"  
80 FOR ...
```

With colons:

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

All the statements in the path where $A \neq 0$ is true are on the line after THEN.

Careful!



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

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

This is not the same as

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

This is 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 GOTO statements must be last on a line. Anything following them is ignored.

Correct: 35 P=3:REM P is the price

Wrong: 35 REM P is the price:P=3

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

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

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

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

A REM Shortcut

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

This: 10 INPUT N\$ ' The name of the user
is like this: 10 INPUT N\$:REM The name of the user

You don't need to put a colon before the single quote.

Assignment 15

1. Write a program that uses each of the 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 use LOCATE effectively, you need to think of the screen as an array, with 40 (or 80) characters across and 24 lines down. Remember the phrase *row, column* to get the order of arguments in the LOCATE *R,C* statement. (We will find that the order is reversed when we get to graphics. There we will need the phrase *X,Y* with *X* across and *Y* down.)

Later, there will be two lessons on the graphics statements, but here we show graphics to accustom the student to several ideas. One is the use of down-across graph paper and another is making 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 the background color before the new object is drawn.

It is best to delay erasing until just before the new object is drawn. This reduces 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 at the left of line 12 on the screen, what statement do you use?
2. If you want to print the next character on line 6 of the screen, 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 word `CAT` starting at column 25, and then after a delay, print `FAT` starting at column 5 of the same line.

Lesson 16. Moving Graphics and LOCATE

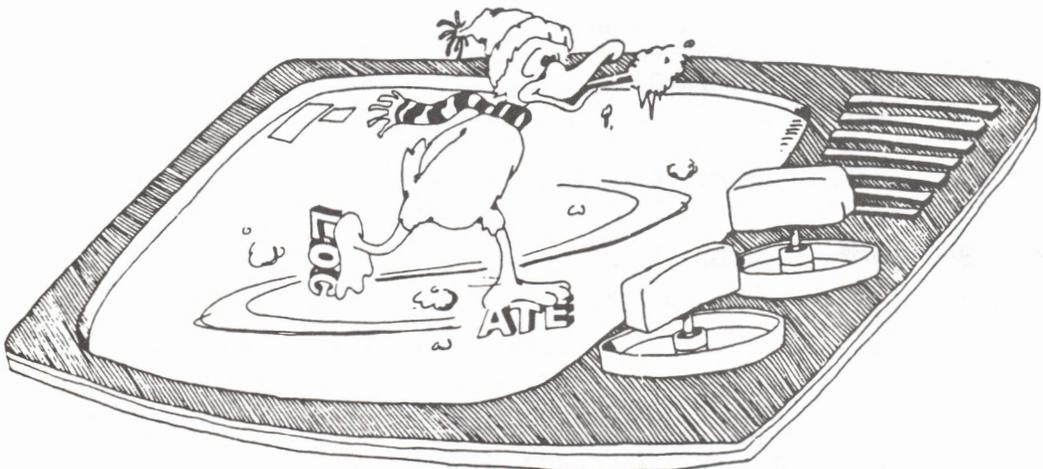
There is room for 24 lines of typing on the screen. The lines are numbered from 1 at the top to 24 at the bottom.

Each line can hold 40 or 80 characters (depending on your setup). They are numbered from 1 on the left to 40 (or 80) on the right.

Run this:

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

The first number in a LOCATE statement tells which row the print cursor will go to.



Jumping Anywhere on the Screen

Run:

```
10 REM column and row
15 CLS
20 INPUT "which row <1-25>";R
25 IF R<1 OR R>25 THEN 15
30 INPUT "which column <1-40>";C
35 IF C<1 OR C>40 THEN 15
40 LOCATE R,C:PRINT "*";
50 FOR T=1 TO 1000:NEXT T
60 GOTO 15
```

Press the Break key to stop the program.

The second number tells which column the print cursor will go to.

Erasing What You Write

```
10 REM jumping here
15 CLS
20 C=INT(RND(9)*35)+1
25 R=INT(RND(9)*23)+1
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$=" " THEN 20 ' erase
42 IF C$="d" THEN F=1 ' draw
43 IF C$="e" THEN F=0 ' not draw
44 IF C$<" " 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$="m" THEN Y=Y+1 ' move dot down
52 IF C$="j" THEN X=X-1 ' move dot left
53 IF C$="k" THEN X=X+1 ' move dot right
60 IF X<1 THEN X=1
61 IF Y<1 THEN Y=1
62 IF X>40 THEN X=40
63 IF Y>23 THEN Y=23
75 LOCATE Y,X:PRINT CHR$(219);
99 GOTO 30
100 '
101 ' 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":PRIN
T
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
199 GOTO 20

```

When you enter this program, you can omit most of the REMs. Before you run the program, save it to disk. Add your own REM statements using the information below.

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.)

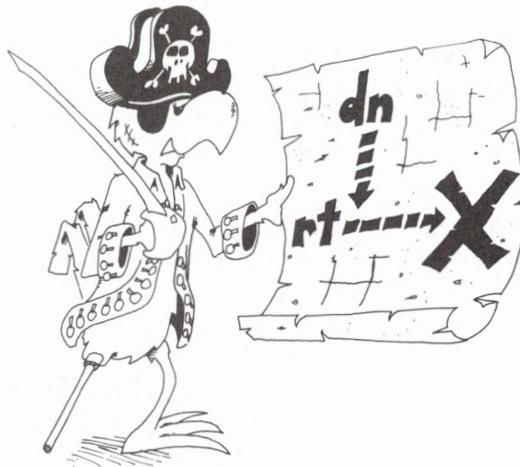
Line 46 picks a color number and calls it P. The ASCII number for 0 is 48, for 1 is 49, and so forth. Subtracting 48 from C gives a number from 0 to 7 to use as a color.

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 puts the dot on the screen.



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 shape on the screen.

Instructor Notes 17. FOR-NEXT Loops

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

Nested loops are introduced by using an example 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 programs, but it may cause an unexpected bug if you copy certain non-IBM programs into IBM BASIC.

The FOR statement is evaluated just once when the loop is entered. It puts the starting value of the loop variable into variable storage where it is treated just as 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 statement. Upon reaching NEXT, the loop variable is incremented by the value of the STEP and compared with the end value. If the loop variable is larger than the end value (or smaller in the case of negative STEPs), NEXT passes control to the statement after itself. Otherwise, it sends control to the statement after the FOR statement.

Because the loop variable is treated just like any other variable by BASIC, it can be used or changed in the body of the loop. 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) this may cause bugs that will be hard to find.

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 ten down to zero Indians.

Lesson 17. FOR-NEXT Loops

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

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

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

Run this:

```
10 REM counting
20 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 drawing arrows:

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



STEP

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

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

Assignment 17A

Have the computer count by fives from 0 to 100.

Countdown Loops

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

Try this:

```
10 REM *** APOLLO 11 ***
20 CLS
30 PRINT " T minus 12 seconds and counting"
40 FOR I=11 TO 1 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 the "Apollo 11" program, we have one loop inside another.

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

The inside loop is in line 60.

These are nested loops. They are like a child's set of toy boxes which fit inside each other.

Loop Variables

To make sure that each FOR knows which NEXT belongs to it, the NEXT statement 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, I is the loop variable:

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

Badly Nested Loops

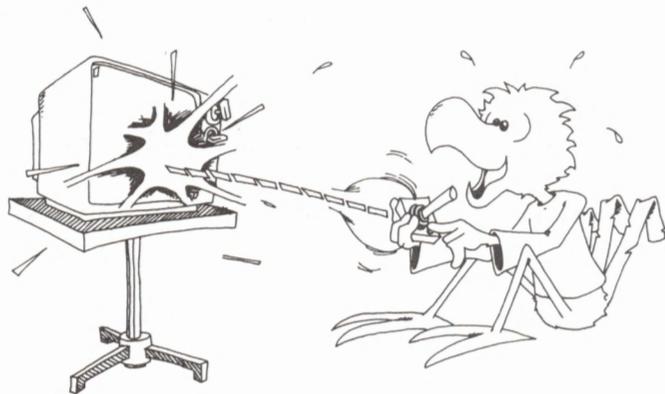
The inside loop must be all the way inside:

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 the counter, I, should start at 5 and get bigger, until it is larger than 3. But it is bigger than 3 to start with.

So you would like 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 is used that hasn't been defined, 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. 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 line, and keep switching until each name is written five times.



Instructor Notes 18. DATA, READ, and RESTORE

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

You can never change any of the data in the statements unless you rewrite the program lines. 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. And 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 here and may be mentioned to the student when other ideas about DATA are well-entrenched.)

In the IBM PC and PCjr, you can skip data by arranging it in different DATA statements and point 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 instructor's hand, pointing to items in a DATA statement, helps clarify this concept.

Using DATA saves some typing errors if you have a lot of data. Moreover, it is useful in cases where there is not much data, because it clearly separates the actual data from the processing of the data. This helps when debugging programs. 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 as the program runs.

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

```
10 REM The second kind of data
20 CLS
30 X=57
40 Y$="flavors"
50 PRINT X;Y$
```

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

Storing Lots of Data

It is all right 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,Thursd
ay,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), box D2\$ holds the second item (Monday), and so on.

Strange Rules

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

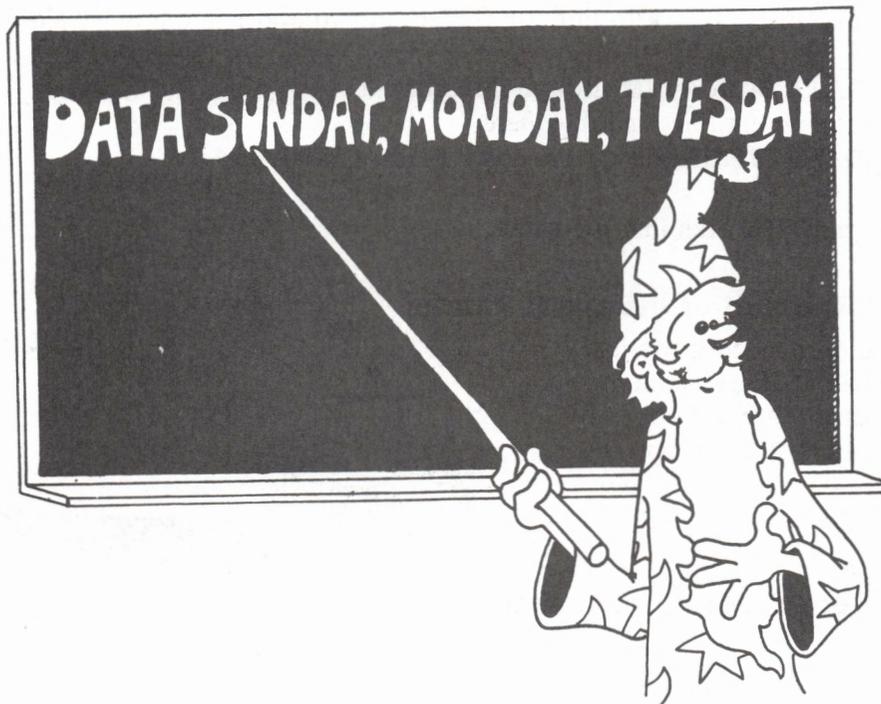
Change line 30 in the above program to line 90. (Remember to erase line 30.) Run the program. It works just the same.

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

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

READ uses a *pointer*. It always points to (indicates) 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 statement, the pointer moves to the next item in the DATA list.

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

It doesn't matter if there are a lot of lines in between.

Do this. Change line 90 back to line 30. (Erase line 90 and leave line 91 alone.)

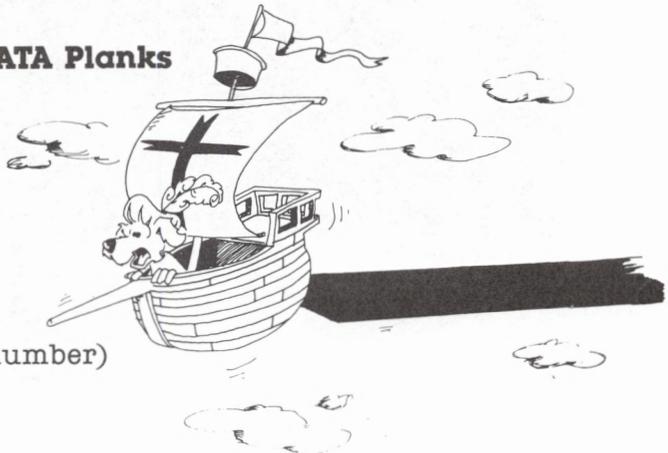
```
30 DATA Sunday, Monday, Tuesday  
Other program lines could go here.  
91 DATA Wednesday, Thursday, Friday, Saturday
```

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 in (line number)



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: The READ pointer moves ahead one item.
2. You can RESTORE the pointer to the start of the DATA: The READ pointer is put back to the beginning of the first DATA statement in the program.
3. You can RESTORE the pointer to the start of any line: For instance, if you use RESTORE 30, the READ pointer is put on the first item in the DATA statement at line 30 in your program.

Mixtures of Data

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

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

Correct:

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

Wrong:

```
70 DATA 77,fuzz
75 READ B$      Okay, B$ box holds 77
80 READ N      Syntax error in 70
```

The error is because you can't READ a string (fuzz) into a number box. Change READ N to READ N\$.

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 or sisters, for example, each with a DATA statement.
2. Use DATA and READ statements to write an invisible message program. READ the messages from DATA statements and use PRINT to display them. 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 statement makes a tone of specified pitch and duration.

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

The IBM PC and PCjr computers do not have full sound-effect capability, mainly because they lack a *white noise* generator. They also lack an *envelope* generator to control the attack and decay of notes. They can, however, make musical tones that are accurate in pitch, because pitch can be specified to five digits of precision.

Two arguments are needed in the SOUND statement. The first is the pitch in cycles per second, or hertz (Hz). This variable takes values from 37 to 32767. Of course, human perception is in the range of 20 to 20,000 hertz, or less in older people.

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 counts 18.2 times a second.

Music is most easily made if you use the PLAY statement described in lesson 23.

When using sound in graphics situations, you get the most elaborate effects if you interleave the sound commands with the graphics movement commands.

The DATA statement 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

IBM BASIC has three sound statements: BEEP, SOUND, and 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.

You'll learn 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=37 TO 100 STEP 10
20 SOUND I,20
25 NEXT I
30 FOR I=100 TO 1000 STEP 100
35 SOUND I,20
40 NEXT I
50 FOR I=1000 TO 5000 STEP 1000
55 SOUND I,20
60 NEXT I
```

The first number after SOUND is the pitch. Any number from 37 to 32767 can be used. Larger numbers give higher notes. In fact, the number gives the pitch in *hertz*, which means cycles per second.

You can play simple musical tunes using SOUND statements.

Notes of Different Length

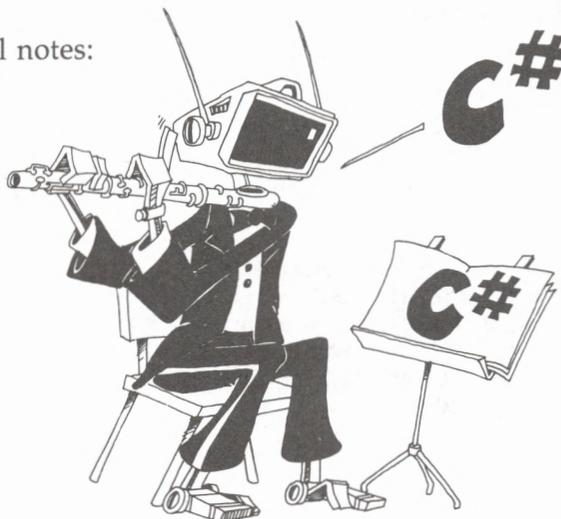
The second number in the SOUND statement gives the length of the note. Any number from 0 to 65535 can be used.

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

It plays A above middle C for about five seconds.

You may not be able to hear anything for pitch numbers much above 10,000.

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
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. Use DATA statements to store the pitch numbers.



Instructor Notes 20. COLOR

If your screen looks smeary or washed out when you use color graphics, try putting a line with

```
screen 0,1
```

in the early part of your programs.

A different style of drawing is used in color from that used 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 this 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. It's often helpful to use graph paper to draw out the picture first. 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's easy to move the whole figure if necessary for animation or just for correction of the composition. See the "Jumping J" program in lesson 24.

Questions

1. What does the statement `COLOR 0,7` 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 statement `LOCATE X,Y`?

Lesson 20. COLOR

Adjusting the Monitor for Color

If your monitor is black and white, skip to "Rainbow Ball" below.

If you have a PC, load the "Colorbar" program from your student disk and run it.

```
load"colorbar.bas"  
run
```

If you have a PCjr, use the Colorbar display when you first turn on your computer to adjust the color settings on the monitor or TV.

These are the colors:

0 Black	8 Gray
1 Blue	9 Light blue
2 Green	10 Light green
3 Cyan (blue-green)	11 Light cyan
4 Red	12 Light red
5 Magenta (reddish 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

In Chapter 2 you learned how to use the COLOR statement to make a colored background (with black writing on it).

Like this:

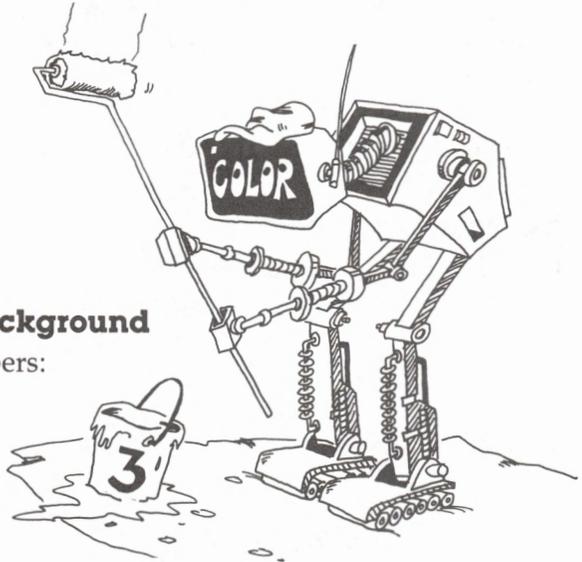
```
10 REM colored backgrounds  
15 CLS  
20 COLOR 0,4  
30 PRINT "black letters on red"
```

The second number after the statement 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
11 CLS
12 CLS
15 FOR I=0 TO 15
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 colors of background. 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=16 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 this:

COLOR I,J,K

2. Make K change 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
54 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 CLS
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.

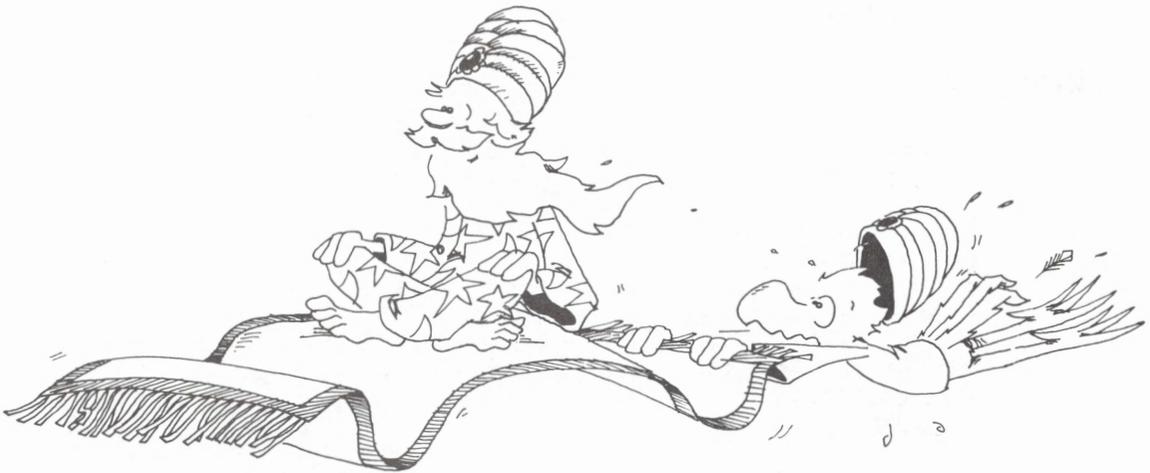
Then line 40 waits for a moment.

Line 41 erases the ball that was just printed.

Line 50 increases the color number (first set in line 16) 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 that it falls instead of moving across the screen.
2. Add to the number guessing game in lesson 12 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 the number of colors in the rug and what colors are used . 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 making line drawings.

The IBM PC can make medium-resolution graphics (320 × 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 0 tells the computer to go back to text-only display. You can still PRINT and use LOCATE in graphics 1 mode, which is a real convenience.

If you have a green screen monitor, SCREEN 2 gives higher resolution (a 640 × 200 screen), and the restriction to use of only two colors doesn't hurt. We will not specifically describe graphics 2 mode here; 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 statements PSET, LINE, and CIRCLE all use the notation (X,Y) for points needed in the statement.

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 (-20,-30),120 and put part of a circle on the screen. (Even the center is off the screen!)

A handful of powerful graphics statements will not be discussed in this book. They are DRAW, and PUT and GET, which allow animated graphics.

Questions

1. Where on the screen will PSET (160,100) 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 statement fixes the screen so that you can use the PSET, LINE, and CIRCLE statements to draw pictures. It needs IBM Advanced BASIC and the Color/Graphics Monitor Adapter.

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

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

The PSET statement 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 Statement

SCREEN 1 makes the screen ready for medium-resolution graphics, 320 points across \times 200 points down.

SCREEN 0 changes the screen back to text only, 40 letters across \times 25 lines down.



The PSET Statement

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.

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. Or have your set adjusted properly.

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

Upper left _____

Upper right _____

Lower left _____

Try them and see if they work.

Run:

```
10 REM stars
12 SCREEN 1:CLS
20 FOR I=1 TO 100
30 X=INT(RND(9)*320)
31 Y=INT(RND(9)*200)
33 PSET(X,Y)
35 FOR T=1 TO 50:NEXT T
40 NEXT I
50 SCREEN 0
```

Add a REM to line 31.

The LINE Statement

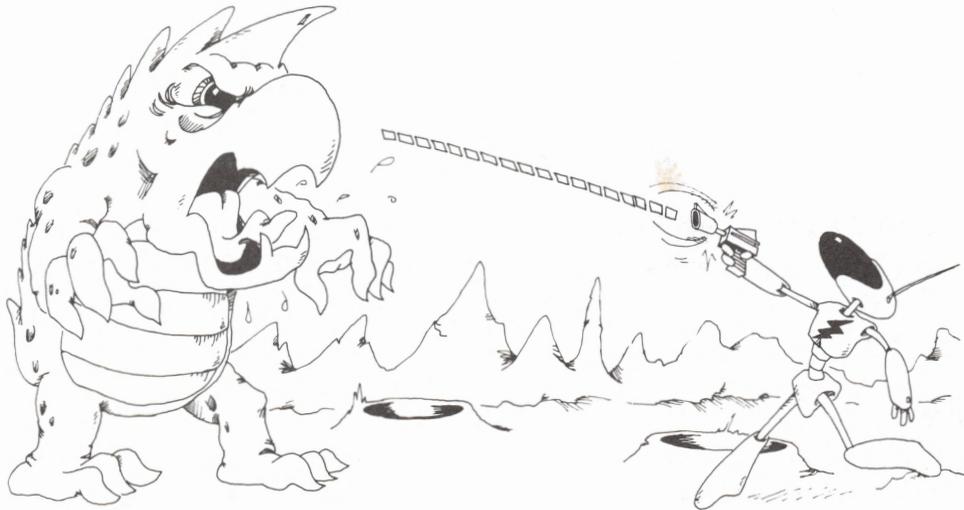
The LINE statement draws a line between two points on the screen.

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 Statement

The CIRCLE statement draws a circle with a center point given by (X,Y) and a radius R .

```
10 REM circles
12 SCREEN 1:CLS:PRINT:PRINT:PRINT
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 choose.

Mixtures of Letters and Graphics

You can mix letters and graphics 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 snowperson, LINE for the 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 statement 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 a *brush*.

There are only two palettes allowed, 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 palette 0 colors. This may seem strange as you remember your watercoloring days in kindergarten. When adding colored lights instead of pigments, yellow (which is made of red and 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 statement 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 statement because it specifies at which color (brush number) the filling in of color will stop.

The LINE statement also allows you to draw rectangular boxes. Just add the letter *B* on the end of the command. Adding the letter *F* (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 (30,40) mean in the statement

PAIN~~T~~ (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 palettes, something about borders, and something about PAIN~~T~~.)

Lesson 22. Color Graphics

Pick Your Palette

Run:

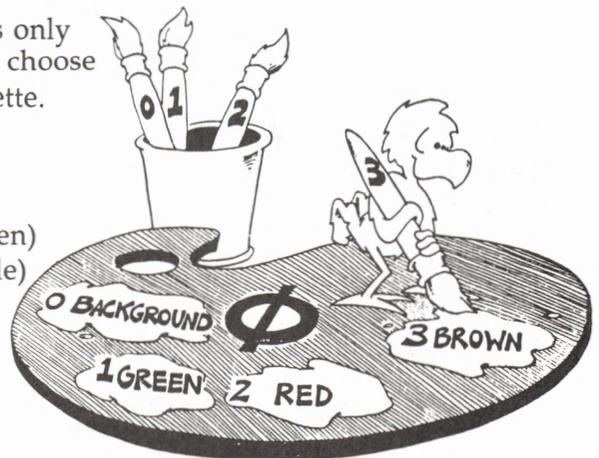
```
10 REM dueling artists
12 CLS:SCREEN 1
20 FOR B=0 TO 15
22 FOR P=0 TO 1
25 COLOR B,P
26 LOCATE 7,20:PRINT"palette ";P
27 LOCATE 5,20:PRINT"background color";B
30 CIRCLE (50,50),70,3
31 PAINT (50,50),1,3
32 CIRCLE (80,99),80,1
33 PAINT (91,120),2,1
34 LINE (90,70)-(280,150),3,BF
40 LOCATE 3,5:PRINT"color 1"
41 LOCATE 17,4:PRINT"color 2"
43 LOCATE 10,25:PRINT"color 3"
49 FOR T=1 TO 500:NEXT T
60 NEXT P,B
98 SCREEN 0,1
```

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

Example: 15 SCREEN 1:COLOR 13,1 gives background color 13, palette 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
30 LINE (20,20)-(100,100),2
40 CIRCLE (160,100),50,1
50 PSET (160,100),3
```

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



Colored Box in Outline

The LINE statement can also draw a rectangle. Just add the letter *B* after the brush number.

Run:

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

Enter SCREEN 0,1 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. It means you fill in the box outline with color.

Solid Colors for Your Drawings

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

Run:

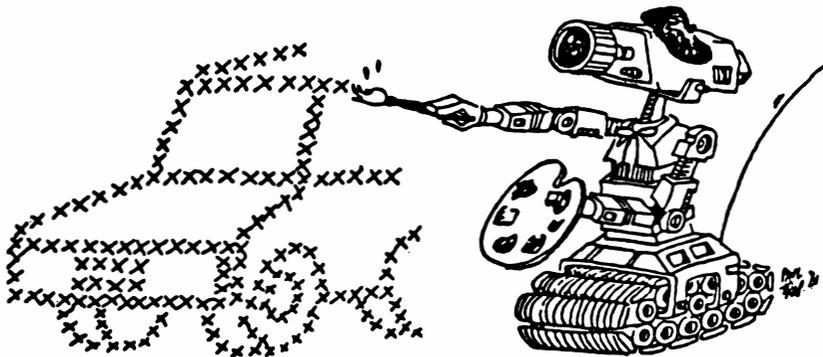
```
10 REM paintbox
12 KEY OFF:SCREEN 1:COLOR 7,0
19 REM a triangle outline in green
20 LINE(50,50)-(100,0),1
22 LINE(50,50)-(100,100),1
24 LINE(100,0)-(100,100),1
30 LOCATE 12,20:PRINT"green outline"
40 FOR T=1 TO 1000:NEXT T
50 LOCATE 14,20:PRINT"use brush 2 to fill with
   red"
60 FOR T=1 TO 1000:NEXT T
70 REM brush 2 (red).fill to green (1) boundar
   y
71 PAINT (60,50),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 character 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 statement. For example, B \flat is okay (for B-flat), but B \sharp is not (it is the same as C). The octave symbol sets the octave (0 to 6) that the notes are chosen from. 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 of n compared with 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 P n , 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 it can be improved by using staccato and legato. Staccato sounds as expected, but legato really comes out as a tie. That is, two legato C quarter notes played together will 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. 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$="MN T100 03 L4 C C L6C L16D L4E L6E L16D
    L6E L16F L2G 04 L12CCC 03GGG EEE CCC L6G 1
    16F 16e 116D L2C"
30 PLAY M$
```

Run it and save to disk.

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

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

The computer 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)
Rests:	Full note rest	P1 (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: If a musical note is *dotted* (followed by a period), it means you should play

the note one and a half times its normal length. For example, a dotted half note should be played like a half note and a quarter note added together.

Example: m\$ = "cdeplc#d#f.pl."
means m\$ = "c d e pl c# d# f. pl."

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

The following symbols are put in just one time. They will be in effect from then on or until you make a change by putting in a new symbol.

Octave: O0 O1 O2 O3 O4 O5 O6 O7
Octave 0 is the lowest.
Octave 3 starts with the note middle C.
Octave 7 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: Larghissimo very slow T 32 to T 39
Largo T 40 to T 59
Larghetto T 60 to T 65
Grave
Lento
Adagio T 66 to T 75
Adagietto
Andante slow T 76 to T107
Andantino
Moderato medium T108 to T119

Allegretto		
Allegro	fast	T120 to T167
Vivace		
Veloce		
Presto		T168 to T208
Prestissimo		T209 to T255



- 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: m\$ = "mf ms t100 l8 o4 c d e f g a "
 m\$ = "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$; t220 xm$; "
```

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 that it plays another tune.
3. Crossing friends: While music plays, make your name move down the screen while a friend's name moves across the screen. Use the MB symbol in the string. Make the names cross in the middle.



Instructor Notes 24. Pretty Programs, GOSUB, and 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 statement following the GOSUB after the subroutine is finished executing. This is accomplished by storing the statement address following the GOSUB statement on a *stack*. When the computer encounters a RETURN statement, it pulls the address off the stack and returns control to that statement.

Subroutine calls can be nested at least 30 deep.

The END statement can be put anywhere in the program and you can use as many END statements as you wish. All that END does is to stop the program and 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 in 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 statement END is executed?
2. How is GOSUB different from GOTO?
3. What happens when RETURN is executed?
4. If RETURN is executed before GOSUB, what happens?

-
-
5. What does *call the subroutine* mean?
 6. How many END statements are you allowed to put in one program?
 7. Why do you want to have subroutines in your programs?

Lesson 24. Pretty Programs, GOSUB, and RETURN

Run this program and then save it to disk:

```
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
```

Remember to save it to disk.

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

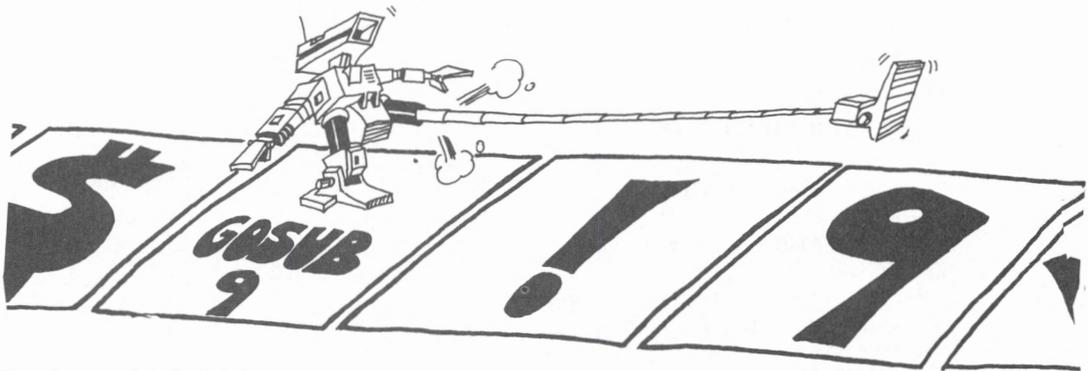
Where there are PRINT statements, you can put many more program lines.

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

Lines 120 and 140 *call the subroutine*. This means the computer goes and performs the commands in the subroutine, then comes back.

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

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



Assignment 24A

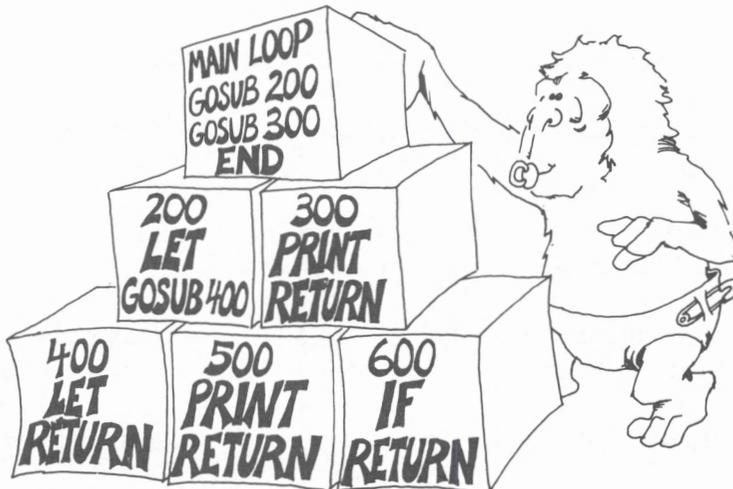
The delay loop is used three times in the above program. Add another subroutine with a delay loop in it, and GOSUB every time you need a delay.

What Good Is a Subroutine?

In a short program, it's not much good.

In a long program, it does two things:

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



The END Statement

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

Rule: The END statement tells the computer to stop running and go back to the edit mode.

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

Moving Pictures

Run this:

```
10 '??? Jumping J ???
20 CLS
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
95 CLS
98 COLOR 7,0
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
```

Remember to save to disk.

The picture is the letter *J*. The subroutine starting in line 110 draws the *J*. Before you GOSUB 110, you pick what color you want the *J* to be, using a COLOR statement. Look at lines 30 and 35. If you pick color 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 ten 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.

Put three subroutines in it. Call one of them twice from the main program. Call one of them from another of the subroutines. Call one of them from an IF statement.

2. Enter the "Jumping *J*" program and run it. Then make these changes:

Change the subroutine so that it prints your own initial. Change the color of your initial to blue.

Change the jumping to sliding (so the *J* moves across the screen instead of up and down).

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 the total distance moved is 10 squares.

Change the sliding so that it slides uphill. Use this:

$$X=X+D;Y=Y-D$$

Change the program so that 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 a different color. Then make them jump up and down one at a time.

Instructor Notes 25. ASCII Code and ON-GOTO

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

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

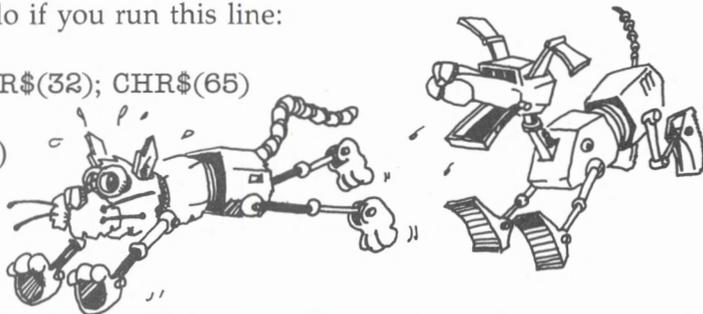
Strictly speaking, there are only 128 ASCII characters, and some of these are supposed to signal mechanical actions on a Teletype machine. It is more convenient to define a full byte's worth (256) of characters, and each computer manufacturer uses the extra characters in a unique way. IBM assigns them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

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. Answer the 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 I? 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 and ON-GOTO

Numbering the Letters in the Alphabet

"That's 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 Characters into Numbers

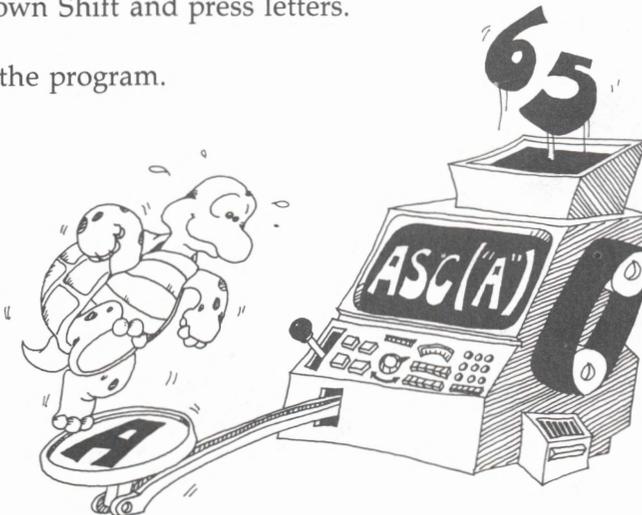
Use the ASC function to change characters into ASCII numbers.

Run:

```
10 REM *** What number is this key? ***  
20 PRINT "Enter keys to see ASCII number"  
30 INPUT C$  
40 PRINT TAB(10);C$;TAB(15);ASC(C$)  
50 GOTO 30
```

Save the program to disk before running it. Try out some letters, digits, and punctuation marks. Hold down Shift and press letters.

Press the Break key to end the program.



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
```

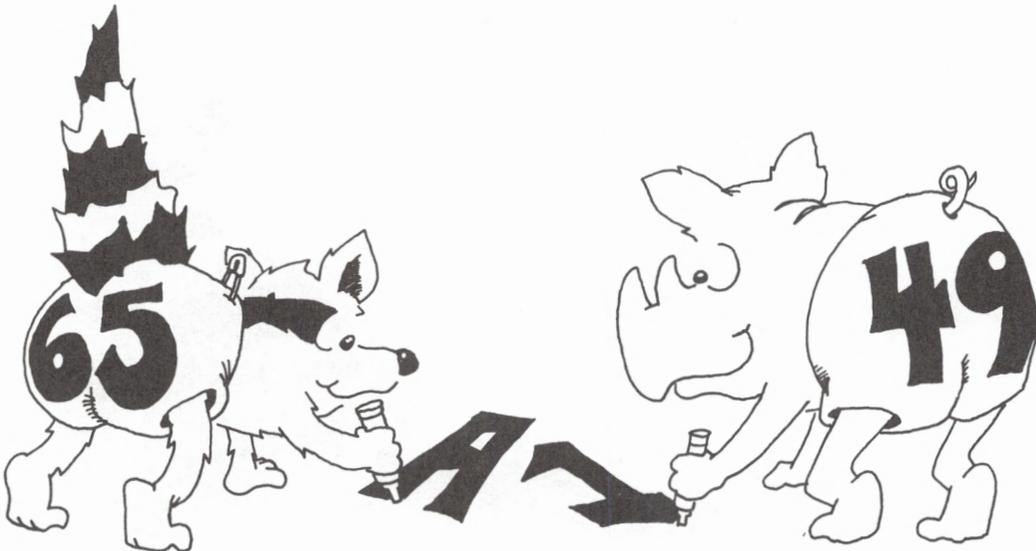
Save the program to disk.

Use CHR\$ to print the many ASCII characters that are not on the keyboard. They range from graphics like the smiley face and borders and blocks to foreign letters like Greek and German.

CHR\$ Is the Reverse of ASC

ASC gives you the ASCII number for the *first* character in the string.

CHR\$ does the reverse. It gives you the character belonging to the ASCII number.



The ASCII Numbers for Characters

Here are the groups of characters and their ASCII numbers:

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

Alphabetical List

These functions 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 Statement

The "Snake" program below uses the ON-GOTO statement.

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

If K is 1, GOTO 120
2, GOTO 130
3, GOTO 140
4, GOTO 150

If K is something else go on to 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 >>>>>
20 WIDTH 40
30 CLS
40 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-(Y=1):GOTO 160
130 X=X-1-(X=1):GOTO 160
140 Y=Y+1+(Y=24):GOTO 160
150 X=X+1+(X=40)
160 COLOR 7,0:LOCATE Y,X,0: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=20: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 3000:NEXT T
3200 CLS
3999 GOTO 100
```

Assignment 25

1. Write a program that 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\$, and LEN

In this lesson the functions LEFT\$, MID\$, RIGHT\$, and 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 operator (+) 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 function will snip the third and fourth letters out of a word.
7. Write a short program that takes the word *computer* and makes it into *putercom*.

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

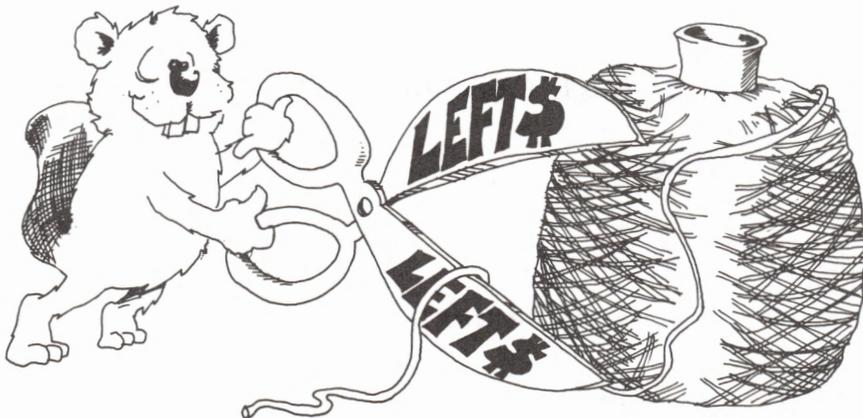
Concatenation means "make a chain." Maybe we should call them chains instead of strings.

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 parentheses.

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

Try another. Change line 40 to

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

Run the program again. This time the computer prints

```
789
```

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

More Snipping and Gluing

Run:

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

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

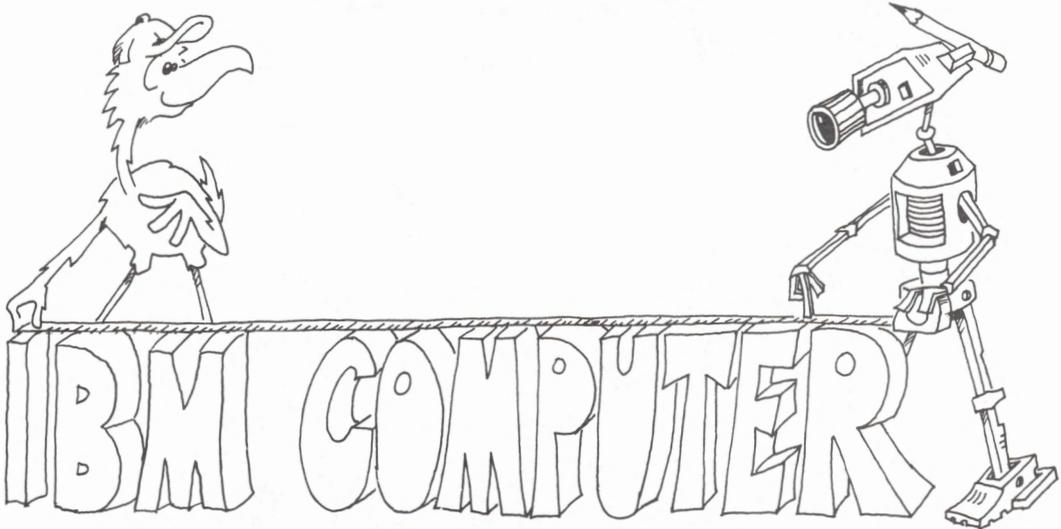
```
55 IF I=4 THEN PRINT: PRINT R$ + L$ :PRINT
```

How Long Is the String?

Run:

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

The function LEN tells the number of characters in the string. It counts everything in the string, even the spaces.



Cutting a Piece Out of the Middle

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

Run:

```
10 REM ### middle ###
20 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:

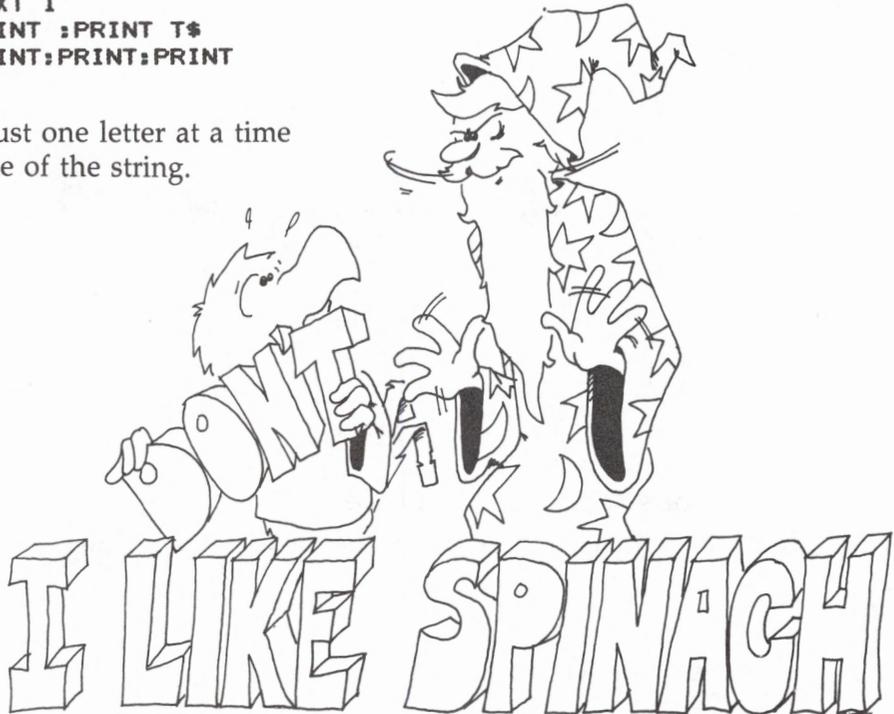
```
10 REM >>> no spaces >>>
11 REM
20 PRINT:PRINT
30 PRINT"Give me a long sentence":PRINT
```

```

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 just one letter at a time from 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 line and then change the < to a > and run again. The less than and greater than signs can be used to see if two strings are in alphabetical order.

Assignment 26

1. Write a secret cipher-making program. You give it a sentence, and it figures how long the sentence is. Then it switches the first letter with the second, the third with the fourth, and so on. For 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. Here's an 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 adds only *ay*. Examples:

box becomes *oxbay*
apple becomes *appleay*



Instructor Notes 27. Switching Numbers with Strings

This lesson treats two functions, STR\$ and VAL. It also gives a general review of the concept of a function.

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

VAL does just the opposite. It takes a string and makes a numeric value from it. It accepts decimals and scientific notation (for example, 1.2E+13). If the first character is not a decimal digit or + or -, it returns the value 0. Otherwise, it scans the number, terminating at the first nonnumeric character (other than the E of the scientific notation).

This conversion between the two main types of variables adds great flexibility to programs that involve 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 leapfrog by repeatedly putting its rear digit in the front.

A BASIC function returns a value—data in the form of a number or a string—to the expression in which it is used. You can also say that functions are called just as subroutines are called. The reason, of course, is that functions are implemented as subroutines on the machine code level.

Questions

1. If your number marches too quickly in the program of assignment 27, how do you slow it down?
2. Assume 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 birth date from the string and convert it to a number.)
3. What is a *value*. What is meant by "a function returns a value"? What are some of the things you can do with the value?

-
-
4. 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\$.

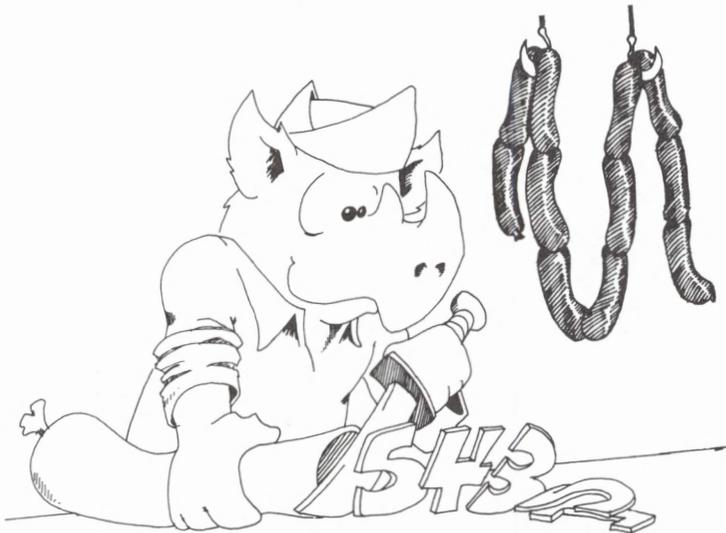
Making Strings into Numbers

We have two kinds of variables, strings and numbers. We can change one kind into the other.

Run:

```
10 REM Making strings into numbers
12 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
15 CLS
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, and CHR\$.

Rules About Functions

Functions always have parentheses with one or more arguments inside them. Here's an 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 (gives us some information). 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.

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

Practice with Functions

For each function in the list below, answer these questions:

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

Is each argument a variable, constant, or a function?

`RND(9)`

fn _____

arg _____

`INT(Q)`

fn _____

arg _____

MID\$(R1\$,E,2)

fn _____

arg _____

arg _____

arg _____

VAL(ER\$)

fn _____

arg _____

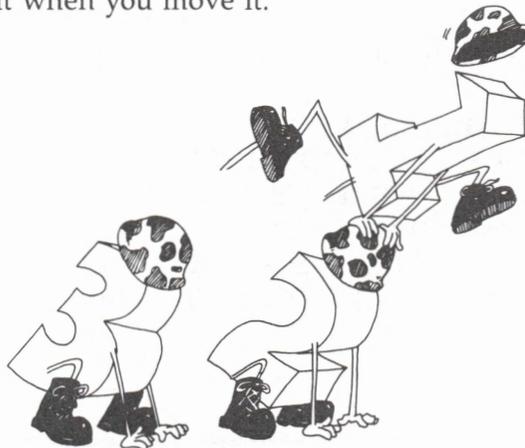
STR\$(INT(RND(8)))

fn _____

arg _____

Assignment 27

1. Write a program that asks for a number. Then it makes another number that is backward from the first and adds them together. It prints all three numbers like an addition problem (with a + sign and a line under the two numbers that are added).
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 right end. Keep repeating. Don't forget to erase each digit when you move it.



Instructor Notes 28. Logic: AND, OR, and NOT

This lesson treats the AND, OR, and NOT relations and the numeric values for *true* and *false*.

Two abstract ideas in this lesson may give difficulty. The first is that true and false have numeric values of -1 and 0 . Any expression in the form of an assertion (a phrase A) has a numeric value of 0 or -1 . This number is treated just like any other number. It can be stored in a numeric variable, printed, or used in an expression. Most often, it is used in an IF statement.

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

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

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

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

Questions

1. For each IF statement, tell if 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, and NOT

Another Teenager Program

Enter, save, and run this program:

```
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
"
```

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. The program says two things:

IF you are 12 OR you are 20 THEN you just missed being a teenager.

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

True and False Are Numbers

How does the computer understand true and false? It says true and false are numbers.

Rule: True is the number -1. False is the number 0.

(It is easy to remember that 0 is false because 0 is the grade you get if your homework is incorrect.)

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 0, meaning false. Try another example:

```
print 3=3
```

The computer checks to see if $3=3$. It does, so the computer prints -1 , meaning true.



Putting True and False in Boxes

The numbers for true and false are treated just like other numbers and can be stored in boxes with numeric variable names on the front. Run this:

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

The number 0 is stored in the box N because $3=22$ is false. Try 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 Statement Tells Little White Lies

The IF statement looks like this:

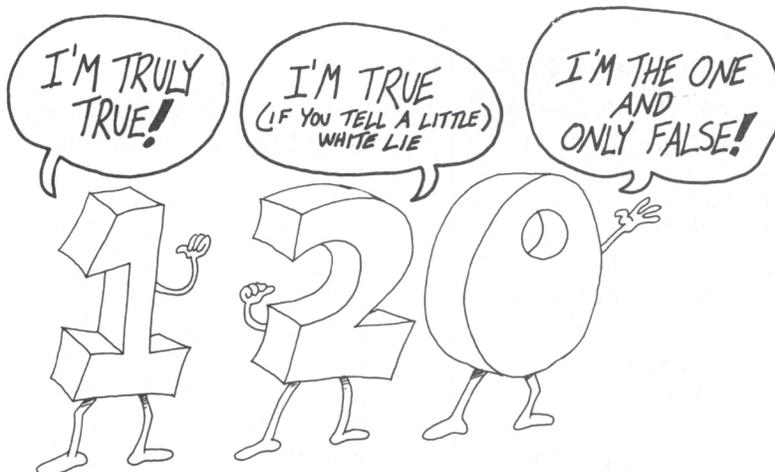
```
10 IF phrase A THEN command C
```

Try these in the edit mode:

```
if 0 then print "true"  
if -1 then print "true"  
if 22 then print "true"
```

What does the last entry print?

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



If it is 0, the computer says phrase A is false and skips what is after THEN.

If it is not 0, the computer says phrase A is true and obeys the commands after THEN.

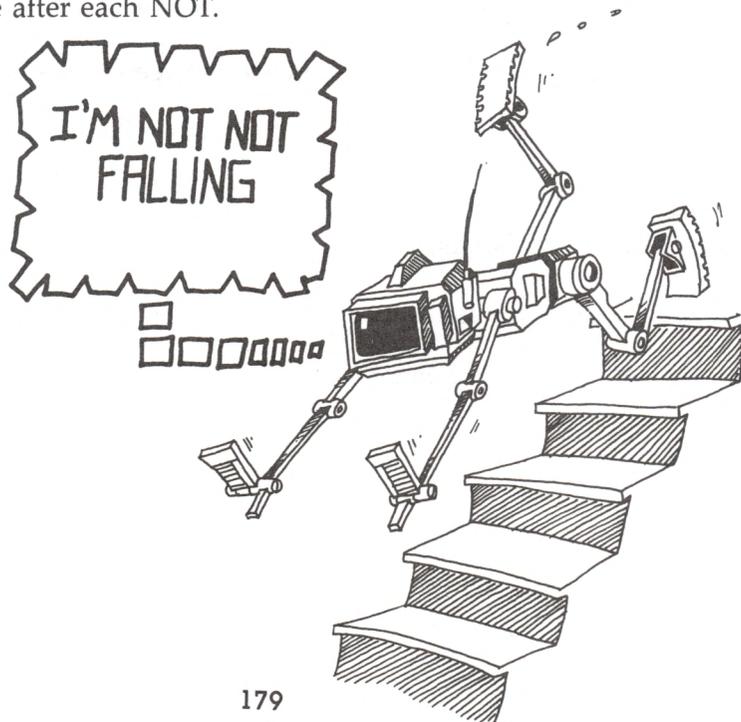
The IF statement tells little white lies. True is supposed to be the number -1 , but the IF stretches the truth to say *true is anything that is not false*; that is, any number that is not 0 is true.

What Does NOT Mean?

NOT changes false to true and true to false. Enter, run, and save 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 n  
  o ..."  
61 REM means "I do have ...."
```

Be sure to put a space after each NOT.



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 this:

$$20 N = 3$$

You do not get true or false as the numbers -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. To see the difference between < and <= look at these examples:

- 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's mean the same:

$$2<=Q \quad (2<Q) \text{ OR } (2=Q)$$

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 0 THEN PRINT "Jellybean"
IF 1 THEN PRINT "Jellybean"
IF 9 THEN PRINT "Jellybean"
IF 3<>0 THEN PRINT "Jellybean"
IF 2 AND 4 THEN PRINT "Jellybean"
IF 0 OR 1 THEN PRINT "Jellybean"
IF NOT 3 THEN PRINT "Jellybean"
IF "a"="z" THEN PRINT "Jellybean"
IF NOT (3) AND 2 THEN PRINT "Jellybean"
IF NOT (0) OR 0 THEN PRINT "Jellybean"
IF 4<=5 THEN PRINT "Jellybean"
```

3. Write a program to detect a double negative in a sentence. It looks for and counts the negative words like *not*, *no*, *don't*, *won't*, *can't*, and *nothing*. If there are two negative words, it's a double negative. Test the program on this 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\$ function lies just in this fact. 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 keypressing is required. This makes INKEY\$ useful in user-friendly programming.

The INKEY\$ function 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 an earlier lesson.

Questions

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:

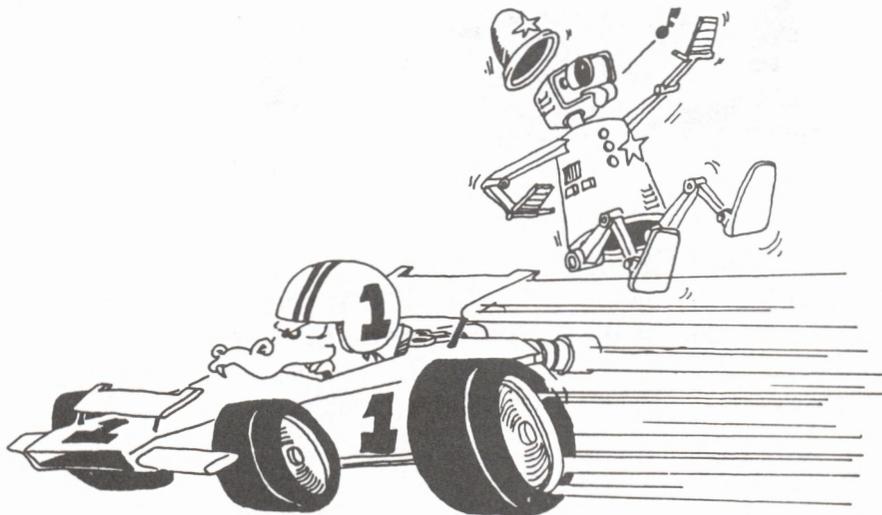
```
10 INPUT A$  
10 INPUT N
```

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 have finished entering.



The INKEY\$ Function

The INKEY\$ function 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 when you use INKEY\$: No question mark will show, no cursor will show, and 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
```

Use 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 the letter *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$="" THEN 10
20 PRINT K$
```

Now the computer is more polite! It keeps looking until a key is pressed.

Secret Writing

Use INKEY\$ in guessing games. You can enter a word or number to be guessed without other players being able to see it.

Run this program:

```
10 REM ---secret---
20 CLS
30 PRINT "Press any key"
40 K$=INKEY$: IF K$="" THEN 40
45 BEEP
50 PRINT "The key you pressed was ";K$
55 FOR T=1 TO 500:NEXT T
99 GOTO 20
```

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"
60 PRINT W$
```

Line 42 will not let the program continue until you press a key.

Making Words from Letters

The INKEY\$ function 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 should 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 a choice by typing a single letter. For example,

```
PRINT "Which color? R=Red, B=Blue, G=Green"
```

2. Write a sentence-making game. Each sentence has a subject, a verb, and an object. The first player types a subject (like *The donkey*). The second player types a verb (like *sings*). The third player types another noun (like *the toothpick*). Use INKEY\$ so that 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 in some professionals) is to impose structure on their programs. Structuring has gone by many names, such as *structured programming* and *top-down programming*, and employs 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 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 to be done. Only when the English description is itself sufficiently detailed does the BASIC programming begin. This is like *prewriting* an essay before attempting the complete version.

Of course, there is always some improving 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.

Some programmers advocate planning the whole program on paper before starting any coding at all. This may work for some people, but children are unlikely to adopt this style of working. Besides, if an advantage of word processors is 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 do you do (in English) to the outline next?
 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 word-guessing game where you draw part of a dragon each time you make a wrong guess for a letter.

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

The outline could be

```
10 REM *** Dragon 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 *** Dragon 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
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 to see if guess is right
--- if wrong, GOSUB 600:REM draw dragon part
--- if right, GOSUB 700:REM see if game is over
590 RETURN
599 REM
600 REM Add to the drawing
--- add to the dragon 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
--- show dragon melting your ice cream
890 RETURN
899 REM
900 REM End of game message
--- message for when guesser wins
990 RETURN
```

Once you have written the outline of the program, save it to disk.

Now it's 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 okay, take the STOP out. See lesson 33 for more information.)

Start by writing the subroutine at 300. 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 starts with a letter, and it can have letters, numbers, or periods in it.

The name must not be one of BASIC's reserved words like FOR, TO, IF, and STOP. There is a list of reserved words in an Appendix of this book.

But the name can have a reserved word inside it. For example,

FOR cannot be a variable name.
FORMULA is okay.
FOR.ME.TOO is okay.

Assignment 30

1. Finish the "Dragon Game." This is a long project. Start by writing the "get the word" subroutine. Then save it to disk. You may want to write one subroutine each day until the program is done.
2. Write a game you can play against the computer. Organize and outline it before programming it.



Instructor Notes 31. Arrays and the DIM Statement

This lesson introduces arrays and describes the DIM statement.

Arrays with one index are described first. The array itself is compared to a family, and the individual elements of the array to family members, with the index value being the first name of the family member.

Two-dimensional arrays are compared to the numbers on a month page of a calendar or to the rectangular array of cells on a TV screen.

The concept of arrays is not too difficult. The trick is to see how they help in programming. There is a large variety of uses for arrays, and many do not seem to fall into recognizable categories.

You can use arrays to store lists of information. Connected lists also can occur. The "Phone List" program uses two linear arrays—one for names, the other for numbers. They are indexed in the same way, so a single index number can retrieve both the name and the number that goes with it.

Another general use of arrays is to store numbers which cannot neatly be obtained from an equation. An example is the number of days in 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$(0) = "Dad"  
24 F$(1) = "Mom"  
26 F$(2) = "Bianca"  
28 F$(3) = "Matthew"
```

Each member of the family is a variable. The F\$ family are string variables.

Here is a family of numeric variables:

```
35 N(0) = 43  
37 N(1) = 13  
39 N(2) = 0  
41 N(3) = 0
```

The family has a "last name" like A or B\$. Each member has a number in parentheses for a "first name." The array always starts with the first name being the number 0.

Instead of family we should say *array*.

Instead of first name we should say *index number*,
or *subscript*.



The DIM Statement Saves Boxes

When the array family goes to a movie, they always reserve seats first. They use a DIM statement to do this.

The DIM statement tells the computer to reserve a row of boxes for the array. DIM stands for *dimension*, which means *size*.

For example, the statement

```
18 DIM A(3)
```

saves four memory boxes, one each for the variables A(0), A(1), A(2), and A(3). These boxes are for numbers and they contain the number 0 to start with. Here's 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\$(0) through B\$(4) are for strings and are empty to start with.

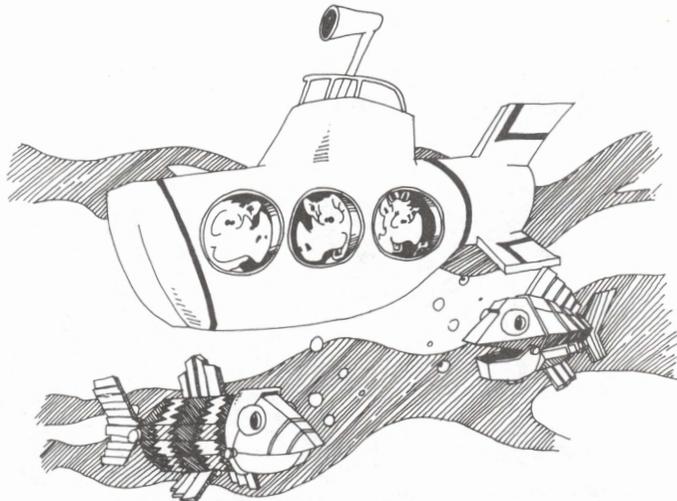
Rule: Put the DIM statement very early in the program, before the array is used in any other statement.

Making a List

Enter:

```
10 REM +++ In a row +++
20 CLS:PRINT
30 DIM A$(3)
35 PRINT"Enter a word"
40 FOR N=0 TO 3
45 IF N>0 THEN PRINT"Another"
50 INPUT A$(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=0 TO 3
120 PRINT A$(I);" ";
130 NEXT I
```

Save the program before you run it.



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:

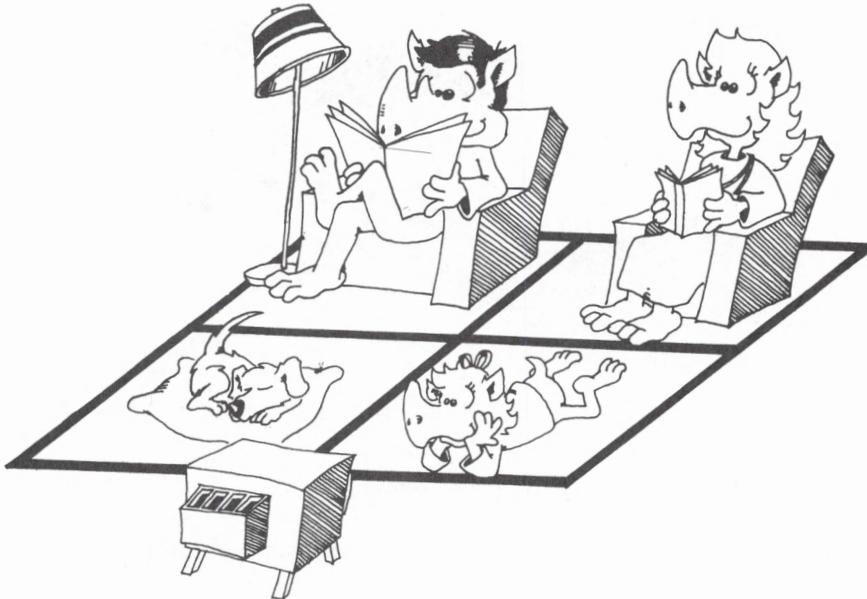
```
10 REM Phone list
20 CLS:PRINT
30 DIM NA$(20),NU$(20)
35 I=0
40 PRINT "enter names and numbers"
50 PRINT : INPUT "Name?",NA$(I)
60 INPUT "number?",NU$(I)
70 I=I+1:GOTO 50
```

Save and run the program.

One Dimension, Two Dimension, . . .

Arrays that have one index are called *one-dimensional* arrays.

But arrays can have more than one index. Two-dimensional arrays have their family members put in a rectangle like the days in a month on a calendar.



Eight Queens

The "Eight Queens" puzzle asks you to put eight chess queens on the chessboard in such a way that no queen is attacked by any other. If you are not familiar with chess, look up the moves of the queen in an encyclopedia. Obviously, you can't have two queens on the same row or column. Queens attack along the diagonal also. There are 92 patterns of queens on the board that solve this puzzle.

```
1 REM 8 queens
2 GOTO 1000
100 REM main loop
115 M=R(I)
120 M=M+1:IF M=9 THEN GOSUB 600:GOTO 115
130 IF B(I,M)=0 THEN GOTO 700
140 GOTO 120
200 REM
201 REM update attacked squares
```

```

202 REM
210 FOR L=1 TO 8
215 B(I,L)=B(I,L)+D
220 B(L,J)=B(L,J)+D
225 NEXT L
500 REM
501 REM diagonal
502 REM
510 FOR K=1 TO 8
515 X=I+K
520 IF X>8 THEN 530
522 Y=J+K: IF Y>8 THEN 525
523 B(X,Y)=B(X,Y)+D
525 Y=J-K: IF Y<1 THEN 530
526 B(X,Y)=B(X,Y)+D
530 X=I-K
535 IF X<1 THEN 590
540 Y=J+K: IF Y>8 THEN 550
545 B(X,Y)=B(X,Y)+D
550 Y=J-K: IF Y<1 THEN 590
555 B(X,Y)=B(X,Y)+D
590 NEXT K
595 B(I,J)=Q
599 RETURN
600 REM
601 REM go back
602 REM
610 R(I)=0
612 I=QN
615 IF I=0 THEN END
620 J=R(I)
630 D=-1:Q=0:GOSUB 200
640 QN=QN-1
690 REM gosub 900
699 RETURN
700 REM
701 REM go ahead
702 REM
710 R(I)=M:J=M
715 QN=QN+1
720 D=1:Q=-1
730 GOSUB 200
735 NM=NM+1
740 IF QN=8 THEN GOTO 800
780 I=I+1
785 KB=1
786 IF KB<>0 THEN GOSUB 900
790 REM gosub 900
799 GOTO 115
800 REM

```

```

801 REM solution
802 REM
810 BEEP
814 NS=NS+1
815 GOSUB 900
840 GOSUB 612
899 GOTO 115
900 REM
901 REM display
902 REM
910 FOR X=1 TO 8
915 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-4);"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 digit
s on
1024 ' NUMERICAL keypad, then let up on Alt ke
y
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);"  ♀♀♀♀♀♀♀♀♀♀
♀"
1040 FOR I=1 TO 8
1045 PRINT TAB(T);"  ♀♀♀♀♀♀♀♀♀♀"
1046 PRINT TAB(T);"  ♀♀♀♀♀♀♀♀♀♀"
1047 NEXT I=1
1048 LOCATE 19,3:PRINT TAB(T);"  ♀♀♀♀♀♀♀♀♀♀
♀"
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 I column is located. $B(I,J)$ 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 $1, 2, 3$, and so on, tell how many queens are attacking that 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 for the month (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 the "Eight Queens" program to work for a standard 8×8 chessboard. Change the program so that the user can choose any size board.
4. Change the 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 lesson presents a format for writing programs. While different methods of imposing order on the task are largely a matter of taste, the methods used here can serve to introduce the ideas.

User-friendly means that screen displays are easy to read, keyboard input is Enter-key-free as much as possible, and errors are trapped. The program should ask if entries are okay. If not, give an opportunity to fix things.

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

It is hard to teach the writing of user-friendly programs. Success depends mostly on the programmer's attitude. The best advice is to "turn up your annoyance detectors to high" as you write and debug a program.

Most young students will not progress very far toward user-friendly programming. Just becoming acquainted with the desirability of friendly programming and using some simple techniques toward accomplishing that goal are satisfactory achievements.

Questions

1. Should your program give instructions whether the user wants them or not?
2. What is a *prompt*? Give two examples.
3. What is *scrolling*? How can you write to the screen without scrolling?
4. How do you allow the user to enter a single letter from the keyboard without using the Enter key?

-
-
5. What is an *error trap*? How would you trap errors if you ask your user to enter a number from 1 to 5?
 6. In what part of the structured program are most of the GOSUB statements found?
 7. Why put the “starting stuff” section of the program at the end of the program (at high line numbers)?

Lesson 32. User-Friendly Programs

There are two kinds of users:

1. Most want to run the program. They need:

- instructions
- prompts
- clear writing on the screen
- no clutter on the screen
- erasing old stuff from the screen
- not too much keypressing
- protection from their own errors

2. Some want to change the program. They need:

- a program made in parts
- each part with a title in a REM
- explanations in the program

(Don't forget you are a user of your own programs, too. Be kind to yourself!)



Programs Have Three Parts

Starting Stuff:

gives instructions to the user
draws a screen display
sets variables to their starting values
asks the user for starting information

Main Loop:

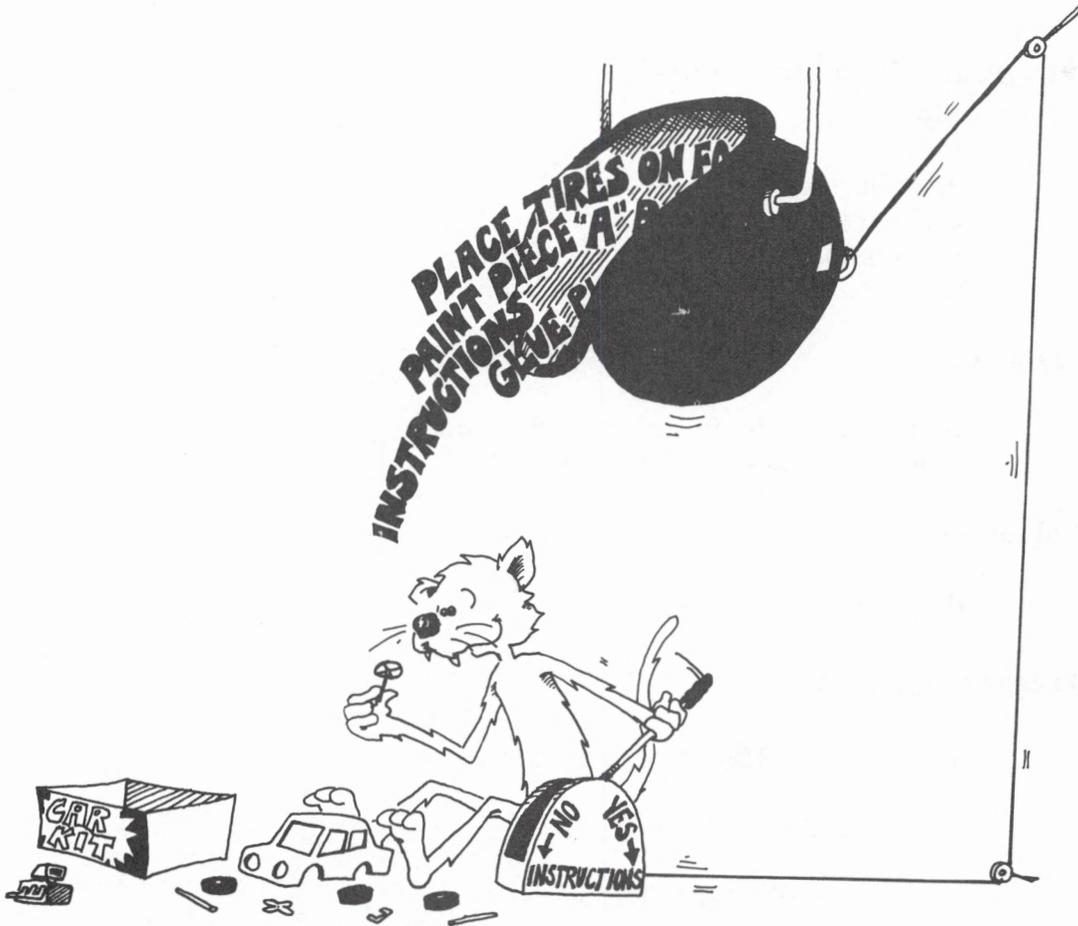
controls the order in which tasks are done
calls subroutines to do the tasks

Subroutines:

do parts of the program

Program Outline

```
1 GOTO 1000:REM *** program name ***  
---  
100 REM MAIN LOOP  
---  
---      calls subroutines  
---  
199 END  
1000 REM  
1001 REM *** program name ***  
1002 REM  
---      REMs that give a description of the  
---      program, variable names, etc.  
---  
1999 REM  
2000 REM STARTING STUFF  
---  
---      ask for starting information  
---      set variable values  
---      give instructions  
---  
2999 GOTO 100
```



Put the Main Loop at the Beginning of the Program

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

Put Starting Stuff at the End of the Program

Put the starting stuff near the back because it may be the biggest part of the program, and you may keep adding to it as you write to make the program more user-friendly. It does not need to run fast.

Put Subroutines in Three Places

Subroutines that must run fast should go between lines 2 and 99, starting stuff subroutines should go after line 2999, and the rest of the subroutines should be put between lines 200 and 999.

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 of what choices they have.

Example: Use <Y/N> or something similar 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 just where you want. Erase by printing a string of blanks to the same spot.

Use delay loops so the writing stays on the screen while the user reads it.

Ouch! My Fingers Hurt

Use the INKEY\$ function 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

Add this line to the above lines:

```
386 IF R$<> "n" THEN 380
```

Line 380 asked for only two choices, Y or N. If users press some other key, line 386 sends them back to line 380.

Traps make your program *bomb-proof* so that users will be unable to goof it up!

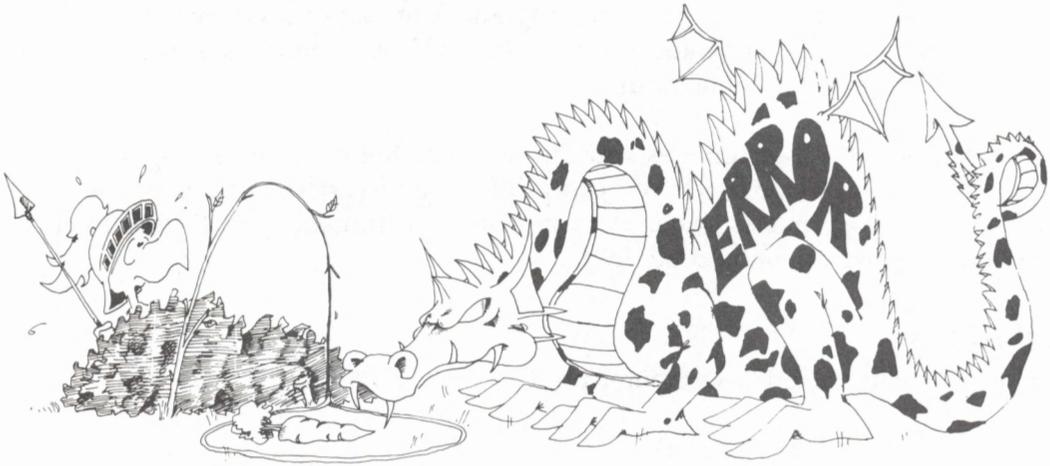


Assignment 32

1. Make a program that writes 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:

DRAGONETBCFH IJKLMPQSUVWXYZ = Cipher alphabet
ABCDEFGHIJKLMN OPQRSTUVWXYZ = Normal alphabet

The user chooses to encode or decode from a menu.



Instructor Notes 33. Debugging: STOP and CONT

Since the sigh-and-moan technique is a loser, our students need a bag of tricks that help isolate program bugs. They should practice on the programs they write as they go through this book.

The inexperienced debugger feels hopeless when a program doesn't work right. Rather than sitting and staring, it is more useful to try some changes. Any changes are better than none, but random changes are very inefficient. The best changes are those that eliminate sections of the program from the list of possible hiding places for the bug.

As programs grow in complexity, more of the bugs result from unforeseen interactions between separate parts of the program. The bag of tricks we offer helps find these also. Delay loops and PRINT 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 the Break key, STOP, or END. You can print out any variable values you like to see what the program has done. You can also do arithmetic with PRINT to check for what the program should be doing.

Questions

1. How can you make the computer print

Break in line 55

by adding a statement to the program?

2. How are the STOP and END statements different?
3. How are the STOP statement and Break key different?
4. What does the CONT command do?
5. Why would you put STOP statements in your program?

-
-
6. How do delay loops help you debug a program?
 7. How do extra PRINT statements help you debug a program?
 8. Why do you remove the STOP and extra PRINT statements from the program after you have fixed the errors?
 9. Can you pick in what line the Break key will stop the program? Can you pick using the STOP statement?

It is like END except it prints the number of the line in which the program quit running.

You can have as many STOP statements in your program as you like.

STOP is used for debugging your program.

Another Way to Stop Running the Program

You can stop running the program with the Break key sequence as well.

Try it:

```
10 REM GO FOREVER
15 CLS:PRINT:PRINT:PRINT
16 GOSUB 90
20 PRINT:PRINT"MUD"
21 GOSUB 90
22 PRINT:PRINT"  TURTLES"
23 GOSUB 90
24 PRINT:PRINT"    OF"
25 GOSUB 90
26 PRINT:PRINT"      THE"
27 GOSUB 90
28 PRINT:PRINT"        WORLD"
29 GOSUB 90
30 PRINT:PRINT"          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 by using the up cursor key and just changing the line number. Also, all the lines starting PRINT:PRINT" can be done by using the up cursor, changing the line numbers, and changing the word at the end.)

Run the program. Hold down the Ctrl key (on the PCjr use the Fn key) and press the Break key. This stops the program where it is. It prints

```
^C
Break in XX
```

Then it enters the edit mode. (The XX in the *Break* message above will be the line number where the program quit running.)

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 run the program. After it stops, look to see what happened.

(Or you can use the Break key to stop the program, but it may not stop in the spot where the trouble is.)

Put on your thinking cap. Ask yourself questions about what happened as the program ran.

You are in the edit mode. You can list parts of the program and study them. Use the PRINT statement to look at variables. Do they have the values you expected? Do little calculations on the computer in the edit mode to check what the computer is doing.

Use the LET statement to change the values of variables.

If you find the trouble, you may add a line, change a line, or delete a line.

Starting the Program Again

There are four ways to restart 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	to try again.

You may use the CONT command if you have not used Break to stop the program, 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 the GOTO statement.

If you have changed the program, your only choice is to start at the beginning or at some other line number 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 and then execute the program. RUN XX starts at line XX.

CONT can only restart a program that was stopped with a break from a STOP or a Break key. But RUN, RUN XX, and GOTO XX can also start a new program.

Debugging

Little errors in your program are called *bugs*.

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

1. If the computer printed an error message, it tells what line it stopped on. Careful, the mistake may really be in another line!
2. If the computer just keeps running but doesn't do the right thing, stop it and put some PRINT lines in that will tell what is happening.
3. Put STOP statements in the program.
4. If the program runs so fast that you can't tell what is happening, put in some delay loops to slow it down.

After you have fixed the program, take the PRINT lines, the STOP statements, and the delay loops out of the program.

Assignment 33

1. Go back to the "Snake" program (lesson 25) and fix up some of the bugs. For example, the program crashes when the snake hits a wall. Add food for the snake. Add scorekeeping. Let the game end if the snake touches a wall.
2. Go back and fix up some other program that you have written.

Appendix A. Disk Use

Disk Care

The student should be instructed on the care of disks. Especially:

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



Preparing a Disk for the Student

Using the computer will be much easier for your student if you prepare a special disk. The disk will automatically start up the computer, set the screen, list the files on the disk, and leave the student in BASIC, ready to write programs or load files.

The instructions below are for DOS 2.1. If you have another version of DOS, refer to your DOS manual.

Place the DOS disk in the disk drive and turn on the computer (on PC systems with two disk drives, use drive A, the left drive). 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 disk 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 (or one that contains information you no longer need) in the drive and press a 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
```

Also, you will be told the total disk space and the available disk space. These messages vary slightly depending on the model of computer you are using.

The computer will also ask:

```
Format another (Y/N)?
```

Answer no by pressing the N key. You will again see the prompt

A>

(PCjr users, skip the next section about BASICA and go to "Student Program for Start-up.")

Now we need to put the "Basica" file on the disk. This section assumes you are using a PC with two disk drives, the most common configuration. Remove the student disk from drive A and put it in drive B. Put the DOS disk back in drive A. Then enter

```
copy basica.com b:
```

The drive will run, then the computer will print

```
1 File(s) copied
```

In a similar way, copy the "Colorbar.bas" file from the *DOS 2.1 Supplemental Programs* disk to the student's disk.



Student Program for Start-up

We want the student's disk to prepare the computer for the student automatically. So we will put an "Autoexec.bat" file on the disk, which will automatically load and run a BASIC program named "student". Insert the student's disk in the drive (drive A if you have two drives) and enter this:

```
copy con: autoexec.bat  
basica student
```

Now press the F6 key (PCjr: first, press the Fn key). You will see the prompt

^Z

Press the Enter 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 from the disk (or activate the cartridge of the PCjr). If the screen is hard to read, enter

screen 0,1

Then enter this program, putting your own student's name in line 70.

```
30 SCREEN 0,1:REM if you are using a TV or non
  -IBM monitor
50 WIDTH 40:REM omit if you have a phosphor (g
  reen) screen monitor
60 CLS:PRINT:PRINT:PRINT
70 PRINT TAB(13);"Student 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 first. Then enter

save "student"

This is the student's own disk, which will boot the system and will contain all the programs the student writes. If you have more than one student, you may want to have them put their names in line 70 of the above program.



Appendix B. IBM BASIC Reserved Words

Reserved words discussed in this book are in bold.

ABS	DEFDBL	HEX\$	MERGE
AND	DEFINT		MID\$
ASC	DEFSNG	IF	MKDIR
ATN	DEFSTR	IMP	MKD\$
AUTO	DELETE	INKEY\$	MKI\$
	DIM	INP	MKS\$
BEEP	DRAW	INPUT	MOD
BLOAD		INPUT#	MOTOR
BSAVE	EDIT	INPUT\$	
	ELSE	INSTR	NAME
CALL	END	INT	NEW
CDBL	ENVIRON	INTER\$	NEXT
CHAIN	ENVIRON\$	IOCTL	NOT
CHDIR	EOF	IOCTL\$	
CHR\$	EQV		OCT\$
CINT	ERASE	KEY	OFF
CIRCLE	ERDEV	KEY\$	ON
CLEAR	ERDEV\$	KILL	OPEN
CLOSE	ERL		OPTION
CLS	ERR	LEFT\$	OR
COLOR	ERROR	LEN	OUT
COM	EXP	LET	
COMMON		LINE	PAINT
CONT	FIELD	LIST	PEEK
COS	FILES	LLIST	PEN
CSNG	FIX	LOAD	PLAY
CSRLIN	FN	LOC	PMAP
CVD	FOR	LOCATE	POINT
CVI	FRE	LOF	POKE
CVS		LOG	POS
	GET	LPOS	PRESET
DATA	GOSUB	LPRINT	PRINT
DATE\$	GOTO	LSET	PRINT#
DEF			PSET
			PUT

RANDOMIZE
READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT\$
RMDIR
RND
RSET
RUN

SAVE
SCREEN
SGN
SHELL
SIN
SOUND
SPACE\$
SPC
SQR
STEP
STICK
STOP
STR\$
STRIG
STRING\$

SWAP
SYSTEM

TAB
TAN
THEN
TIMER
TIME\$
TO
TROFF
TRON

USING
USR

VAL
VARPTR
VARPTR\$
VIEW

WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
WRITE#

XOR

Appendix C. Answers to Selected Assignments

Assignment 1.3

```
10 REM Greeting
20 PRINT " Hi There,"
30 PRINT " Computer"
```

Assignment 2.1

```
10 REM Names
14 SCREEN 0,1
15 COLOR 0,1
20 PRINT " Minda"
30 PRINT " Anne"
40 PRINT " Carlson"
50 COLOR 7,0
```

Assignment 3.5

```
10 REM Birds
15 CLS
20 PRINT
22 BEEP
25 PRINT " ---0---"
30 PRINT
40 PRINT
42 BEEP
50 PRINT "      ---0---"
60 PRINT
70 PRINT
75 BEEP
80 PRINT "  ---0---"
```

Assignment 4.3

```
10 REM Smile
12 CLS
20 PRINT
30 PRINT
40 PRINT
50 PRINT "      00      00      "
60 PRINT
61 PRINT
62 PRINT
63 PRINT " *                * "
64 PRINT " *                * "
65 PRINT " *                * "
66 PRINT " ***** " " "
```

Assignment 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!"
```

Assignment 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$"
52 PRINT
55 PRINT "It is:"
57 PRINT
60 PRINT C$
```

Assignment 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$;" plays ";T$
```

Assignment 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%
```

Assignment 8.3

```
10 REM Teen Power
11 REM
15 COLOR 5,0
20 PRINT "Teen Power"
21 PRINT
22 PRINT
23 PRINT
30 GOTO 20
35 REM press break key to stop
```

Assignment 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 Break key to stop program
99 GOTO 20
```

Assignment 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!!!"
```

Assignment 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 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$= "y" THEN 70
66 IF Y$<>"n" THEN 60
67 B=B-1
70 PRINT
75 PRINT "You were born in";B;"."
```

Assignment 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
```

Assignment 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
```

Assignment 11A.2

```
10 CLS
20 B$="!!!!"
30 C$="----- how are you?"
40 PRINT "What is your name?"
50 INPUT A$
60 CLS:BEEP
70 PRINT A$;B$;TAB(15);C$
```

Assignment 11B

```
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
```

Assignment 12B.3

```
10 REM I got your number!
20 CLS
25 PRINT
26 PRINT
27 PRINT
30 PRINT "Give me a number between zero and te
n:"
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"
```

Assignment 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
```

Assignment 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 "Press Ctrl-Break to end game"
27 PRINT
28 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 end the game by pressing
91 REM      'Break' key

```

Assignment 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 b
   eautiful Hog Wallow, Mich.":GOTO 90
42 IF D<5 THEN PRINT "Enter a pickle eating co
   ntest 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

```

Assignment 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
```

Assignment 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 SCREEN 0,1:COLOR C,0
35 LOCATE Y,X:PRINT N$
45 FOR T=1 TO 500:NEXT T
50 NEXT S
```

Assignment 16.2

```
10 REM NAMEX
20 WIDTH 40:CLS
30 INPUT "NAME";N$:N$=LEFT$(N$,13)
40 CLS
50 FOR X=1 TO 10
60 LOCATE X,X+5:PRINT N$
70 NEXT
80 FOR X=1 TO 10
90 LOCATE X,29-X:PRINT N$
100 NEXT
```

```
110 LOCATE 11,17:PRINT N$
120 FOR X=1 TO 10
130 LOCATE X+11,16-X:PRINT N$
140 NEXT
150 FOR X=1 TO 10
160 LOCATE X+11,X+18:PRINT N$
170 NEXT
```

Assignment 17A

```
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
```

Assignment 17B.2

```
10 REM Your name is falling
20 CLS
25 PRINT "Your name"
27 PRINT
30 INPUT N$
33 CLS
35 FOR I=1 TO 30 STEP 2
40 PRINT TAB(I);N$
42 FOR T=1 TO 200:NEXT T
45 NEXT I
```

Assignment 17B.4

```
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$
36 PRINT F$
38 PRINT
40 FOR T=1 TO 300:NEXT T
50 NEXT I
```

Assignment 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
```

Assignment 19.2

```
10 REM DATA MUSIC
20 READ A
30 IF A=-1 THEN END:REM END OF TUNE
40 SOUND A,2
50 FOR T=1 TO 100:NEXT T:REM DELAY
60 SOUND 32767,2:REM REST
70 GOTO 20
80 DATA 502,502,742,742,842,842,742,-1
```

Assignment 20B.3

```
10 REM Sinbad's Magic Carpet
11 RANDOMIZE
12 SCREEN 0,1: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 "■"
121 LOCATE 22-J, I:PRINT "■"
122 LOCATE 22-J,31-I:PRINT "■"
123 LOCATE J, 31-I:PRINT "■"
190 NEXT J:NEXT I
195 FOR I=1 TO 9999:NEXT I
199 COLOR 7,0,0
```

Assignment 20B.4

```
10 REM cross my heart
20 SCREEN 0,1:COLOR 4,7
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"
80 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
```

Assignment 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$)
39 K=1
40 FOR I=97 TO 97+26
41 REM test letters in alphabet
42 REM tosee if in word
45 FOR J=1 TO L
50 G=ASC(MID$(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$
```

Assignment 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$
```

Assignment 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$
```

Assignment 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 +1 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)
```

Assignment 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
```

Assignment 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
```

Assignment 27.2

```
10 REM leapfrog
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
```

Assignment 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 apostrophe":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

```

Assignment 29.1

```

10 REM menu maker
12 SCREEN 0,1
20 PRINT"which color do you like?"
21 PRINT
22 PRINT" <y> yellow"
23 PRINT" <g> green"
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$="g" THEN COLOR 0,2:CLS
37 IF X$="b" THEN COLOR 0,1:CLS
40 GOTO 12

```

Assignment 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 sente
nce":PRINT
115 PRINT"no player can see what the other ent
ers":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

```

Assignment 31.1

```

10 REM Month
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);"day
s."
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31

```

Assignment 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
```

Assignment 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":PR
    INT
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

```

Appendix D. Glossary

argument

The variable, number, or string that appears in the parentheses of a function. N is the argument of the integer function in $\text{INT}(N)$.

array

A set of variables that have the same name. The members of the array are numbered. The numbers appear in parentheses after the variable name. *See also* subscript. Examples:

$A(0)$ is the first member of the array A.

$B\$(7)$ is the eighth member of the array B\$.

$CD(3,M+1)$ is a member of the two-dimensional array CD.

arrow keys

Four keys on the computer have arrows on them. They move the input cursor to the left and right, up and down.

ASCII

American Standard Code for Information Interchange. Each character has an ASCII number.

assertion

The name of a phrase that can be *true* or *false*. The phrase A in an IF statement is an assertion. An assertion has a numeric value of 0 or -1 . *See also* expression, false, logic, phrase A, true.

background

The part of the screen that is blank, having no characters on it.

BASIC

Beginner's All-purpose Symbolic Instruction Code. A computer language originated by John Kemeny and Thomas Kurtz at Dartmouth College in the early 1960s.

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

To start up the computer from scratch—the computer “picks itself up by its own bootstraps.” 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-THEN statement is a branch. So is an ON-GOTO statement. A branch is not the same as a jump (where there is no choice). *See also* jump.

buffer

A storage area in memory for temporary storage of information being sent from the computer or received from another device.

call

Using a GOSUB calls a subroutine. Putting a function in a statement calls the function. The computer does what statements are in the subroutine or performs the function and then returns to just beyond the calling spot.

carriage return

On a typewriter, you push the lever that moves the carriage that carries the paper so a new line can begin. In computing, the cursor is moved to the start of the line, but not down to the next line. *See also* CRLF, linefeed.

character

Letters, digits, and punctuation marks 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 also* I/O.

clear

To erase. Used in terms like *clear the screen* and *clear memory*.

column

Material arranged vertically. *See also* row.

command

In BASIC a command makes the computer perform some action, usually in edit mode. Examples are RUN, LIST, and LOAD. *See also* expression, statement.

concatenation

Sticking two strings together.

constant

A number or string that does not change as the program runs. It is stored in the program line, not in a box with a name on the front. *See also* line.

CRLF

Short for *Carriage Return with Linefeed*. On a typewriter, it's just called a carriage return. *See also* carriage return, linefeed.

cursor

A marker that shows where the next character on the screen or in a storage buffer will be placed. Cursor means "runner." The cursor runs along the screen as you type. There are two kinds of cursors in the IBM computer: the input cursor (a flashing line) and the print cursor (invisible).

data

BASIC has two kinds of data: numeric and string. Logical data (true, false) are types of numeric data.

debug

To run a program to find the errors and fix them. You fix the errors by editing the program. *See also* edit.

delay loop

A part of the program that just uses up time and does nothing else. Example:

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

duration

A number in a SOUND statement that tells how long the sound will last.

edit

In line and program editing, you retype parts of the line or program to correct it.

edit mode

When Ok and the flashing cursor show, you are in the edit mode. The computer awaits commands or program line entry.

enter

To put information into the computer by typing, then pressing 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 also* clear.

error trap

Part of a program that checks for mistakes in information that the user has entered or that checks to see if computed results are within reasonable bounds.

execute

To run a program or perform a single command or statement.

expression

A portion of a statement that has a single value, either a number or a string. *See also* value.

false

The number 0 represents false. *See also* assertion, logic, true.

fork in the road

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

function

BASIC has a number of functions built-in. Each function has a name followed by parentheses. In the parentheses is one or more arguments. The function has a single value (numeric or string) determined by its arguments. *See also* argument, value.

garbage

A random mess of characters in memory. Usually due to human or machine error.

graphics

Computer 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*. In $Q(7, J)$, 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 center and a *fire* button.

jump

GOTO makes the computer jump to another line in the program rather than execute the next line.

line

Program lines start with a number followed by a statement, or statements. Direct or edit mode lines do not start with a number.

line buffer

The storage space that receives the characters you type in. *See also* buffer.

linefeed

Moving the cursor straight down to the next line. The ASCII number 10 signals this command to the screen or printer. *See also* carriage return, CRLF.

line number

The number at the beginning of a program line. The line number tells the computer where to store the line.

listing

A list of all the lines in a program.

load

To transfer the information in a file on tape, disk, or other memory device 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 also* assertion, phrase A.

loop

A part of the program that is done over and over again. There are many kinds of loops, most notably FOR-NEXT loops.

loop variable

This is the number that changes as the loop is repeated. For example, in `40 FOR I=1 TO 5: NEXT I`, I is the loop variable.

memory

The part of the computer where information is stored. Memory is made of semiconductor chips, but in this book we think of it as boxes with labels on the front and information inside. Some memory is permanent; it is called *ROM* (*Read Only Memory*). The temporary kind of memory, where programs you type in are stored, is called *RAM* (*Random Access Memory*).

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 one.

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 that is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a program that allows you to write and examine machine language code.

nesting

When one thing is inside another. In programs, we often nest loops. Inside a statement, we can nest expressions or functions.

number

One type of information in BASIC. The other type is *string*. The numbers are generally decimal numbers. *See also* integers, string.

operation

In arithmetic: addition, subtraction, multiplication, and division, with symbols +, −, *, and / respectively. The only arithmetic operation for strings is concatenation.

phrase A

A phrase that stands for an assertion in an IF statement in this book. *See also* assertion. In the following line, $A > 4$ is phrase A:

```
IF A>4 THEN 500
```

pitch

The number in a SOUND statement that tells the number of cycles per second of the sound (how high or low it sounds).

pixel

A loose abbreviation for *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 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

This is a little message you put on the screen with an INPUT to remind the user what kind of answer you expect. Its name comes from the hint that actors in a play get from the prompter if they forget their lines.

pseudorandom number

A number that is calculated in secret by the computer using the RND function. It is usually called a random number. *Pseudorandom* emphasizes that the number

really is not random (since it is calculated by a known method) but is just not predictable by the user of the computer.

punctuation

Characters like the period, comma, /, ?, !, \$, and so on.

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 ten 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 called, its spot in the expression is replaced with a value (a number or a string). This is called returning a value.

row

Material arranged horizontally (across). *See also* column.

run mode

The action of the computer when it is executing a program is called operating in the run mode. You get into the run mode from the edit mode by entering run. When the computer ends the program for any reason, it returns to the edit mode.

save

To put the program that is in the computer's memory on disk or other memory storage.

screen

The monitor screen (similar to a TV screen) that is hooked up to the computer. *See also* monitor.

scrolling

The usual way the computer writes to the full screen is to put the new line at the bottom of the screen and push all the old lines up. This is called scrolling.

simple variable

A variable that is not an array variable.

stack

A memory area where the computer temporarily stores data, such as the return location for a GOSUB statement. In many cases, the last data item put on the stack is the first one taken off. This is called a *LIFO stack*, for last-in, first-out.

starting stuff

The name given in this book to initialization material in a program. It includes adding REMs for describing the program, input of initial values of variables, setting up array dimensions, drawing screen graphics, and any other things that need to be done just once at the beginning of a program run.

statement

A reserved word that usually begins each program line. Examples are PRINT, INPUT, and READ.

store

To put information in memory or to save it on disk or other long-term memory-storage device.

string

A type of data in BASIC. It consists of a group of characters. *See also* number.

subroutine

A section of a program that starts with a line called from a GOSUB statement and ends with a RETURN statement. It may be called from more than one place in the program.

subscript

Another name for *index*. A number in the parentheses of an array. It tells which member of the array is being used. *See also* index.

syntax

The way a statement in BASIC is spelled. A syntax error means the spelling of a command or variable name is wrong, the punctuation (including spacing) is wrong, or the order of parts in the line is wrong.

timing loop

A loop that does nothing except use up a certain amount of time. *See also* delay loop.

title

The name of a program or subroutine. Put it in a REM statement.

true

Has the value -1 . *See also* assertion, false, logic.

typing

Pressing keys on the computer. It is different from entering. *See also* enter.

value

The value of a variable is the number or string stored in the memory box belonging to the variable. *See also* variable.

variable

A name given to an imaginary box in memory. The box holds a value. When the computer sees a variable name in an expression, it goes to the box and takes a copy of what is in the box back to the expression and puts it where the variable name was. Then it continues to evaluate the expression. *See also* variable name.

variable name

A variable is either a string variable or a numeric variable. The name tells which. String variables have names ending in a dollar sign. Numeric variables do not. The variable name 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.

Topical Index

- arithmetic operations (addition, division, multiplication, subtraction) 58, 60
- arrays 192, 193–96
 - one-dimensional 192, 196
 - two-dimensional 196
- ASCII code for characters 106, 156, 157, 159
- beeps 10
- bugs in a program 213
- characters 12
- clearing the screen 11
- colon 93, 97–99
- color
 - background 127
 - border 127
 - character 127
- coloring the screen 11–12
- concatenation 41, 45, 162
- cursor 3
- data
 - kinds of 116
 - mixture of 119
 - pointer 118–19
 - storage of 116–17
- dimension 194
- disk
 - care 215
 - explanation 87
 - preparation 215–17
 - start-up program 217–18
 - use of 215–18
- dotted notes (music) 145–46
- drawing pictures 21, 104–5, 131, 133–36
 - line drawings 131, 133–36
- dumb loop 46
- duration 121
- entering a line 5
- envelope generator 121
- equal sign 58, 63–64
- erasing 26
 - files 90
 - letters in a program line 39
- error messages 32–33, 96–97
- error traps 206
- file commands 91
- filename 87, 89, 90–91
 - extension 89
- FOR-NEXT loops 107, 109
- functions 68, 78, 171
 - argument of 68
 - nesting 78
 - rules 172
- graphics
 - brush 137, 140
 - color 137, 139–42
 - movement of 101, 103–6
 - palette 137, 139
- hertz (Hz) 121, 122
- index number 193
- input cursor 37
- insert letters in a program line 39
- keyboard 24–28
 - auto-repeat feature 22
 - editing 24–28
- logical signs 180
- loops 48–50, 109–12
 - delay 65, 69–70, 107, 109, 206, 208, 213, 214
 - inside 110, 111
 - nested 107, 110–11
 - outside 110, 111
- loop variable 65, 107, 111. *See also* loops, delay
- medium-resolution graphics 131, 133
- memory
 - erasing a line 18
 - explanation of 14, 17
- music 143, 145–47
 - format of the PLAY string 145–47
 - scale of musical notes 123
 - rests 124
- not equal sign 56–57
- numbers 60, 61–62
- numeric constants 62
- numeric operations 58
- numeric variables 58
- output cursor 35
- phrase A 52, 54, 55, 72
- pitch 121, 122
- plus sign 41, 45, 162
- printing an empty line 11
- programs
 - adding a line 19
 - debugging 208, 213
 - definition of 5
 - fixing a line 19–20
 - listing 16
 - loading from disk 89–90
 - numbering the lines 7
 - outline of 203
 - parts of 203–4
 - printing to the screen 14
 - prompts 205
 - saving to disk 87–89
 - stopping 210–12
 - storing in memory 14
 - user-friendly 200, 202
 - writing 202–6
- programmer 33
- programming shortcuts 93, 95–99
- question mark 93, 95

random numbers 78, 80, 83–84
remark 20
repeating keys 24–25
rounding off 78, 82
screen capacity 103
scrolling 206
semicolon 95
 in PRINT statements 35, 38
shortcuts. *See* programming shortcuts
single quote 99
sound effects 124
spaghetti programming 46, 50–51
statement C 52, 54
string constants 9, 12, 13, 62
strings 13, 61–62, 162, 163–66
 cutting 163–65
 gluing 41, 45, 162
 length of 164–65
 switching numbers with 168, 170–71
string variables 29, 32
structured programming 188–91
subroutines 93, 149, 151, 152
subscript. *See* index number
syntax error 4
timing loops. *See* loops, delay
top-down programming. *See* structured programming
true and false 176–80
user 33
variable name 29, 41, 42–43
 length of 191
variables
 kinds of 170
 numeric 62, 193
 string 62, 193
 dollar sign with 61
 value of 41, 42–43, 61
white noise generator 121

Command and Function Index

AND operator 174, 176
ASC function 156, 157, 158, 171
BEEP command 9, 10, 121, 122
CHR\$ function 156, 158, 171
CIRCLE statement 131, 133, 135–36, 137, 140, 141
CLS statement 9, 11, 27, 28, 86
COLOR statement 9, 12, 125, 126, 127, 128, 139
CONT command 208, 210, 212, 213
DATA statement 114, 116, 118, 119, 121
DIM statement 192, 193, 194
DRAW statement 132
END statement 149, 153, 208, 210, 211
FILES command 86, 88, 91
FOR-NEXT statements 107, 109
GET statement 132
GOSUB statement 149, 151
GOTO statement 29, 46, 48, 52, 99, 212, 213
IF statement 29, 52, 54, 71, 72, 73, 98, 174, 178, 179
INKEY\$ variable 105, 182, 183, 184, 185, 186, 206
INPUT statement 29, 31, 34, 60, 95, 96, 182, 183, 184
INT function 78, 80, 81, 82, 171
KILL command 86, 90, 91
LEFT\$ function 162, 163, 171
LEN function 162, 163, 165, 171
LET statement 41, 42, 60, 64, 95, 212
LINE statement 131, 133, 135, 136, 137, 140, 141
LIST command 14, 16, 18, 86, 89, 97
LOAD command 86, 91
LOCATE statement 101, 103, 136, 206
MID\$ function 162, 163, 165, 171
NEW command 1, 3, 4, 5, 18, 86
NOT operator 174, 176, 179
ON-GOTO statement 156, 157, 160
OR operator 174, 176
PAINT statement 137, 141
PLAY statement 121, 122, 143, 145
PRINT statement 1, 3, 4, 7, 14, 21, 29, 34, 35, 37, 60, 63, 86, 93, 95, 136, 184, 208, 212, 213, 214
PSET statement 131, 133, 134, 136, 137, 140
PUT statement 132
RANDOMIZE statement 84–85
READ statement 114, 116, 118, 119
REM statement 1, 3, 4, 8, 14, 20, 86, 99
RESTORE statement 114, 116, 119
RETURN statement 149, 151
RIGHT\$ function 162, 163, 164, 171
RND function 29, 78, 80, 171
RUN command 1, 3, 4, 6, 212, 213
SAVE command 86, 91
SCREEN function 131, 133, 139
SOUND statement 121, 122, 124
STEP value 107, 110
STOP statement 190, 208, 210, 211, 212, 213, 214
STR\$ function 168, 170, 171
TAB function 65, 67, 68
THEN statement 97
VAL function 168, 170, 171, 182, 186

COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**

Call toll free (in US) **800-334-0868** (in NC 919-275-9809) or write COMPUTE! Books, P.O. Box 5058, Greensboro, NC 27403.

Quantity	Title	Price*	Total
_____	COMPUTE!'s PC & PCjr Games for Kids	\$14.95	_____
_____	Easy BASIC Programs for the IBM PC & PCjr	\$14.95	_____
_____	Home Applications in BASIC on the IBM PC & PCjr	\$12.95	_____
_____	COMPUTE!'s Guide to IBM PCjr Sound & Graphics	\$12.95	_____
_____	Beginner's Guide to BASIC on the IBM PCjr	\$14.95	_____
_____	Computing Together: A Parents & Teachers Guide to Computing with Young Children	\$12.95	_____
_____	Personal Telecomputing	\$12.95	_____
_____	COMPUTE!'s Guide to Adventure Games	\$12.95	_____

*Add \$2.00 per book for shipping and handling. Outside US add \$5.00 air mail or \$2.00 surface mail.

Shipping & handling: \$2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).

All payments must be in US funds.

NC residents add 4.5% sales tax.

Payment enclosed.

Charge Visa MasterCard American Express

Acct. No. _____ Exp. Date _____

Name _____

Address _____

City _____ State _____ Zip _____

*Allow 4-5 weeks for delivery.

Prices and availability subject to change.

Current catalog available upon request.

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COM-PUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call **919-275-9809**

COMPUTE!

P.O. Box 5058
Greensboro, NC 27403

My computer is:

- Commodore 64 TI-99/4A Timex/Sinclair VIC-20 PET
 Radio Shack Color Computer Apple Atari Other _____
 Don't yet have one...

- \$24 One Year US Subscription
 \$45 Two Year US Subscription
 \$65 Three Year US Subscription

Subscription rates outside the US:

- \$30 Canada
 \$42 Europe, Australia, New Zeland/Air Delivery
 \$52 Middle East, North Africa, Central America/Air Mail
 \$72 Elsewhere/Air Mail
 \$30 International Surface Mail (lengthy, unreliable delivery)

Name _____

Address _____

City _____

State _____

Zip _____

Country _____

Payment must be in US funds drawn on a US bank, international money order, or charge card.

- Payment Enclosed Visa
 MasterCard American Express

Acct. No. _____

Expires _____ / _____

Your subscription will begin with the next available issue. Please allow 4-6 weeks for delivery of first issue. Subscription prices subject to change at any time.

Grownups, Too!

Don't let the title fool you. *COMPUTE!'s Kids and the IBM PC & PCjr* was written for children from ages 10 to 14, but anyone interested in learning BASIC programming will find this series of lessons fun and easy to learn.

You'll discover exactly how to get the most out of your IBM computer. Everything is explained in nontechnical terms, and the many illustrations and program examples quickly show you the ins and outs of BASIC. You may be a beginner when you pick up this book, but before you know it, you'll be programming your own exciting games and applications. There are even notes before each lesson to help you understand the concepts discussed.

Whether you already know how to program, or have just unpacked your computer, you'll find lots of useful information in *COMPUTE!'s Kids and the IBM PC & PCjr*.

Some of the topics discussed are:

- What to do if you get an error message
- Assignments to help you practice what you've learned (with sample answers)
- Techniques to debug your programs
- Shortcuts to make programming faster
- How to save programs to disk
- And dozens of cartoons to let you laugh as you learn

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

ISBN 0-942386-93-0

