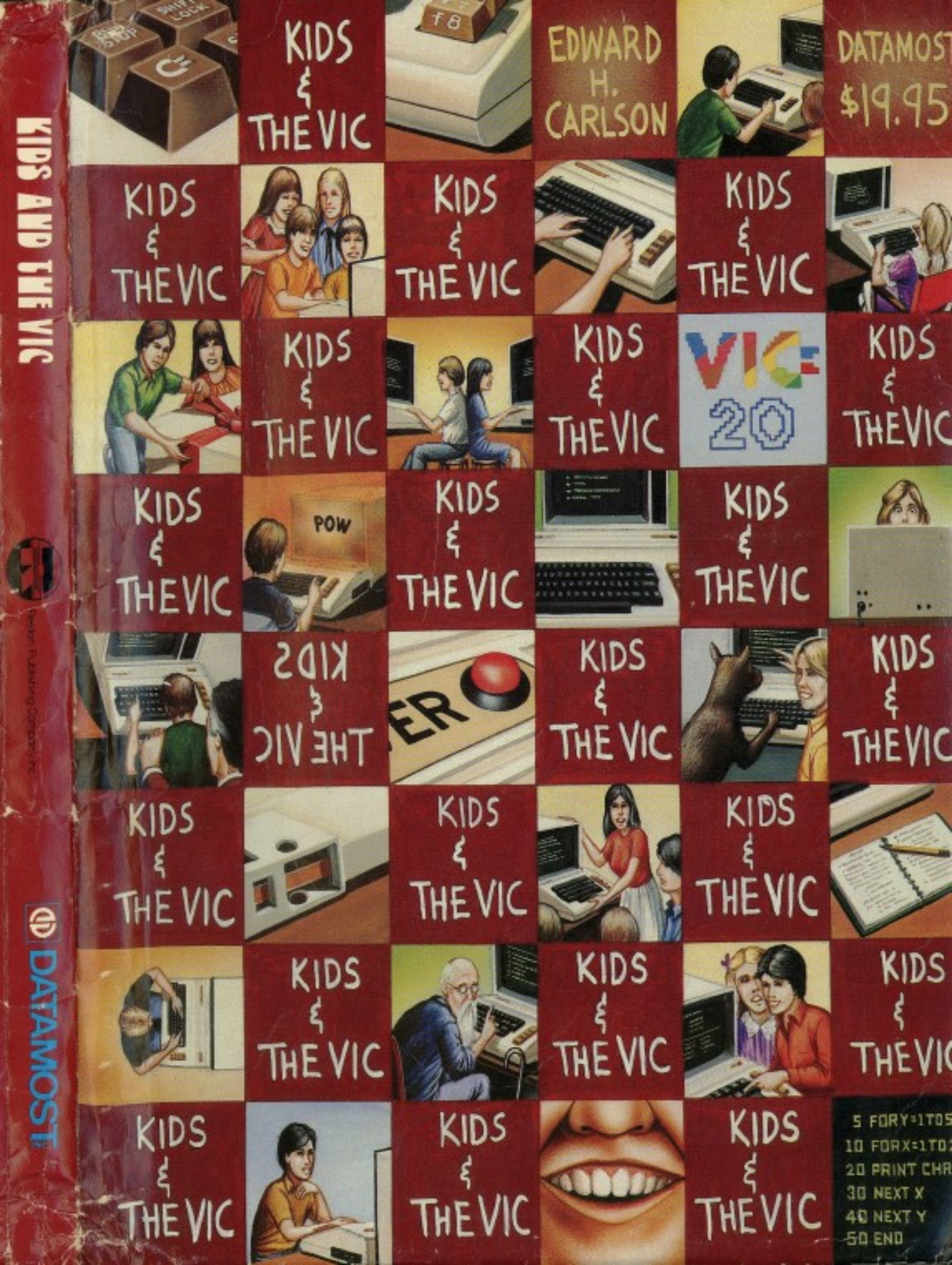
KIDS AND THE VIC



DATAMOST

KIDS AND THE VIC

by

Edward H. Carlson

Department of Physics and Astronomy
Michigan State University

Illustrated by

Paul D. Trap

DISCLAIMER

DATAMOST INC. shall have no liability or responsibility to the purchaser or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this manual or its use, including but not limited to any interruption in service, loss of business and anticipatory profits or consequential damages resulting from the use of this product.

© 1982 by
RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia
ISBN: 0-8359-3671-6

TABLE OF CONTENTS

INTRODUCTION

GRAPHICS, GAMES, AND ALL THAT

ADVANCED PROGRAMMING

ACKNOWLEDGEMENTS

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

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

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

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

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

John Decarli of the Computer Mart in East Lansing helped with the loan of a printer at a crucial point in the preparation of this manuscript.

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

TO THE KIDS

This book teaches you how to write programs for the VIC 20 computer.

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

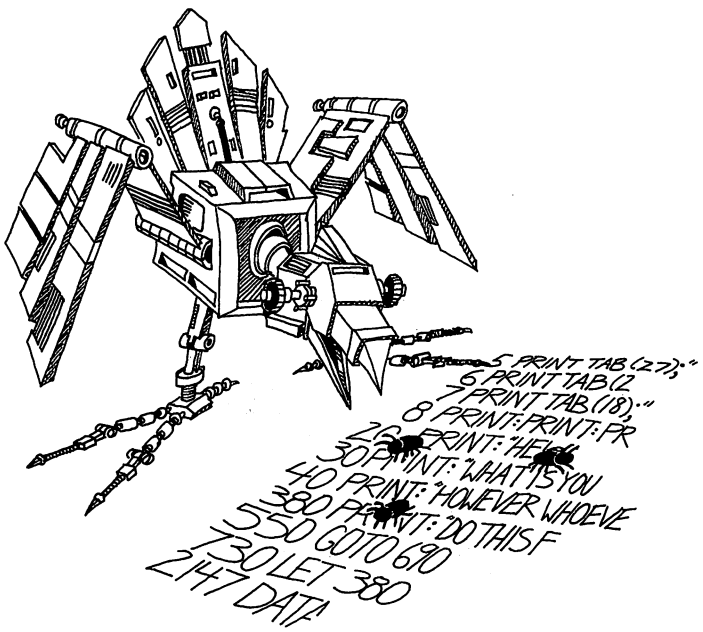
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

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

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

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

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



TO THE PARENTS

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

Your child will probably need some help in getting started and a great deal of encouragement at the sticky places.

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

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

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

TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches VIC BASIC on cassette systems.

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

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

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

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

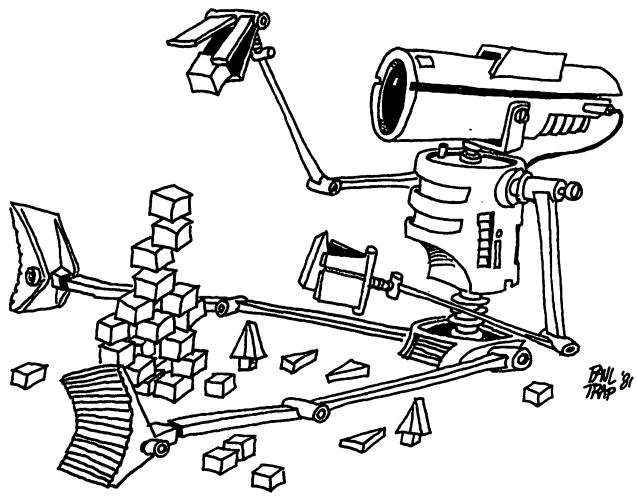
ABOUT PROGRAMMING

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

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

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

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



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

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

ABOUT THE BOOK

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

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

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

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

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

INTRODUCTION

INSTRUCTOR NOTES 1 NEW, PRINT, REM, AND RUN

This lesson is an introduction to the computer. Directions for turning on the machine are given on the next page.

The contents of the lesson:

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

QUESTIONS:

1. Write a program that will print your name.
2. Make the program disappear from the TV screen but stay in memory.
3. Run it.
4. Erase the program from memory.
5. Clear the screen and write a program that prints HELLO.
6. Make it run.
7. Erase it from memory but leave it on the screen.

LESSON 1 NEW, PRINT, REM, AND RUN

HOW TO GET STARTED

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

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

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

TYPING

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

SENDING THE CURSOR HOME

Press the CLR HOME key. The cursor jumps to the upper-left corner of the screen. This spot is the cursor's home.

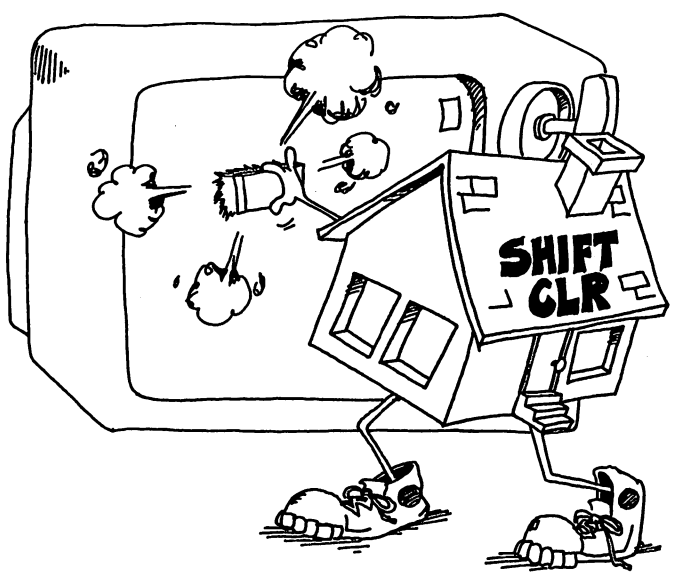
Now type some more. You are writing over what is already on the screen. This is a mess. Let's get a nice clean screen.

ERASING THE SCREEN

Two keys used together erase the screen.

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

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



COMMAND THE COMPUTER

Try this. Type: `GIVE ME CANDY`

and press the RETURN key.

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

The computer printed

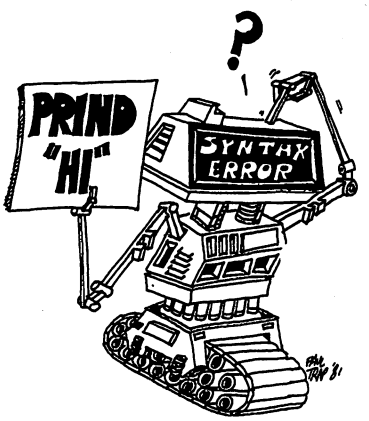
```
?SYNTAX
 ERROR
READY.
```

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

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

Here are the first 4 words to learn:

`NEW, PRINT, REM, and RUN.`

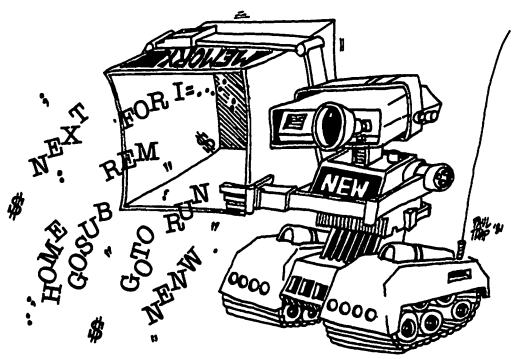


THE NEW COMMAND

Type: `NEW`

and press RETURN.

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



HOW TO ENTER A LINE

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

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

Press HOME and enter this line:

```
10 PRINT "HI"
```

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

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

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

THE NUMBER ZERO AND THE LETTER "O"

The computer always writes the zero like this:

zero Ø

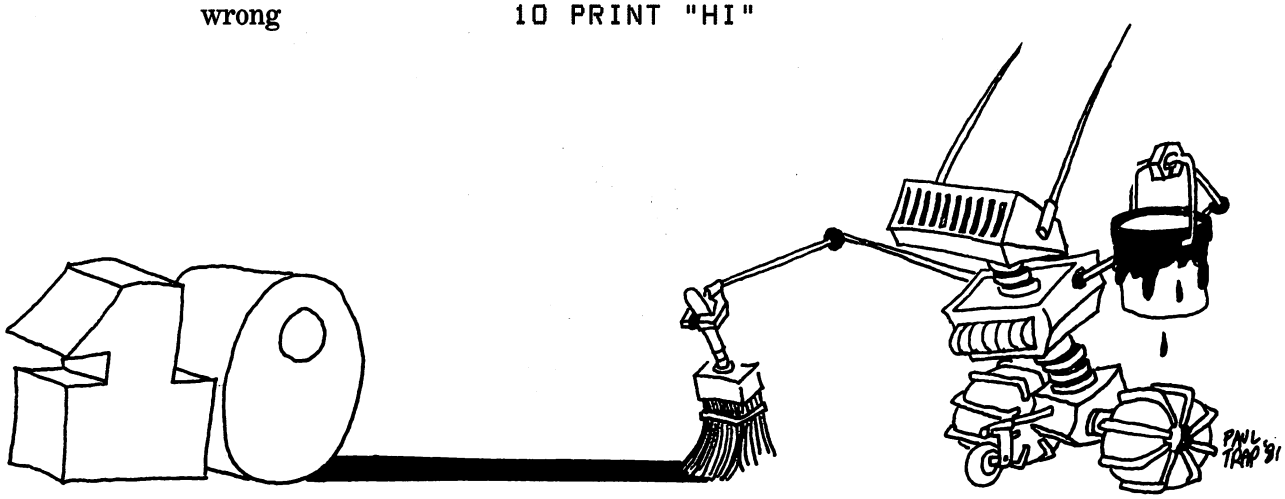
and the letter O like this:

letter O O

You have to be careful to do the same.

right `10 PRINT "HI"`

wrong `1O PRINT "HI"`



WHAT IS A PROGRAM?

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

HOW TO RUN A PROGRAM

A moment ago you put this program in memory:

```
10 PRINT "HI"
```

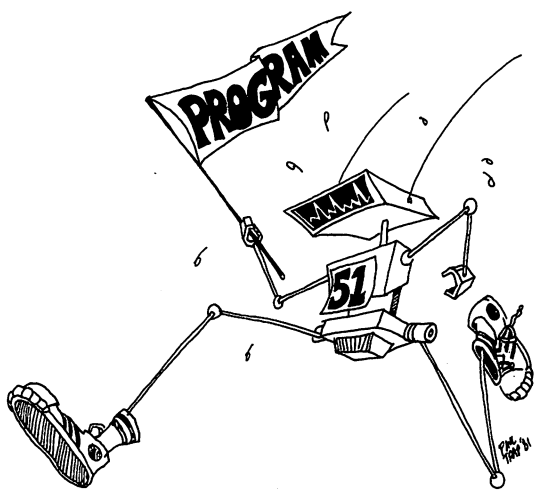
Now enter: `RUN`

(Did you remember to press the RETURN key?)

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

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

```
HI
```



HOW TO NUMBER THE LINES IN A PROGRAM

Clear the screen and enter this program:

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

This program has 3 lines. Each line starts with a command. You have already learned the PRINT command. You will learn about the REM command 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 commands in the lines. It starts with the lowest line number and goes down the list in order.

THE REM COMMAND

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

Assignment 1:

1. Use the CLR HOME key to move the cursor to the home position. Now use it to erase the screen.
2. Use the command NEW. Explain what it does.
3. Write a program that uses REM once and PRINT twice. Then use the RUN command to make the program obey the commands.

INSTRUCTOR NOTES 2 COLOR AND THE KEYBOARD

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

Each key has up to 3 functions, chosen by just pressing the key, or by pressing it while holding down the SHIFT key or the "Commodore Flag" key. For colors, the CTRL key is held down while a color key (one of the number keys) is pressed.

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

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

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

QUESTIONS:

1. How do you do each of these things:
   Make the computer type with red letters?
   Erase the screen?
   Empty the memory?
   Print your name?
2. What is a "string"?
3. What special key to you press to "enter" a line?
4. What is a command? Give some examples.
5. What does the computer mean when it prints "SYNTAX ERROR"?
6. How could you print "FIRE" with each letter in a different color?

LESSON 2 COLOR AND THE KEYBOARD

Empty the memory. (Remember? enter NEW)

Clear the screen. (Remember? SHIFT and CLR HOME keys)

You are ready to start this lesson.

COLOR ME RED

Let's change the color of the letters we type.

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

1. The character "3" in the middle.
2. The character "£" at the top.
3. The word "RED" on the front of the key.

Here are the three things this key can do:

Press the "3" key. It prints 3
Hold down the SHIFT key and press "3." It prints £.
Hold down the CTRL key and press "3" It changes the color!

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

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

There are 8 colors on the number keys. They are:

black	BLK
white	WHT
red	RED
cyan (blue-green)	CYN
purple	PUR
green	GRN
blue	BLU
yellow	YEL

How do you make the computer print with green letters? Do it.

Try the other colors.

THE RAINBOW

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

When this book says the word "red" in a PRINT command, it means you should:

Hold the CTRL key down and press the RED key

Clear the screen and enter this line:

```
20 PRINT"red"
```

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

Surprise! it will look like this on the screen:

```
20 PRINT"     "
```

(Remember: In this book, "red" in little letters is not the same as "RED" in big letters! "RED" is typed exactly as you see it but "red" means the CTRL and RED keys are pressed together.)

Now RUN the program. It will print "READY." in red letters. Add to the program to get

```
10 REM RAINBOW
20 PRINT"red"
30 PRINT"HI"
40 PRINT"grn"
50 PRINT"SALLY"        (Use your own name)
```

You can shorten the program by putting the "red" and "grn" in the same PRINT lines as the words. Do this:

```
10 REM RAINBOW
20 PRINT"red HI grn THERE"
30 PRINT"blk R red A cyn I pur N grn B blu O yel W"
```

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

```
30PRINT"blkRredAcynIpurNgrnBbluOyelW"
```

Aren't you glad I put in the spaces?

OTHER COMMANDS IN PRINT STATEMENTS

Just as "red" in a PRINT statement means CTRL RED, "clr" means SHIFT CLR, and "hm" means HOME.

In each case, you press one or two keys and you see something funny on the screen, not "red", "clr", or "hm."

Run this:

```
10 PRINT "clr"
20 PRINT "red HI"
30 PRINT "blu HI AGAIN"
```

Change line 30 to: 30 PRINT "hm blu HI AGAIN" and run it again.

PRINTING AN EMPTY LINE

Run this:

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

Line 30 just prints a blank line.

CHARACTERS

Look at these PRINT statements:

```
10 PRINT "JOE"
10 PRINT "*D47%%*%"
10 PRINT "19"
10 PRINT "3.14159265"
10 PRINT "I'M 14"
10 PRINT " "
```

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

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

```
10 PRINT " "
```

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

STRING CONSTANTS

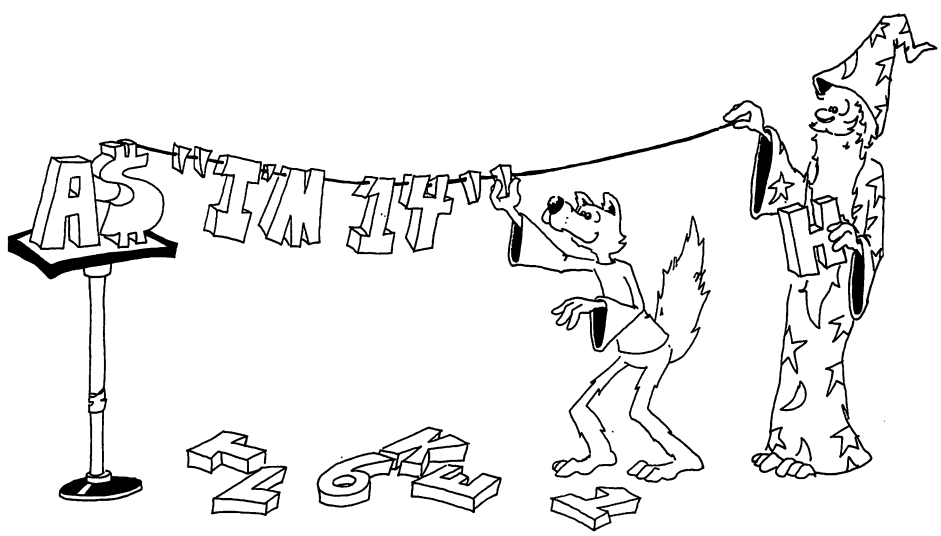
Characters in a row make a "string."

The letters are stretched out like beads on a string.

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

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

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



SPECIAL KEYS

The computer has the usual typewriter keys: letters, numbers and punctuation. It also has some special keys . You have used these:

SHIFT CLR HOME RETURN CTRL

Other special keys are:

INST DEL
RESTORE
CRSR two keys with arrows on them
RUN STOP
SHIFT LOCK
The COMMODORE "flag" key

You will learn how to use these keys later.

DOUBLE AND TRIPLE POWER FOR SOME KEYS

Most of the computer's keys have two or three meanings. They have two or three things written on them.

One thing happens if you just press the key.

Something else happens if you press the key while holding down another key.

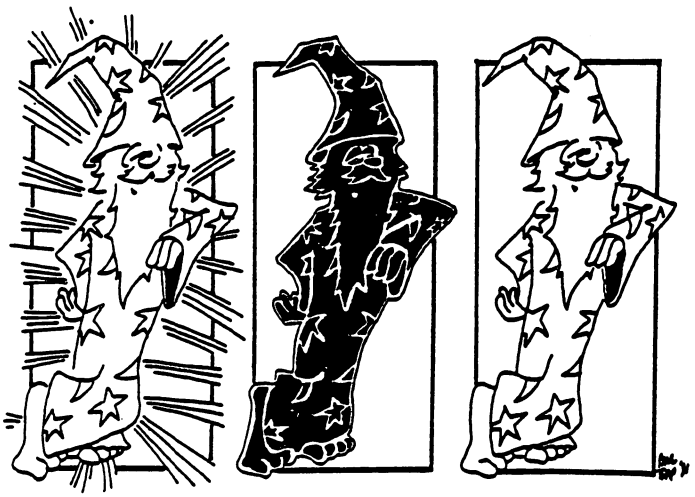
Sometimes you hold down a SHIFT key.

Sometimes you hold down the CTRL key.

Sometimes you hold down the COMMODORE key.

Assignment 2:

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



INSTRUCTOR NOTES 3 LIST, BOXES IN MEMORY

In this lesson:

LIST, LIST 30
Memory boxes holding lines
Erase one line from memory
Add a line between old lines
Replace a line
Drawings using PRINT commands

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

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

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

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

QUESTIONS:

1. How do you erase a line you no longer want?
2. Press CLR. 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, BOXES IN MEMORY

Clear the screen and erase the memory.

(Start each lesson by clearing screen and memory.)

Now enter:

```
10 REM TESTING
20 PRINT"HERE I AM"
```

Run this two line program. Then clear the screen.

The program is gone from the screen.

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

LISTING THE PROGRAM

To ask the computer to show you the whole program, enter:

```
LIST
```

If you want to see just one line of the program, then enter LIST followed by the number of the line, like this:

```
LIST 20
```

and the computer will list just line 20.

THE MEMORY

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

When you entered:

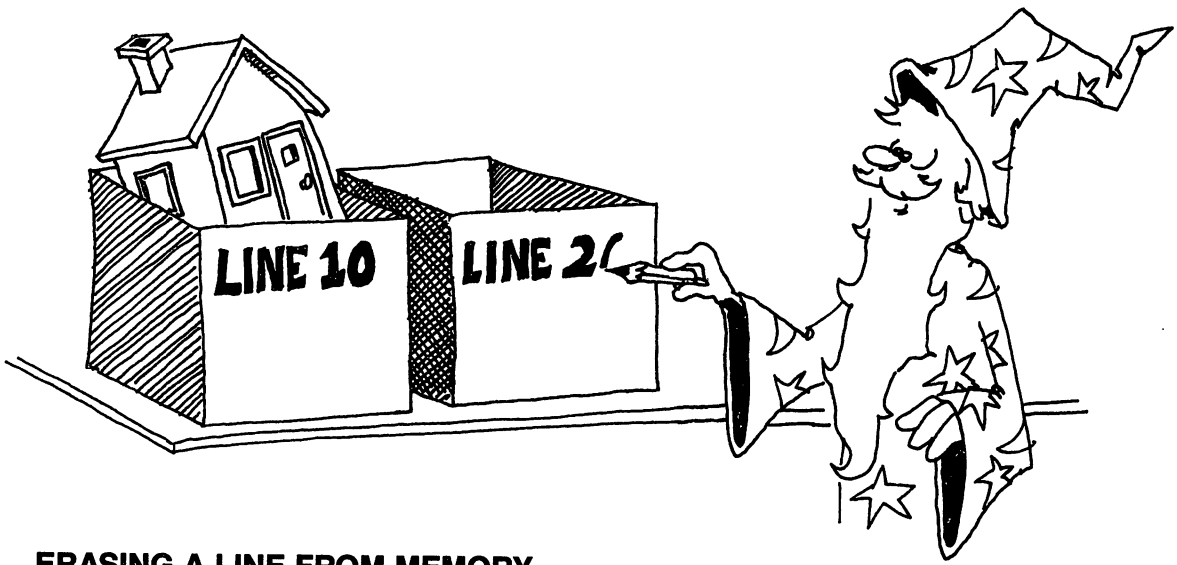
```
10 REM TESTING
```

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

When you entered:

```
20 PRINT"HERE I AM"
```

the computer took the second box and wrote "Line 20" on its label. Then it put the statement PRINT "HERE I AM" in the box and put that box in its place on the shelf.



ERASING A LINE FROM MEMORY

To erase one line of the program, enter the line number with nothing after it. For example, to erase line 20, enter:

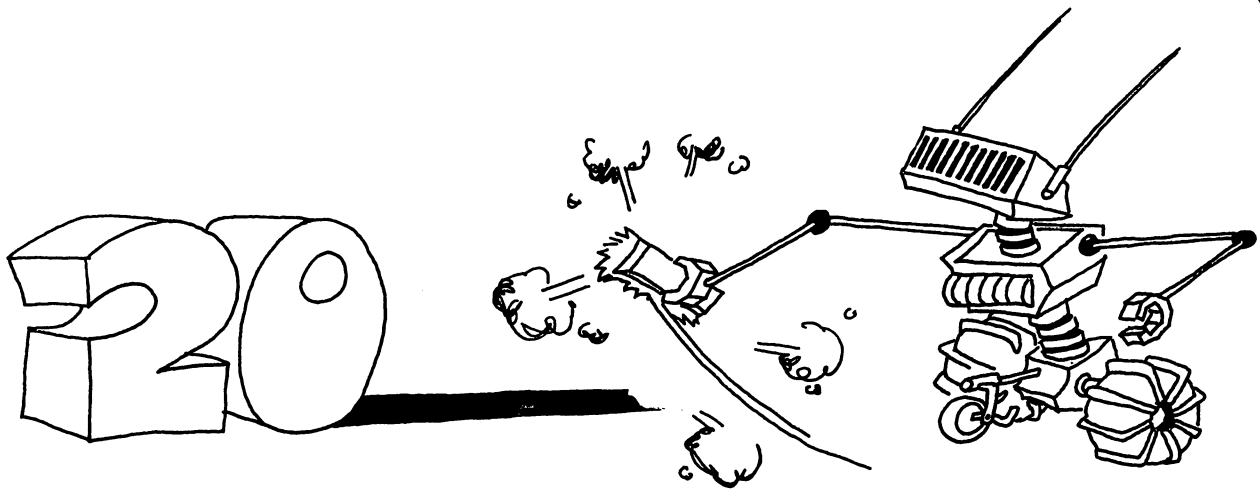
```
20
```

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

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

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

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



ADDING A LINE

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

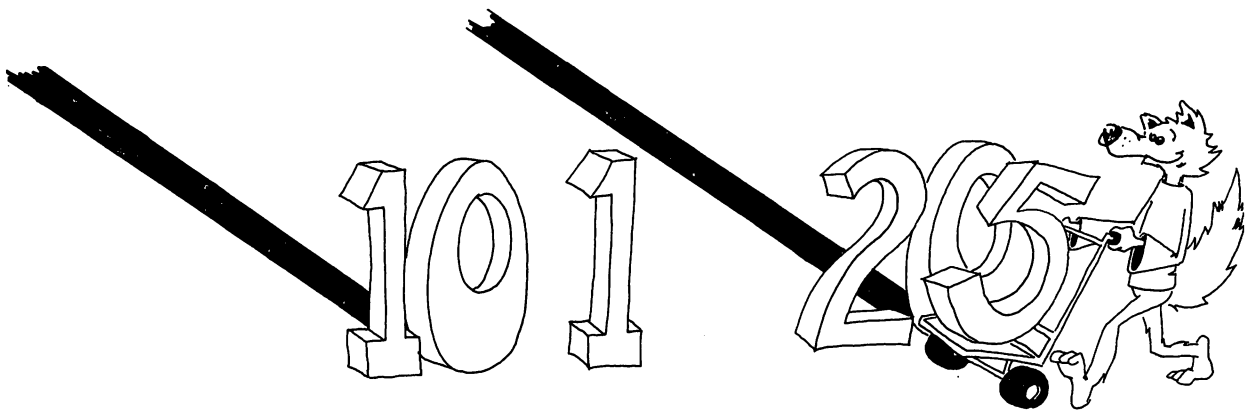
Enter NEW and this:

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

List it and run it. Now add this line:

```
15 PRINT"STILL"
```

List and run it again.

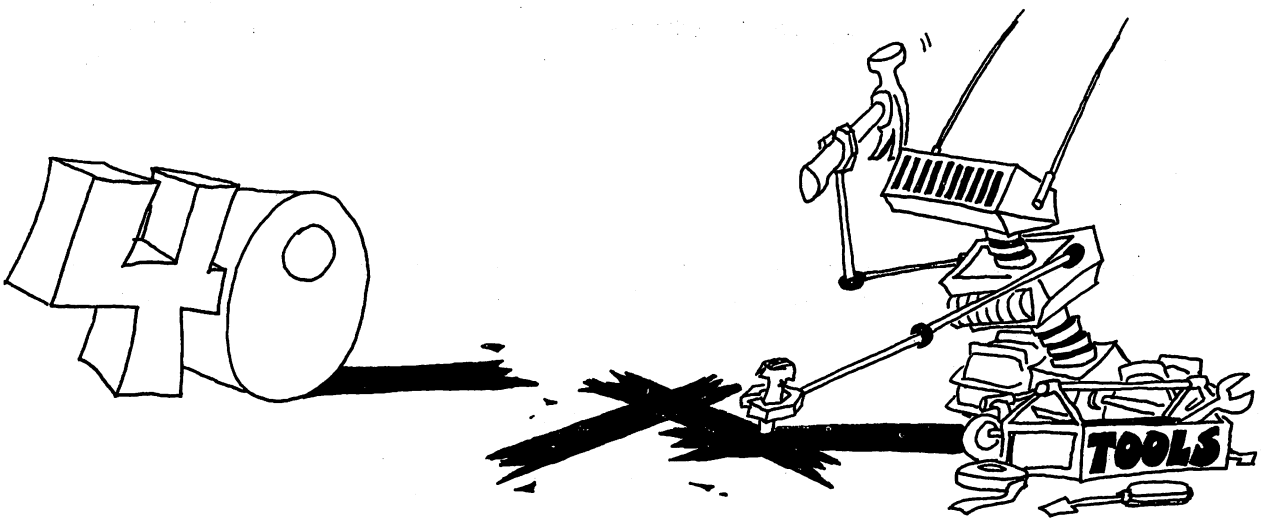


FIXING A LINE

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

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

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



THE REM COMMAND

Use a REM command to put a title on your program.

Enter NEW and this:

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

USING THE CLR KEY IN A PRINT COMMAND

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

Enter this program:

```
10 REM CLASSY CAR
20 PRINT"clr blu MERCEDES-BENZ"
```

Where it says "clr" in the PRINT statement, you should hold down the SHIFT key and then press the CLR key. Where it says "blu" you hold down the CTRL key and press the BLU key. The line 20 will look like this:

```
20 PRINT"♥ MERCEDES-BENZ"
```

Run the program.

PICTURE DRAWING

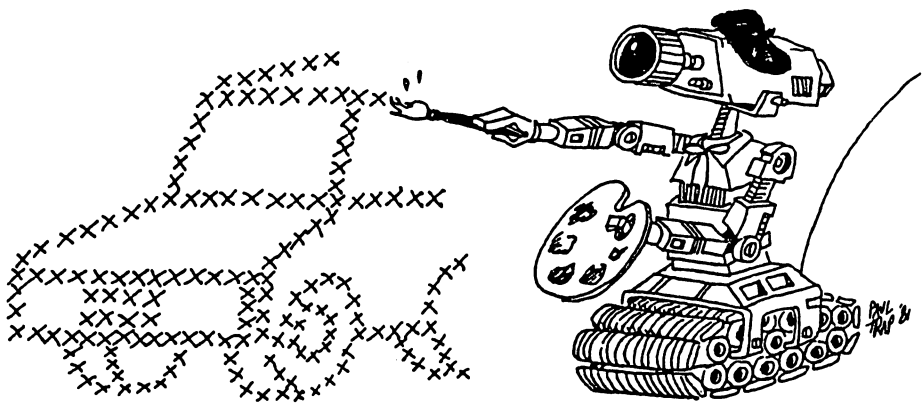
You can use the PRINT command to draw pictures. Here is a picture of a car. Add these lines to your program.

```
10 REM CLASSY CAR
20 PRINT"clr blu"
25 PRINT
30 PRINT" XXXXXX"
40 PRINT"XXXXXXXXXXXX"
50 PRINT" O          O"
```

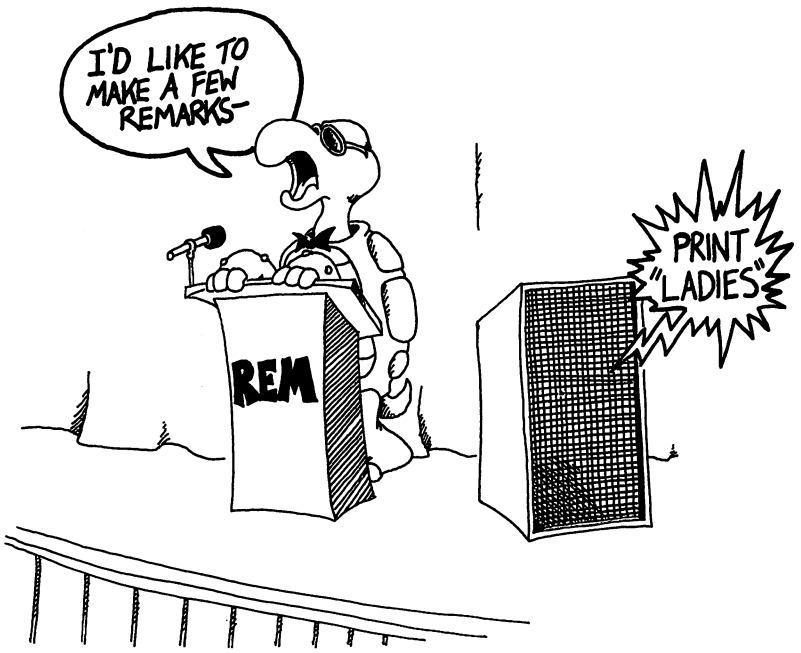
Don't forget to put the spaces in the PRINT lines! They are part of the drawing.

Assignment 3:

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



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



THE DELETE KEY

Erase typing errors with the INST DEL key. DEL stands for "delete." This key erases the character to the left of the cursor.

Try this:

```
10 PRINQ
```

(do not press RETURN)

You want a T instead of a Q.

The cursor is next to the "Q".

Press the DEL key. The "Q" is erased.

Press the T key. You get:

```
10 PRINT
```

If you hold down the DEL key, the cursor starts moving to the left, fast! It erases everything as it goes.

This is good for erasing a lot of stuff, but you have let up on the key quickly, or it will erase everything!

INSTRUCTOR NOTES 4 THE CURSOR KEYS AND DRAWING PICTURES

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

The arrow keys are used in moving the cursor to any point on the screen. In particular, moving the cursor onto any part of a line allows editing of the line. Characters in a line are not affected by the cursor moving over them. Wherever the cursor stops, you can type in new characters. You can even change the line number. For now we will not consider the insertion or deletion of characters, only their replacement by others or by spaces. When all is satisfactory, the line can be entered in the computer by pressing RETURN.

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

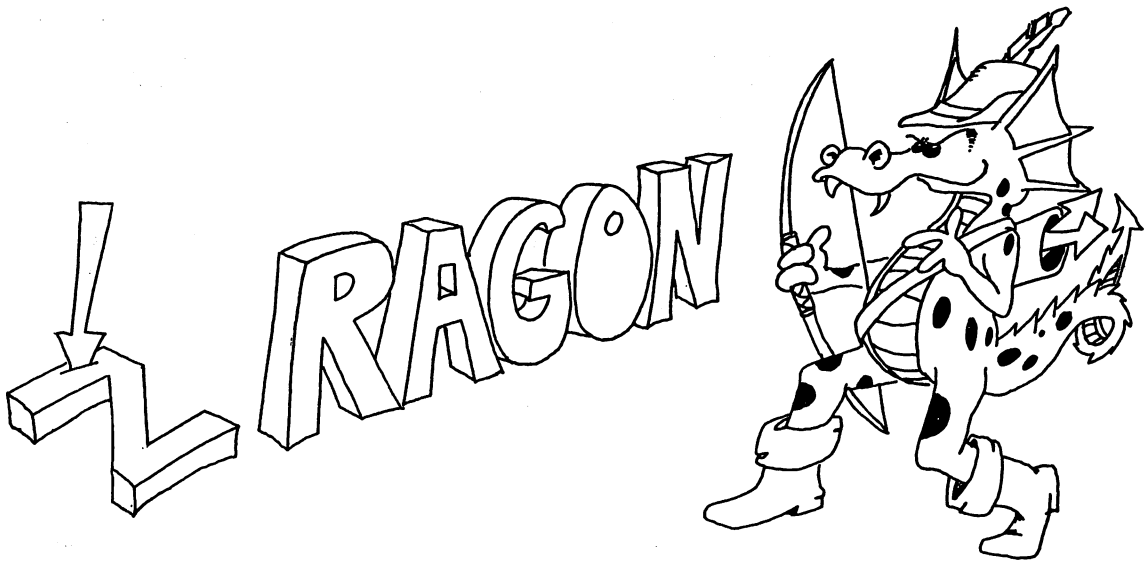
QUESTIONS:

1. What is a "cursor?" What is it good for?
2. Type the "solid ball" character on the screen. What keys do you need to use?
3. Type the triangle graphics character that appears on the asterisk key. What keys do you need to use?
4. Have your student demonstrate how to edit a line. This includes using the arrow keys to move the cursor to the interior of the line, modifying characters there, and pressing RETURN to enter the line.

LESSON 4 THE CURSOR KEYS AND DRAWING PICTURES

THE CURSOR IS A FLASHING SQUARE

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



THE ARROW KEYS

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

One CRSR key has left and right arrows.

The other CRSR key has up and down arrows.

These keys move the cursor.

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

↑ ← < >

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

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

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

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

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

THE GRAPHICS CHARACTERS

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

They are easy to use.

First find the SHIFT key and the "COMMODORE" key. The COMMODORE key looks like a large "C" with a flag flying from its side. It is under the RUN STOP key. It looks like this:

To print graphics characters:

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

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

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

Do this: Use the CRSR keys to move the cursor to the screen center and draw a small red square. (Hint: use the "corner" characters that are on the A, S, Z, and X keys.)

Now draw a large green square around the red square. (Hint: Use the corner characters and the horizontal and vertical lines on the star "*" and minus "-" keys.)

FIXING MESSED UP LINES

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

Enter:

```
10 REM ZRAGON
```

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

```
Type a "D" instead.
```

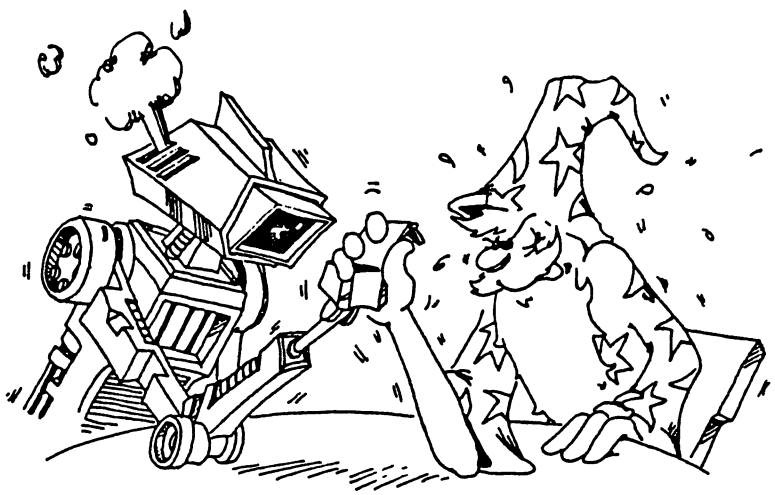
Now the line is correct, reading:

```
10 REM DRAGON
```

Press RETURN to store the correct line in the memory.

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 RETURN key to enter the line into memory.
2. Draw a "smiley face."
3. Draw a valentine.



INSTRUCTOR NOTES 5 THE INPUT COMMAND

This lesson concerns the INPUT command and the idea of a string variable.

We introduce the input command in its simplest form:

```
INPUT A$
```

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

Similarily, we will give only the essential feature of each command for the whole of the introduction of the book (through lesson 14). This allows us to quickly outline the essentials of programming so that the student "sees the forest" and is able to write meaningful programs. The commands required for interesting programs are:

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

Back to this lesson. String variables are introduced using the "box" concept again. Variable names are restricted to one letter for the time being. This does not lead to confusion in short programs and allows faster typing. It also puts off the discussion of the complicated naming rules until after our sprint to the RND command.

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

QUESTIONS:

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

LESSON 5 THE INPUT COMMAND

Use INPUT to make the computer ask for something.

Enter:

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

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

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

When you 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$.

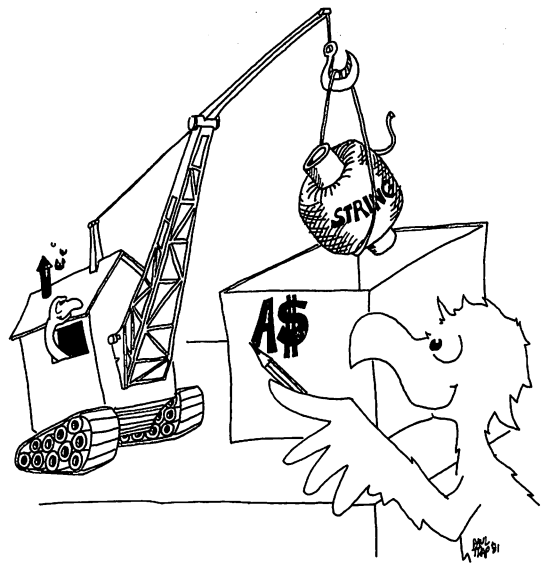
Run the program again and this time say something funny.

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 three times:

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

Try these answers:

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

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

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

```
?EXTRA IGNORED
```

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

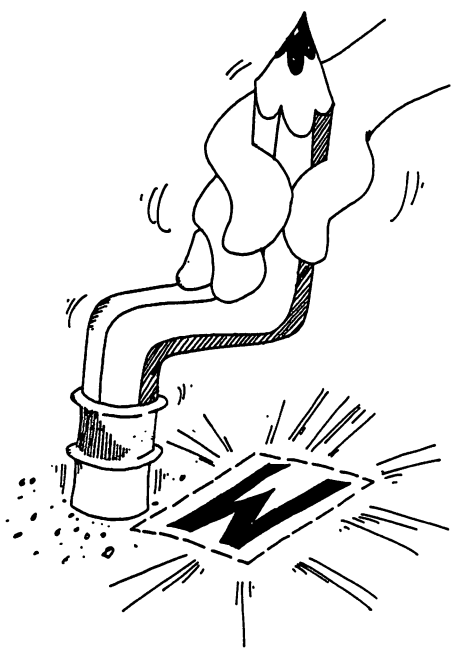
THE DELETE KEY

The DEL key is your "eraser." Try this:

```
20 PRINT  HI THERZ
```
(leave the cursor after the Z)

oops! The Z should be an E. You can erase the Z by pushing theDEL key.

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



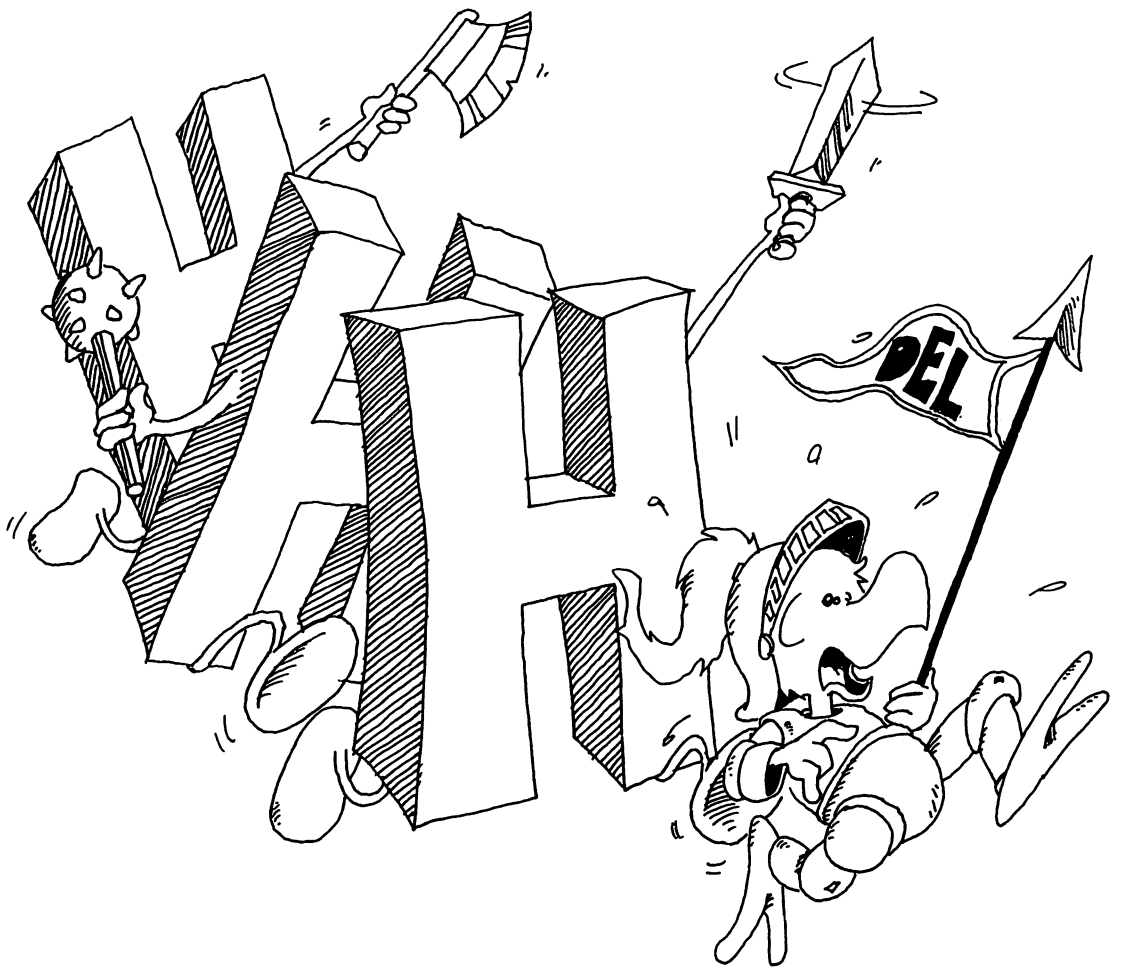
Rule: The DEL key always erases the character next to the cursor on the left.

SPEEDY ERASING

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

Assignment 5:

1. Write a program that asks for a person's name and then says something silly to the person, by name.
2. Write a program that asks you to INPUT your favorite color and put it in a box called C$. Now the program asks you your favorite animal and puts this in box C$ 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, SOUND

In this lesson:

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

The use of commas in PRINT is ignored as it is of little use on a 22 column screen.

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

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

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

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

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

QUESTIONS:

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

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

   If not, how do you put a space between them?

LESSON 6 TRICKS WITH PRINT, SOUND

ONE LINE OR MANY?

Enter this program:

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

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

Now enter:

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

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

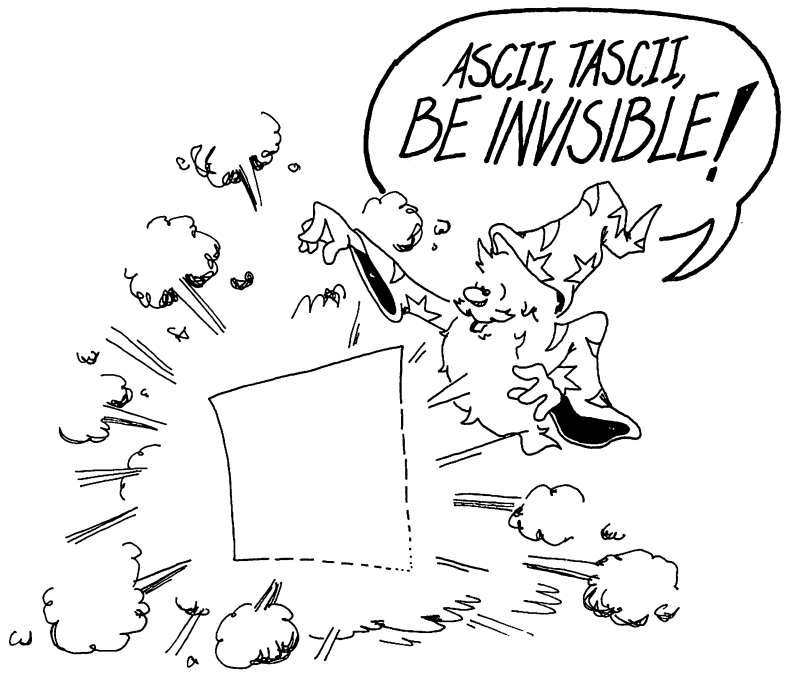
Run it.

What was different from the first time?

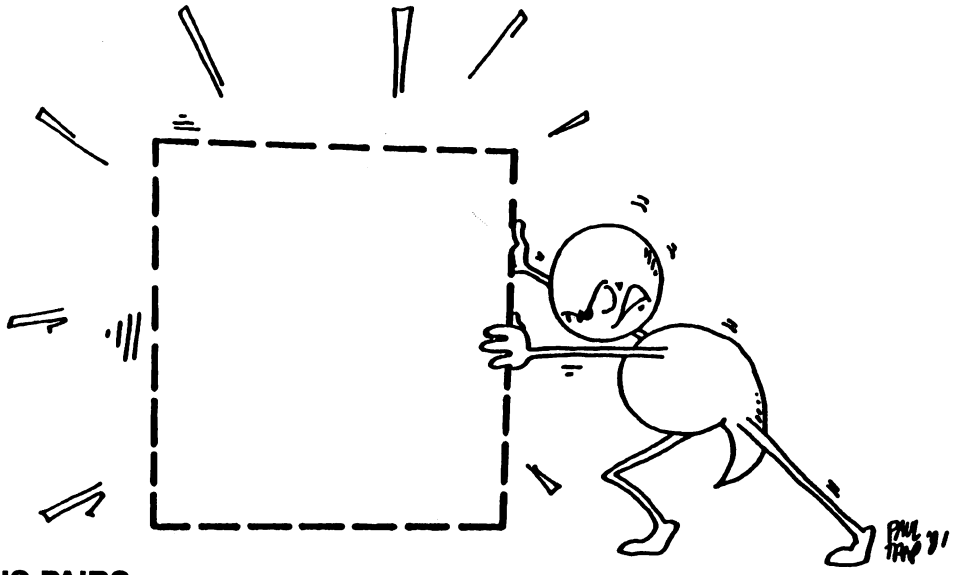
THE HIDDEN CURSOR

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

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



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



FAMOUS PAIRS

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

Enter:

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

(Remember "clr" means hold down the SHIFT key and press the CLR key.)

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

SQUASHED TOGETHER OR SPREAD OUT?

Enter NEW then try this:

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

after you have run it, try also:

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

don't forget the spaces after ROCK and AND.

THE COMPUTER PEEPS

Enter this:

```
10 REM MUSIC
20 POKE 36878,3
30 POKE 36875,250
80 FOR T=1 TO 250:NEXT T
90 POKE 36875,0
```

Be very careful that every number is correct. Then run it.

(If you do not hear a "peep" then turn up the sound on your television and RUN again. The sound will not work if you are using a "monitor" instead of a TV.)

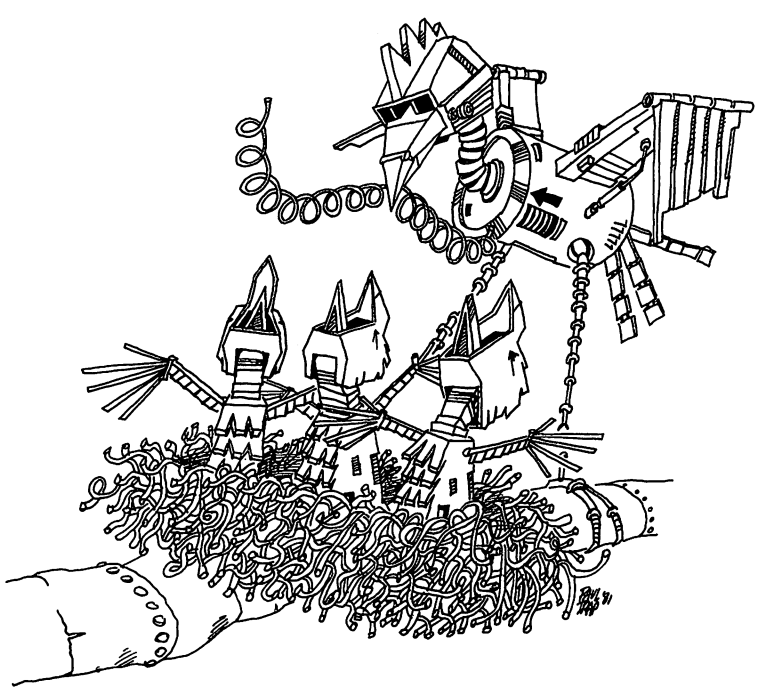
Now change the number 250 in line 30 to some other numbers between 135 and 241. What happens to the sound you hear?

Assignment 6

1. Write a program that asks for the name of a musical group and one of their tunes. Then using just one PRINT command, print the group name and the tune name, with the word "plays" in between.
2. Do the same, but use 3 PRINT commands to print on one line.
3. Make a program play a little tune, using these notes:

   C 135 D 147 E 159 F 163 G 175 A 183 B 191 C 195

   Do this by adding lines between 30 and 80 in the MUSIC program above. After each line like 30 that makes a note, put a line like 80 that makes it play for a while.



INSTRUCTOR NOTES 7 THE LET COMMAND, GLUING STRINGS

The LET command is introduced using the concept of memory boxes. Concatenation of strings glues short strings together to make long ones.

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

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

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

Covered so far:

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

Special keys discussed so far:

RETURN, CRSR arrows, SHIFT, CLR HOME, CTRL, and the "Commodore flag."

QUESTIONS:

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

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

   what will be in box A$ at the end? What will be in box B$?
3. In this program:

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

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

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

LESSON 7 THE LET COMMAND, GLUING STRINGS

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

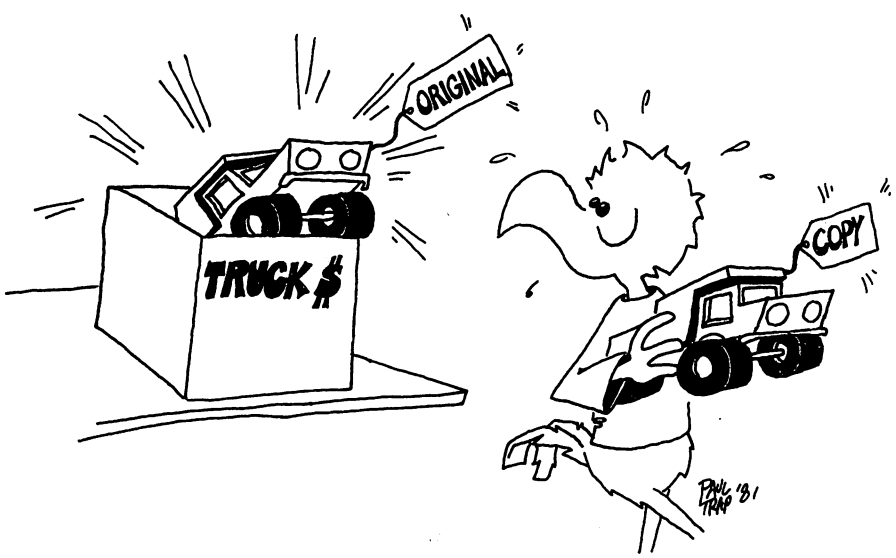
```
10 PRINT"clr"
20 LET W$="MOPSEY"
40 PRINT W$
```

Here is what the computer does:

Line 10 The computer clears the screen.

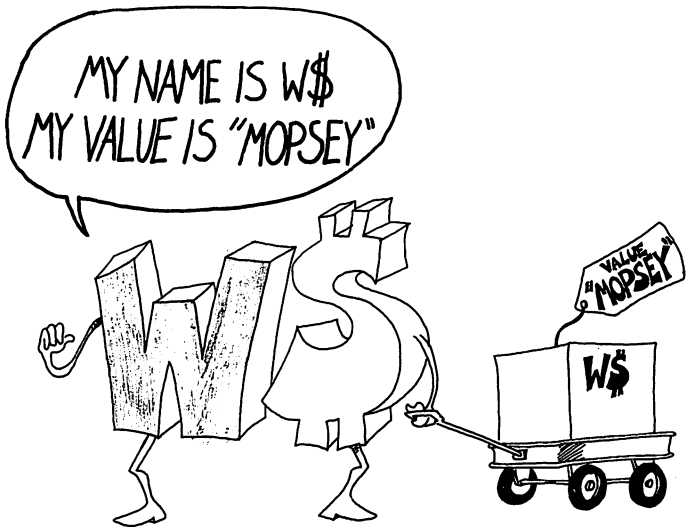
Line 20 It sees that a box named "W$" is needed. It looks in its memory for it. It doesn't find one because "W$" has not been used in this program before. So it takes an empty box and writes "W$" on the front, and then puts the string "MOPSEY" in it.

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



NAMES AND VALUES

The name of the variable is W$. The value of the variable is put in the box. In the program above, the value of W$ is "MOPSEY."



ANOTHER EXAMPLE:

Enter and run:

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

Explain what the computer does in each line.

10 D$ shows that pickles is a string

20 A$ shows that and is a string

30 Also What goes with pickles is a string

35 Z$ is also a string

40 _____

50 _____

GLUING THE STRINGS

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

```
10 PRINT"clr"
20 LET W$="HAR DE "
25 LET X$="HAR "
30 L$=W$+X$
40 PRINT L$
50 PRINT
60 LET L$=L$+X$
70 PRINT L$
```

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

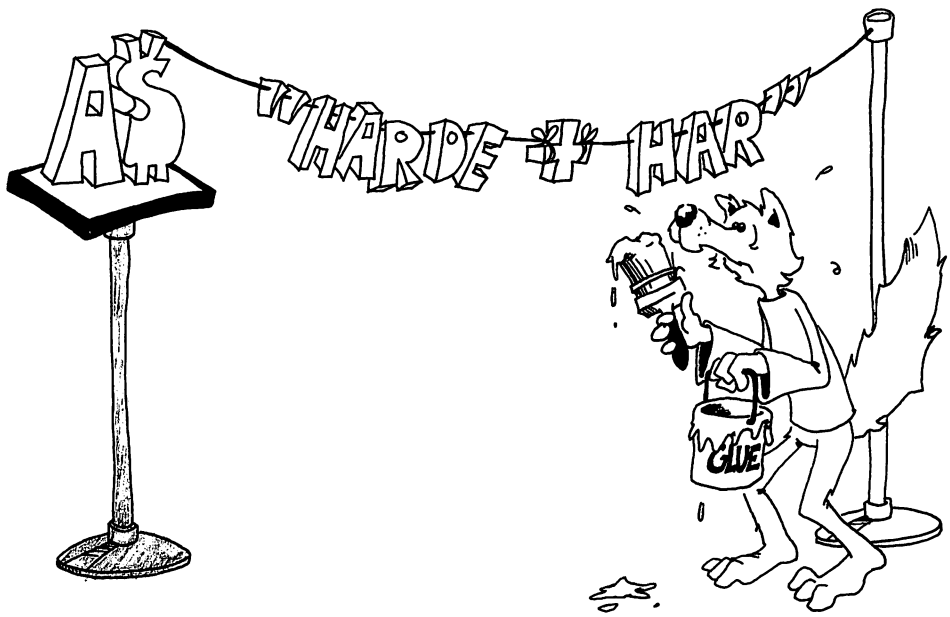
40 _____

70 _____

Now run the program to see if you were right.

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

Line 50 in the program above just prints an empty line. This is good for spacing apart the things your program must print.



Assignment 7:

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

INSTRUCTOR NOTES 8 THE GOTO COMMAND AND THE STOP KEY

The GOTO command allows a "dumb" loop that goes on forever. It also helps in flow of command in later programs, after the IF is introduced. It provides a slow and easy entrance for the student into the idea that the flow of command need not just go down the list of numbered lines.

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

The problem is how to stop it. The STOP key does this nicely.

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

QUESTIONS:

1. In this little program:

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

   what will appear in the screen when it is run?
2. And this one:

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

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

LESSON 8 THE GOTO COMMAND AND THE STOP KEY

JUMPING AROUND IN YOUR PROGRAM

Try this program:

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

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

STOP key.

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

We will use GOTO in a lot of programs.

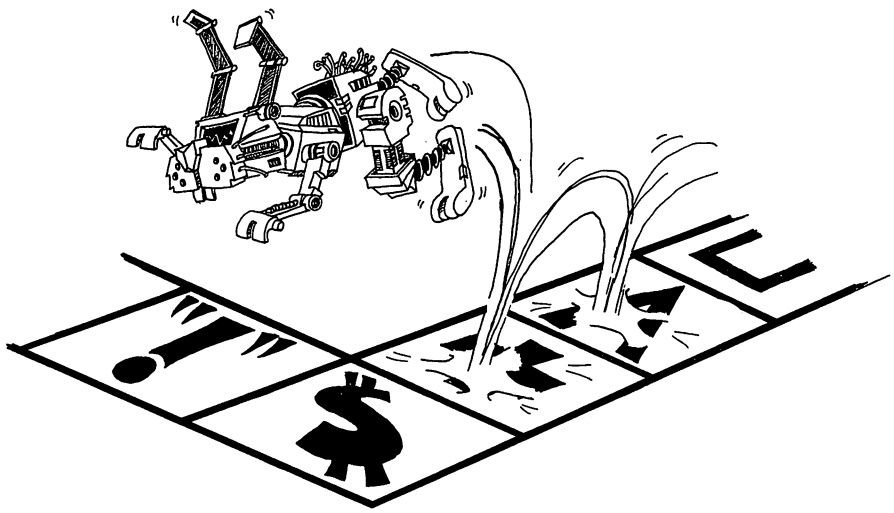
MORE JUMPING

Enter:

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

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

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



KINDS OF JUMPS

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

Jumping back gives a LOOP.

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

The path through the program is like this:

```
10 PRINT "HI"

20 GOTO 10
```

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

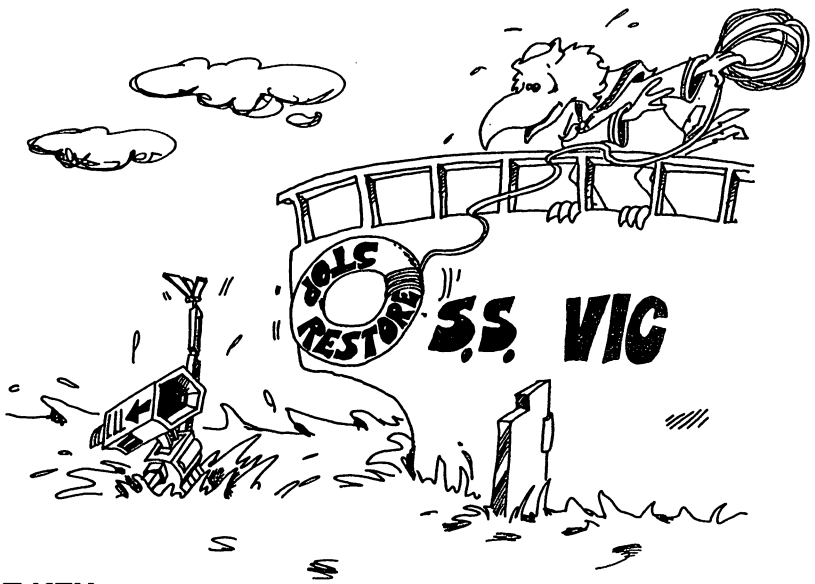
Jumping ahead lets you skip part of the program. It is not useful yet, but we will use it later in the IF command.

THE STOP KEY

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

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

(The RUN part of the key can be used to load programs from tape. Because a file name cannot be used with the RUN key, it is not often used.)



THE INSERT KEY

INST stands for "insert" which means "put in."

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

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

Now use the CRSR arrows to move the cursor onto the U. Then press the SHIFT INST keys four times. Then type WAY.

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

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

THE INSERT KEY GOES CRAZY

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

Assignment 8:

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

   PRINT, INPUT, LET, GOTO. It also should glue two strings together.

INSTRUCTOR NOTES 9 THE IF COMMAND

The IF command is introduced in this lesson. The case where two strings are the same or not the same is treated.

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

The IF command appeals both to our verbal and our visual imagination. The "cake" cartoon and the "fork in the road" cartoon illustrate these ideas. That the flow of commands may be altered has already been introduced with the GOTO command. To that idea is now added the conditional test: if an expression is true, one thing happens, if it is false, another.

The phrase "something A" is used for the expression being tested for truth. Some manuals call "something A" an "assertion." The phrase "command C" is used for the command to be done if the assertion is true.

Two ideas occuring in the "something A" may be confused with each other. They are the "=" relation and the truth of the overall assertion.

The case where the "⟨⟩" sign is used may help distinguish the idea of the "truth" of the assertion from the logic within the assertion.

On the other hand, the "⟨⟩" sign is probably unfamiliar to the student and it is possible that confusion is compounded at this point. In that case, just work with the overall IF idea and use the "" case for examples. We will return to the IF at later points in the book and at that time familiarity with IF will work to the student's advantage. The larger set of relations:

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

will be treated then.

QUESTIONS:

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

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

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

LESSON 9 THE IF COMMAND

Clear the memory and enter:

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

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

YES _____

NO _____

MAYBE _____

THE IF STATEMENT

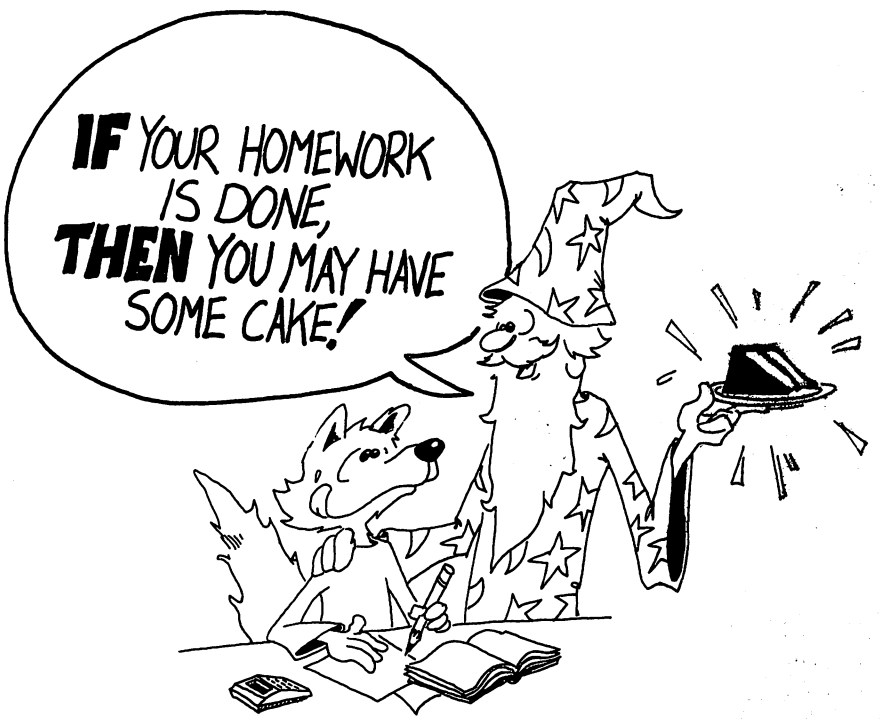
The IF statement has two parts:

```
10   IF   something   A   THEN   command C
```

First the computer looks at "something A."

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

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



It looks like this:

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

or

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

SOME EXAMPLES OF IF

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

Line 50 and line 55 do exactly the same thing.

Assignment 9A:

1. Add these lines to the program:

```
15 INPUT A$
17 LET C$="WHAT?"
20 LET B$="PAY UP"
25 LET N$=A$
85 PRINT C$
87 PRINT "red"
99 GOTO 15
```

   Run the program and enter these words:

   YES, BIRD, READY, NO WAY

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

   Get out of the program with the STOP RESTORE keys.

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

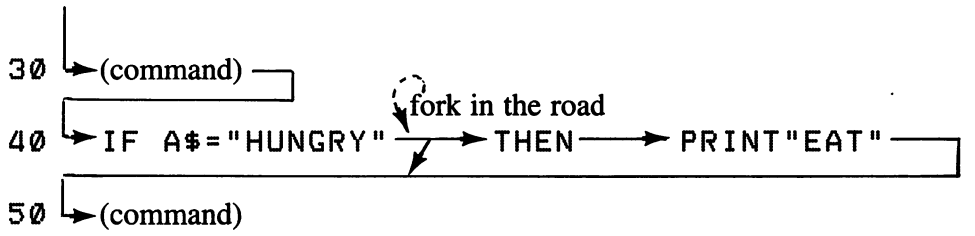
A FORK IN THE ROAD

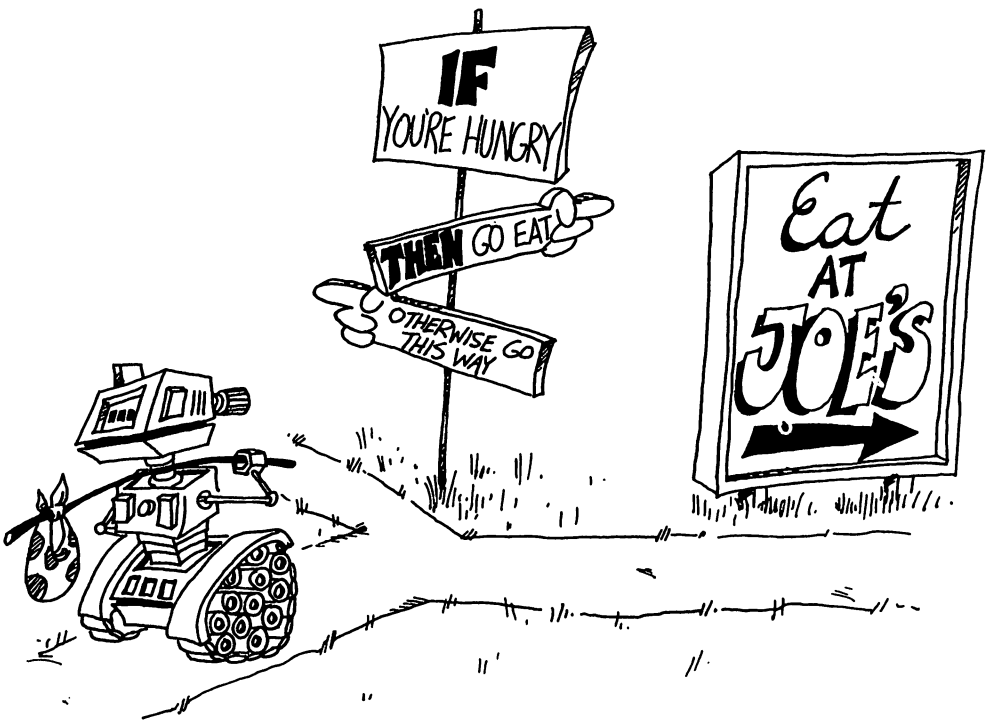
When it sees "IF", the computer must choose which road to take.

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

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

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





THE "NOT EQUAL" SIGN

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

To make the "⟨⟩" sign, hold down the shift key and press the "⟨" key, then the "⟩" key.

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

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

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

and put it in an IF command:

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

If the Q$ box contains "COLD" then Q$ is not equal to "FIRE" and the expression Q$⟨⟩"FIRE" is TRUE. The computer will print "NO SMOKE."
If the Q$ box contains "FIRE" then the phrase:

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

is FALSE and computer will not print anything.

Here is how it looks in a program:

```
10 PRINT"IS YOUR HOUSE ON FIRE?"
20 PRINT"(ENTER 'FIRE' OR 'COLD')"
30 INPUT Q$
40 IF Q$<>"FIRE" THEN PRINT "NO SMOKE"
50 IF Q$= "FIRE" THEN PRINT "HELP"
```

Assignment 9B:

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

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

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

INSTRUCTOR NOTES 10 INTRODUCING NUMBERS

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

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

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

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

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

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

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

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

QUESTIONS:

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

LESSON 10 INTRODUCING NUMBERS

INPUT, LET, AND PRINT

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

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

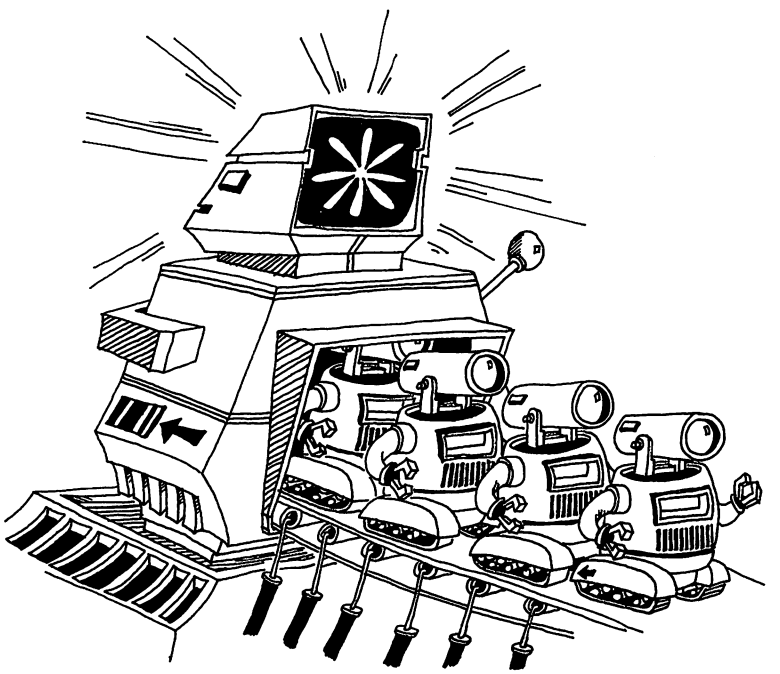
ARITHMETIC

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

Computers use "*" instead of "x" for a multiplication sign.

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

Computers use "/" for a division sign. It is on the "?" key. Answers are given as decimals.

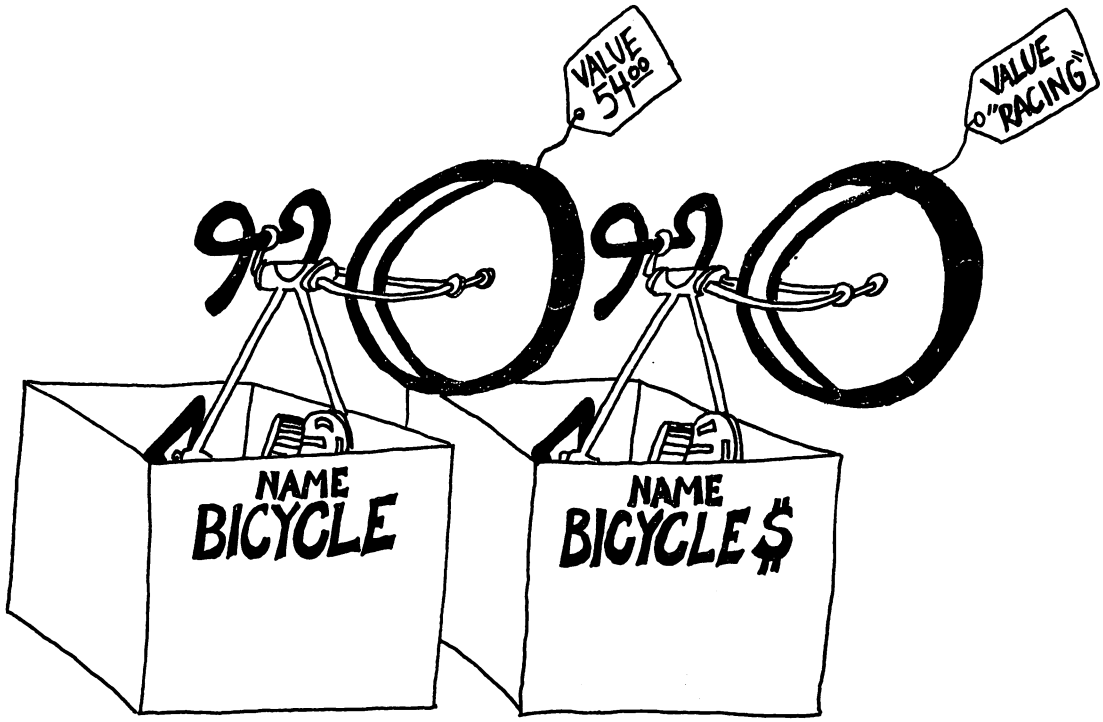


VARIABLES

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

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

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



ARITHMETIC IN THE LET COMMAND

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

Some more examples:

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

CAREFUL!

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

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

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

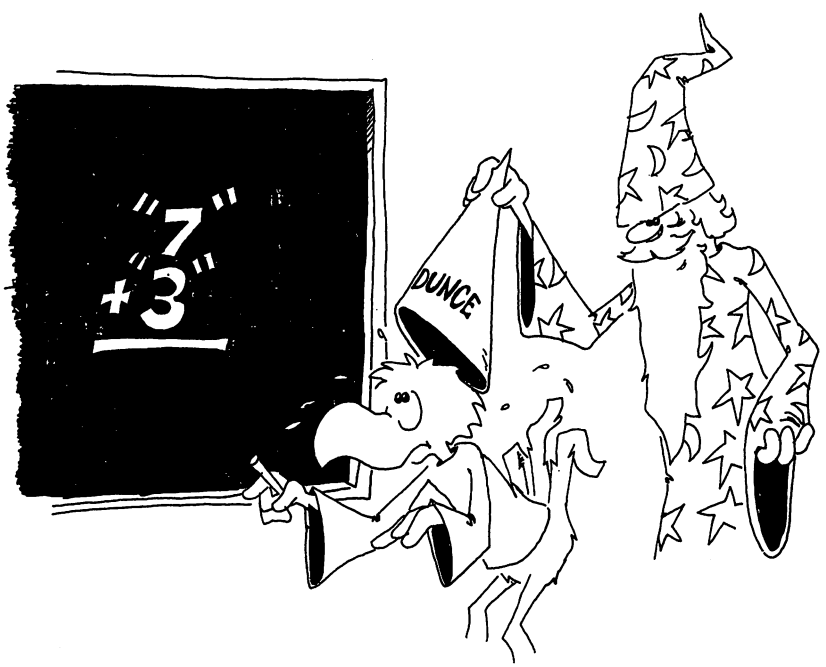
Some string constants: "HI", "7", "TWO", "3.14"

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

Correct:
```
10 LET A = 3 + 7
```

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

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



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

```
TYPE MISMATCH ERROR IN LINE 10
```

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

Enter:

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

Lines 10 and 20 are ok, line 30 is wrong. What will the computer do when you run this little program? Try it. Try to guess what each of these statements will print, then enter the line to see what happens:

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

MIXTURES IN PRINT

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

Correct:

```
PRINT A;"SEVEN ";"7"
PRINT A;B\
```

Run this line.

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

A FUNNY THING ABOUT THE EQUAL SIGN

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

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

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

```
7=7+1
```

which is not correct.

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

Look at this:

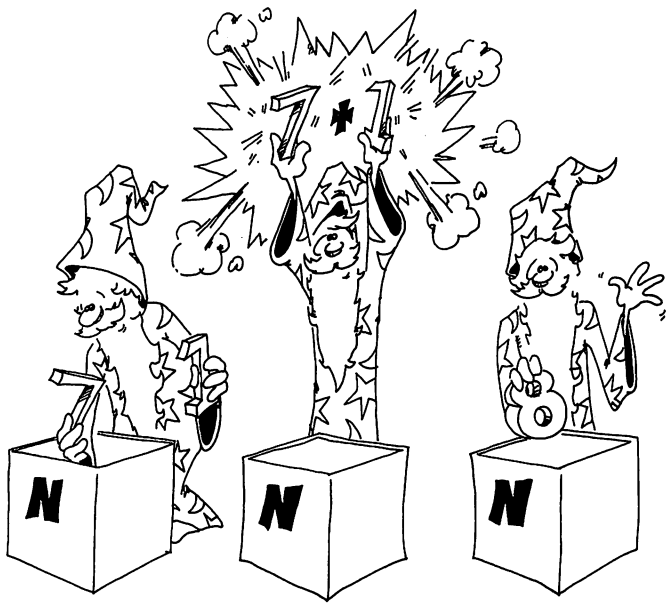
```
10 LET N=N+1
```

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

It takes the number 7 from the box.

It adds 1 to the 7 to get 8.

Then it puts the 8 in the box.



Another way to say the same thing is:

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

Assignment 10:

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

INSTRUCTOR NOTES 11 TAB AND DELAY LOOPS

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

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

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

The VIC has a more general and powerful way of moving the cursor around. One simply puts CRSR arrows in quotes in a PRINT command.

This lesson introduces loops in a painless way.

The delay loop is all on one line, with a colon to separate off the NEXT command. The amount of delay is determined by the size of the loop variable. A value of 1000 gives about a 1- second delay.

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

QUESTIONS:

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

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

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

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

LESSON 11 TAB AND DELAY LOOPS

THE TAB COMMAND

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

(The printing cursor is invisible.)

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

Try this:

```
10 PRINT "123456789ABCDEF"30
30 PRINT TAB(3);"Y";TAB(9);"Z"
```

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

CAREFUL!

Run this:

```
10 TAB(5)
```

You see SYNTAX ERROR IN 10. TAB() has to be in a PRINT command. You cannot use TAB() by itself.

YOU CANNOT TAB BACKWARDS

Try this:

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

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

YOUR NAME IS FALLING!

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

Press STOP to stop the run.

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

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

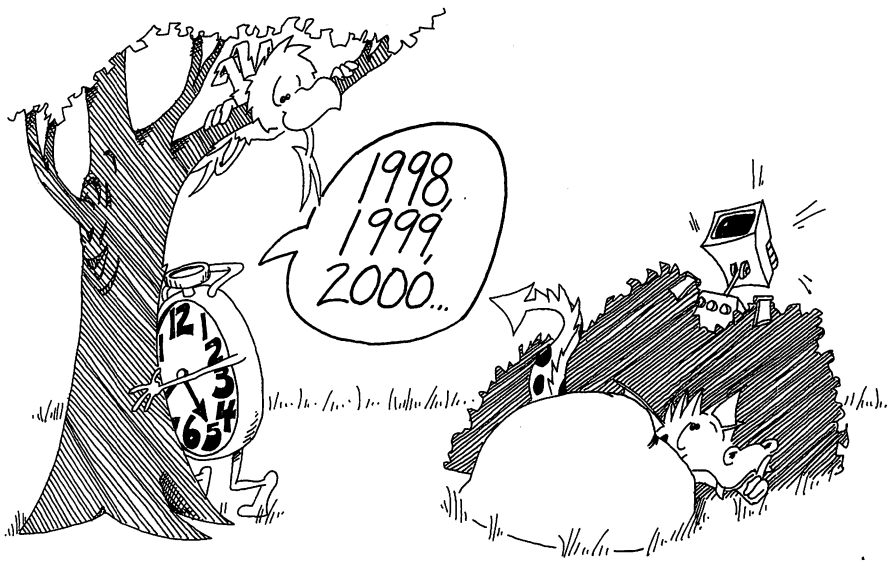
DELAY LOOPS

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

Run this program:

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

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



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

Each 1000 in the delay loop is worth about 1 second of time. Try this:

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

Assignment 11B:

1. Write a "slow poke" program that prints out a 3-word message with several seconds between each word.
3. Write a digital clock program. It uses a timing loop to count seconds. Input the present time in hours, minutes, and seconds. The clock then counts seconds and prints them out. When 60 seconds have gone by, add one to the minutes and set the seconds back to zero. Same with hours. Run the clock a long time and adjust the timing loop so the clock keeps good time.

HOW BIG A SPACE CAN TAB() MAKE?

There are 22 spaces across the screen. You can use any number (1 through 22) inside the TAB( ) parentheses. Larger numbers make the computer skip lines. Numbers larger than 255 will give an error message when the program runs:

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

where XX is the line number.

You can use TAB with strings too:

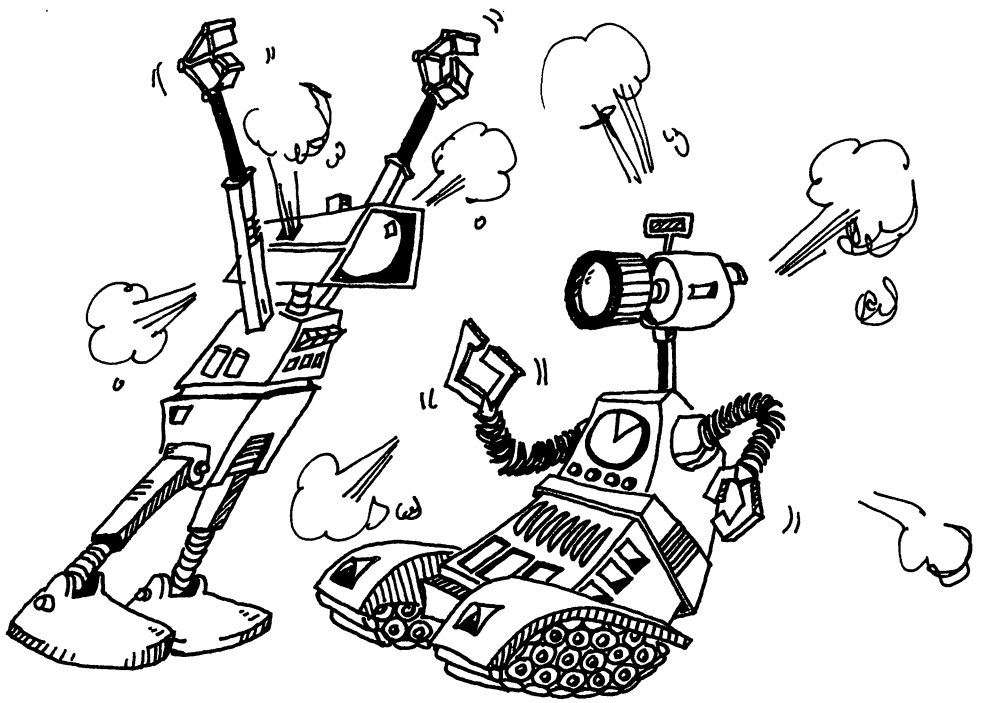
Example:

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

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

FUNCTIONS DON'T FIGHT BUT THEY HAVE ARGUMENTS

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



Assignment 11A:

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

INSTRUCTOR NOTES 12 THE IF COMMAND WITH NUMBERS

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

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

The use of nested IF's is discussed.

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

QUESTIONS:

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

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

4. Same question, but for 3 > 7.

LESSON 12 THE IF COMMAND WITH NUMBERS

Try this:

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

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

```
10 F     something A is true     THEN     do command C
```

"Something A" can have these arithmetic symbols:

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

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

```
A <> B     is pronounced     "A is not equal to B"

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

PRACTICE

For these examples, LET A=7 and LET B=5 and LET C=5. Say each "something A" out loud and tell if it is true or false:

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

AN IF INSIDE AN IF

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

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

To understand this, break it into two parts:

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

(command C) is (IF A<20 THEN PRINT "TEENAGER!")
```

This line first asks "is the age greater than 12?"

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

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

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

Assignment 12A:

1. Draw the "fork in the road" diagram for line 60 above. There will be two forks on the diagram. (See page 49 .)

GUESSING GAME

```
10 REM GUESSING GAME
20 PRINT "clr TWO PLAYER GAME"
30 PRINT "dn red FIRST PLAYER:"
31 PRINT "      HIDE YOUR EYES!"
32 PRINT "dn grn SECOND PLAYER:"
34 PRINT "      ENTER A NUMBER"
35 PRINT "      FROM 1 TO 100 dn"
40 INPUT N
45 PRINT"clr"
50 PRINT "db blk 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 GAME OVER
95 PRINT "clr red dn dn dn THAT'S IT! blk"
```

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

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

Assignment 12B:

1. Tell what happens in lines 50 through 80:

If G is 31 and N is 88:

50 _____

55 _____

60 _____

65 _____

70 _____

80 _____

If G is 88 and N is 88:

50 _____

55 _____

60 _____

65 _____

70 _____

80 _____

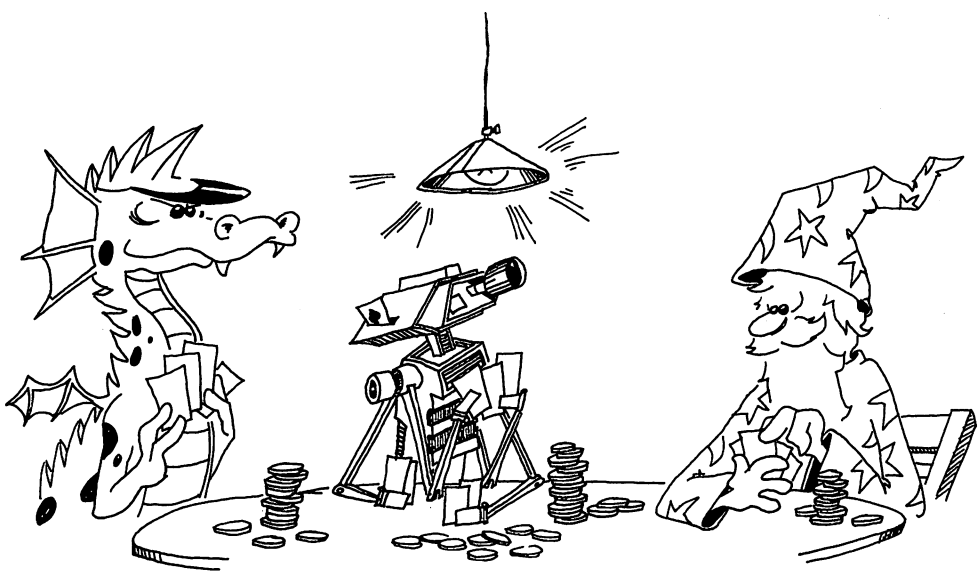
2. Here is another program. What will it print, and how many times?

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

What will it print if line 10 is changed?

```
10 LET N=2
```

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



INSTRUCTOR NOTES 13 RANDOM NUMBERS AND THE INT FUNCTION

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

The RND function produces psuedo-random decimal numbers larger than 0.0 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 a suit of cards are often more to the point.

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

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

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

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

QUESTIONS:

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

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

   and the box G contains: 2, 2.1, 2.95, 3.001, 67, 0, 0.2.
2. Tell how the INT() function is different from "rounding off" numbers. Which is easier for you to do?
3. Tell how to change a number so that the INT() function will round it off.
4. What does the RND(1) function do?
5. How can you get random integers (whole numbers) from 0 through 10. (Hint: INT(RND(8)*10) is not quite right.)
6. How can you get random integers from 5 through 8?

LESSON 13 RANDOM NUMBERS AND THE INT FUNCTION

THE RND FUNCTION

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

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

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

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

Run this program:

```
10 REM RANDOM NUMBERS
20 PRINT"clr dn dn"
25 LET N=RND(1)
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.

Always put the number "1" in the parentheses of the RND function.

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

Change the program above to:

```
25 LET N=RND(1)*52
40 IF N<45 THEN GOTO 25
```

and run it again.

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

But:

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

THE INT FUNCTION

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

Try the INT function in this little program:

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

And in this:

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

And this:

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

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

Try this:

```
10 REM --- INT ---
20 PRINT"clr"
30 PRINT"dn DECIMAL NUMBER?dn"
32 INPUT D
35 LET I=INT(D)
40 PRINT "dn DECIMAL ";D
45 PRINT "INTEGER ";I
50 IF I<>0 THEN GOTO 30
```

Enter 0 to end the program.

ROLLING THE BONES

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

```
10 REM ///ONE DIE///
20 PRINT"clr dn dn dn"
30 LET R=RND(1)
40 PRINT "RANDOM NUMBER"
42 PRINT "          ";R
50 LET S=R*6
55 PRINT "dn TIMES 6"
56 PRINT "          ";S
60 LET I=INT(S)
65 PRINT "dn INTEGER PART"
66 PRINT "          ";I
70 LET D=I+1
75 PRINT "dn DIE SHOWS"
76 PRINT "          ";D
80 PRINT "dn red ANOTHER? <Y/N> dn blk"
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(1))
```

Here is how to save a lot of room.

Instead of:

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

Use just:

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

Assignment 13:

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

INSTRUCTOR NOTES 14 SAVING TO TAPE

We explain how to use the cassette recorder to save programs.

This lesson can be used any time after lesson 3. We put it this late in the book because most programs up to this point are relatively short and uninteresting, not worth saving. The process of programming was being emphasized, not the end result of useful programs.

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

The Commodore tape recorder is just like ordinary recorders that play music, but the computer can turn it on and off automatically.

Ordinary audio tape cassettes can be used. However, the tape must be able to record perfectly. Even one tiny bad spot (such as a fold in the tape) will "drop a bit" and the recorded program will be wrong.

Short tapes are best. Programs use from 20 seconds to 5 minutes worth of tape. You will put one long program or several short ones on one tape. You do not want to fill a 30 minute tape with programs, because it would take a long time to get to the last program, even using fast forward.

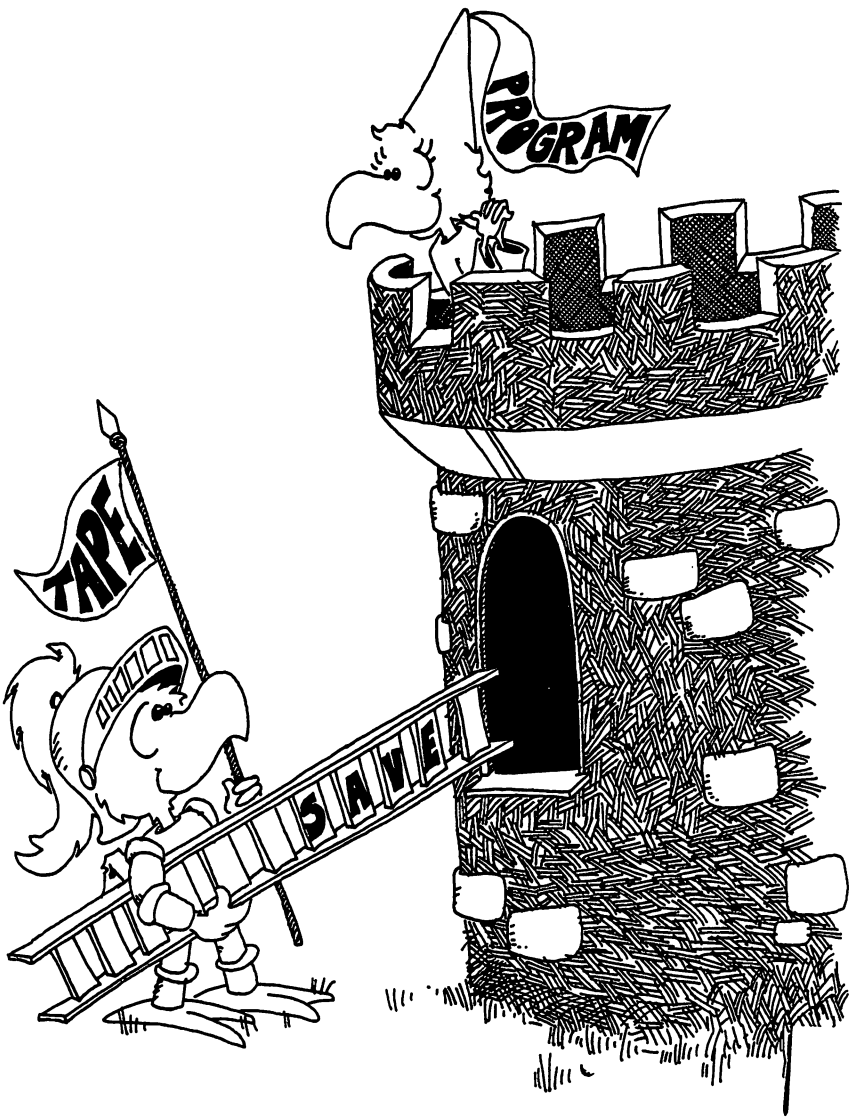
QUESTIONS:

1. What is a "file"?
2. How long can a file name be?
3. How do you check that the program got onto the tape OK?
4. What happens to the program already in memory if you LOAD another program?
5. Does the file name have to be the same as the program name?
6. Write a short program, SAVE it, VERIFY it, do NEW and then LOAD it.
7. If a program is put into a file, is it still in memory?

LESSON 14 SAVING TO TAPE

SAVING YOUR FIRST PROGRAM TO CASSETTE TAPE

Here is how to save one program on a new tape without using a file name.



FOLLOW THESE STEPS

1. List your program to make sure it is still there.
2. Put a new tape in the recorder and rewind it.
3. Enter: `SAVE`
4. The computer will print:

```
PRESS PLAY & RECORD ON TAPE
```

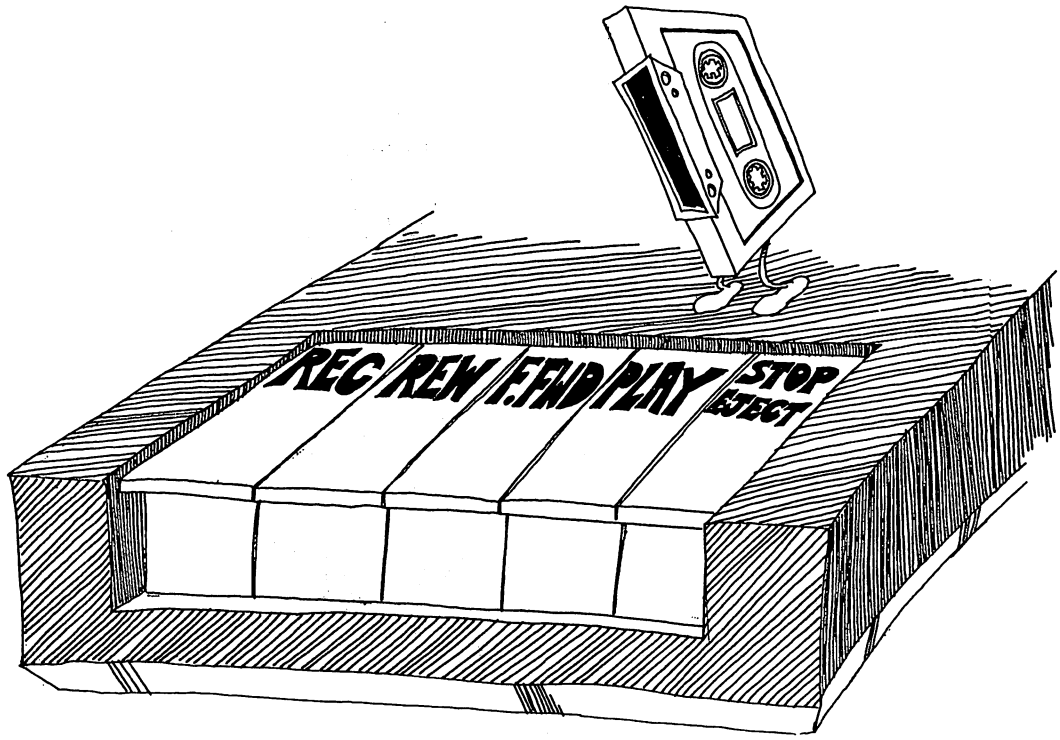
5. Hold down the REC button and press the PLAY button. Both will click and stay down.
6. The computer will print:

```
OK
SAVING
```

7. The computer waits for about 10 seconds to put a "leader" on the tape, then records the program.
8. When it is done recording, it will print the usual Edit Mode prompt:

```
READY.
```

and turn off the tape motor. But the REC and PLAY keys stay stuck down! You should click them back up by pressing the STOP key on the recorder. (Not the STOP key on the computer!)



CHECK TO SEE IF THE PROGRAM IS ON TAPE OK

Now we will check that the program on tape is exactly the same as the program in the computer's memory. Do this:

1. Rewind the tape. Then press STOP to put all the keys on the recorder up.
2. (You may LIST the program to see that it is still in memory.)
3. Enter: `VERIFY`
4. The computer will print:

```
PRESS PLAY ON TAPE
```

5. Do so. The computer will print:

```
OK
SEARCHING
```

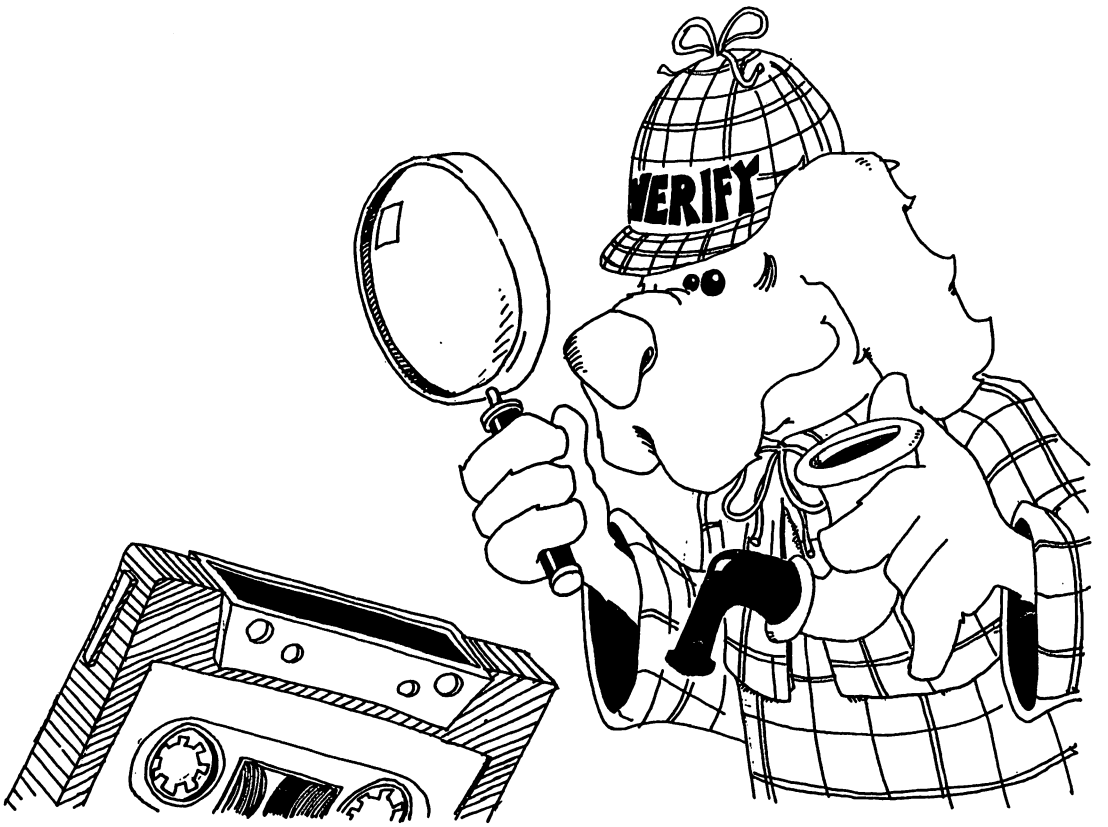
6. In a moment, it will print:

```
FOUND
VERIFYING
```

7. After short wait, the computer says:

```
OK or ?VERIFY ERROR
READY.
```

If the computer says "?VERIFY ERROR", this means the program in memory is not EXACTLY the same as the one on tape. If the one in memory is good, then the one on tape is no good.



LOADING A PROGRAM FROM TAPE INTO THE COMPUTER MEMORY

Let's practice by loading the program we just saved. We will not use a file name. The computer will load the first program it finds on the tape.

CAUTION! Any program already in the computer will be erased when the new program is put in!

1. Rewind the tape. Check that the recorder keys are up.
2. Enter: `LOAD`
3. The computer will print:

```
PRESS PLAY ON TAPE
```

4. Start the recorder by pressing the PLAY key.
5. The computer will print:

```
OK

SEARCHING
FOUND

LOADING
READY.
```

   With delays between words as the tape runs.
6. Now do a LIST to see if the program is in the computer.

WHAT DOES THE PROGRAM SOUND LIKE?

Why don't you put it on an ordinary recorder and find out?

USING FILE NAMES

For practice, we will write a few very short programs and save them on the same tape. Enter NEW and:

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

It is a good idea to think of a name for each program and put the name in a REM at the beginning of the program.

1. Rewind a new tape and enter:

   ```
   SAVE "HOT DOG"
   ```

   We choose "HOT DOG" for the file name.
   The file name can be up to 16 characters long.
   It is a good idea to choose the file name the same as the program name. It is easier to remember.
   The file name will be put on the tape along with the program.
2. The computer says:

   ```
   PRESS RECORD & PLAY ON TAPE
   ```

3. Do so. The computer says:

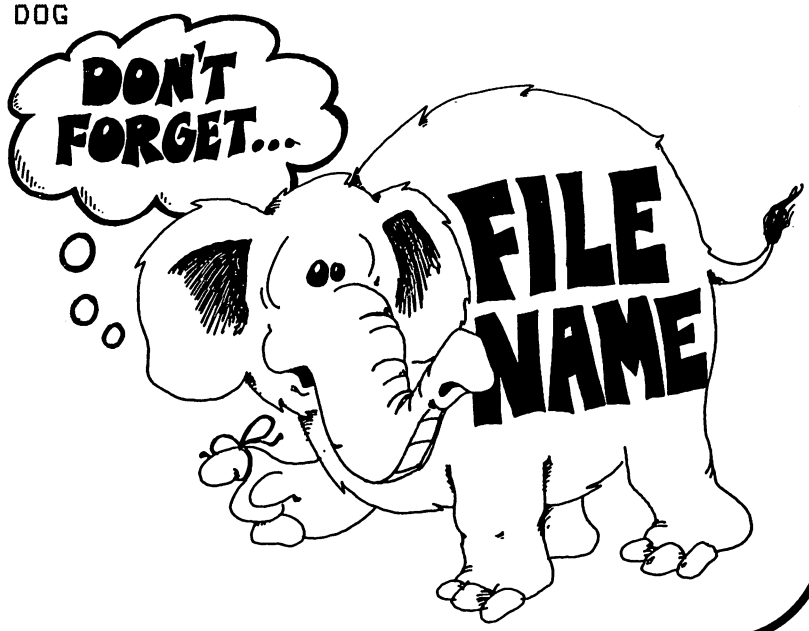
   ```
   OK
   SAVING HOT DOG
   READY.
   ```

4. Rewind the tape and enter:

   ```
   VERIFY "HOT DOG"
   ```

   The computer says:

   ```
   PRESS PLAY ON TAPE
   OK

   SEARCHING FOR HOT DOG
   FOUND HOT DOG

   VERIFYING
   OK

   READY.
   ```



5. We are ready to put the next program in. Enter:

```
NEW
10 REM MICE
20 PRINT "LIKE CHEESE"
```

6. DON'T rewind! Make sure the recorder keys are up. Enter:

```
SAVE "MICE"
```

7. Press REC and PLAY when the computer asks you to. When the computer prints READY, rewind the tape and enter:

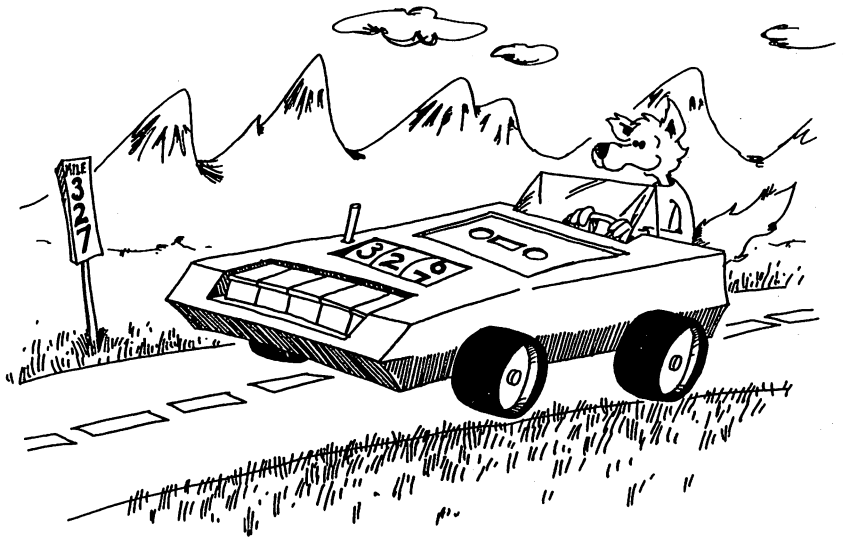
```
VERIFY "MICE"
```

   You will see:

```
PRESS PLAY ON TAPE
OK

SEARCHING FOR MICE
FOUND HOT DOG

FOUND MICE

VERIFYING
OK

READY.
```

8. Don't rewind. Try this: Write another program called ICE CREAM. Put it on the tape after MICE. Rewind the tape and VERIFY "ICE CREAM".

Careful! If you try to SAVE after performing a LOAD or a VERIFY without setting the PLAY key up, the computer will start up the recorder motor and think it is recording on the tape. But the REC key is not down, and it is NOT saving your program on tape!



THE RECORDER HAS A "TRIP MILEAGE INDICATOR" LIKE A CAR

The recorder has a little counter to keep track of where you are on the tape. When you rewind the tape, push the button beside the counter to make it read zero.

Then before you SAVE a program, write the counter number and the program name on the cassette label so you will know where on the tape the program starts.

When you LOAD, use the "fast forward" key (F.FWD) to get close to the spot where your program is on the tape.

WHERE IS THE END OF THE TAPE?

Are you the careless type? Do you sometimes forget what programs are on your tapes? Here is how to find out:

1. Rewind the tape and zero the counter. Then enter:

```
VERIFY          and press PLAY
```

2. The computer will write the name of the first program on the screen. Then it will verify the program and write:

```
?VERIFY ERROR
```

3. Now you know the name of the first program and the counter reading at its end. Do VERIFY again and get the second program's name.
4. Keep doing this until the computer doesn't find any more programs. The last counter number you wrote down shows where you should start if you want to put another program on that tape.

Assignment 14:

1. SAVE, LOAD, and VERIFY several small programs on the same tape.
2. How do you find where on the tape the last program ends (so you can put a new one on the end)?

GRAPHICS, GAMES, AND ALL THAT

INSTRUCTOR NOTES 15 SOME SHORTCUTS

This lesson covers:

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

The sprint is over. We have reached RND and the saving of programs on tape. All the elements are in place for the student to write substantial programs.

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

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

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

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

A question mark is always printed on the screen by INPUT. So when INPUT contains a question as a message, it should not end with a question mark, because that will be supplied automatically.

QUESTIONS:

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

LESSON 15 SOME SHORTCUTS

A PRINT SHORTCUT

Instead of typing PRINT, just type a question mark.

Enter:

```
10 ? "HI"
LIST 10
```

The computer substitutes the word PRINT for the question mark.

A LET SHORTCUT

These two lines do the same thing:

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

also these two:

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

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

AN INPUT SHORTCUT

Instead of:

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

You can do:

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

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

Examples:

```
10 INPUT "AGAIN ? <Y OR N>"; Y$
20 INPUT "LOCATION"; X,Y
30 INPUT "MONTH, DAY, YEAR";M$,D,Y
```

A LIST SHORTCUT

There are 5 ways to use the LIST command:

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

A COLON SHORTCUT

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

Instead of

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

you can write:

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

In memory this line looks like:

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

WHEN TO USE THE COLON SHORTCUT

Use the shortcut:

1. To make the program clearer.

   Put similar statements on the same line. Example:

   Instead of:

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

   write:

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

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

   Example:

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

THE COLON AFTER AN IF COMMAND

You can make neater IF statements using colons.

Without:

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

With colons:

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

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

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

Example:

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

is not the same as:

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

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

SOME MORE MISTAKES WITH COLONS

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

Correct:

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

Wrong:

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

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

Correct:

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

Wrong:

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

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

COMMANDS, STATEMENTS AND LINES

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

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

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

Enter: `NEW` We say we have "entered a command."

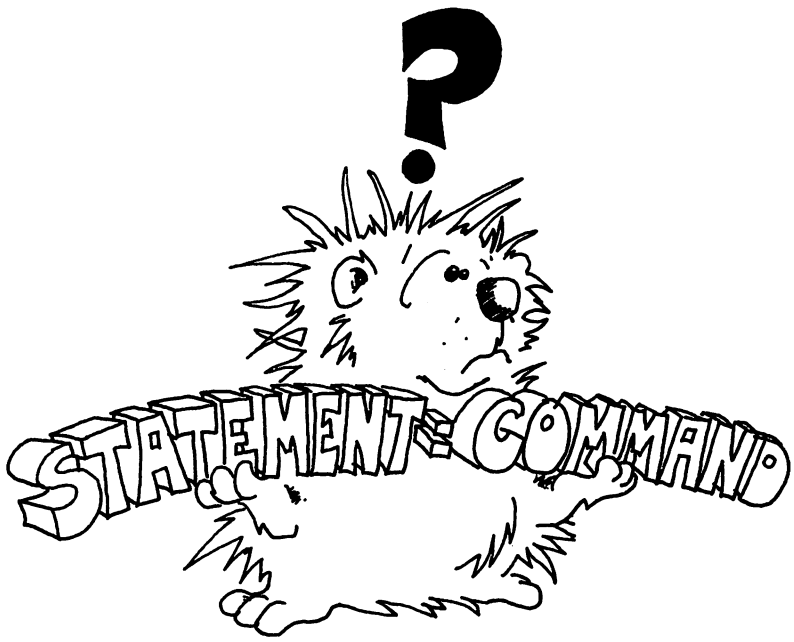
But if we write this line in a program:

`2 NEW` We say that line 2 has one "statement", the NEW command.

Some lines have several statements, separated by colons.

```
30 PRINT"clr":PRINT:LET Z=55
```

is a line with three statements.



Assignment 15:

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

INSTRUCTOR NOTES 16 MOVING PICTURES

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

Also, if you try to move the cursor lower than the bottom of the screen, the whole picture will scroll.

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

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

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

QUESTIONS:

1. What is the difference between "clr" and "hm" in a PRINT command?
2. Show two lines in a program that together will put a letter "A" on the screen at the point 3 lines down and 7 spaces across.
3. In this line, the word "HAPPY" is printed. Where is the PRINT cursor after the word is printed?

```
10 PRINT"hm dn HAPPY";
```

   Can you see the PRINT cursor on the screen?
4. Now write a line 20 that will erase the letters "APP" from the word printed in question 3.

LESSON 16 MOVING PICTURES

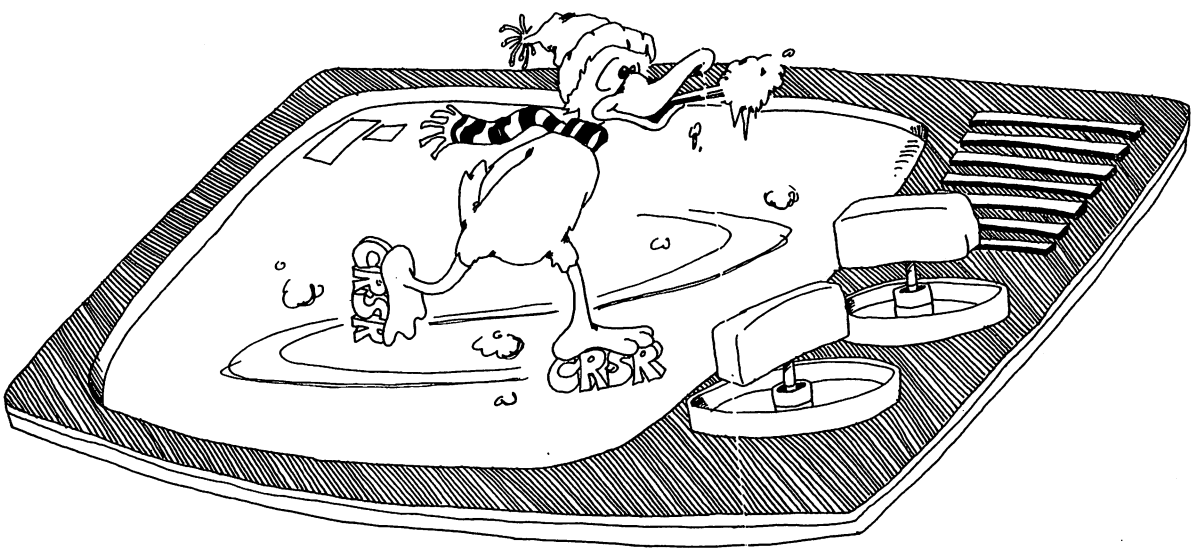
MOVING THE CURSOR UP AND DOWN

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

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

When we print "dn dn" in this book, we mean to put two down CRSR characters in the PRINT quotes. Do not put in any spaces! We write "dn dn" instead of "dndn" so you can easily read it. (If we want you to put a space, we will write "sp".)

Remember that the "dn" character looks like an inverse "Q", "clr" like an inverse heart and "hm" like an inverse "S".



TO REACH A SPOT ON THE SCREEN:

1. Start your PRINT line with a home cursor "hm" or a clear screen "clr".
2. Then move the cursor down as far as you want.
3. Then PRINT right CRSR characters to move over as far as you want.
4. Then PRINT what you want.

A MOVING PICTURE

Try this program:

```
10 REM ::: BIRD :::
20 PRINT"clr dn dn dn dn dn dn";
25 PRINT"rt rt rt rt rt rt rt pur";
30 PRINT" lf lf lf";
40 FOR T=1 TO 200:NEXT T
50 PRINT" lf lf lf";
60 FOR T=1 TO 200:NEXT T
70 GOTO 30
```

Here "lf lf lf" means three left CRSR characters (remember to use the SHIFT key). We use "rt rt rt" for right CRSR characters.

Do you remember what to do when you see "pur"? If you forgot, look at Lesson 2.

Which lines are the delay loops?

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

How does it work?

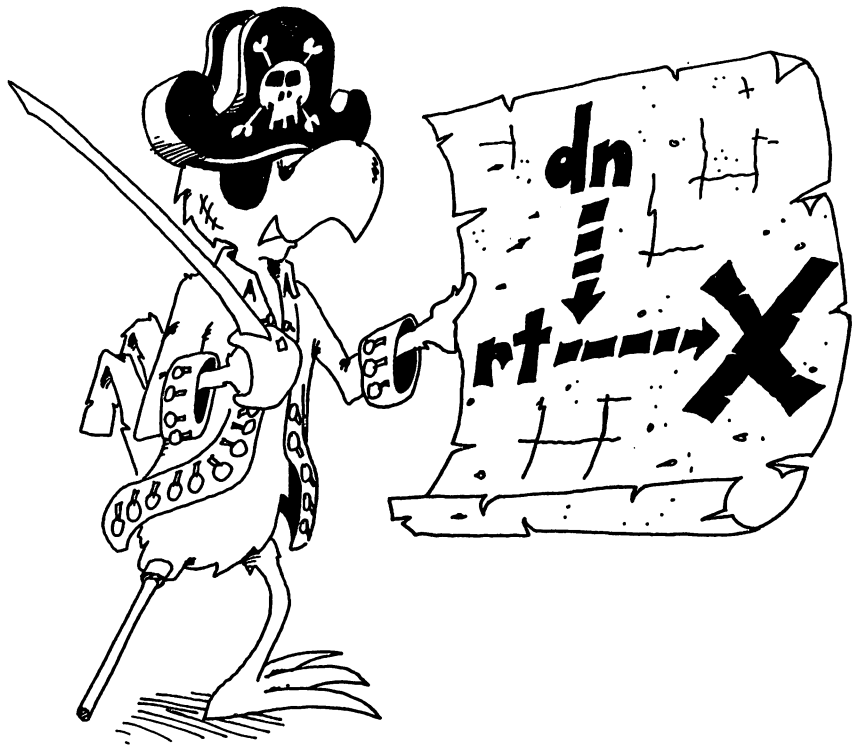
Read lines 30 and 50 character by character.

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

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

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

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



ERASING OLD PICTURES

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

Suppose we want the bird to fly away?

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

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

```
10 REM FLY AWAY
20 PRINT"clr dn dn dn dn ... dn";              (22 of them)
25 PRINT"pur";
30 PRINT"   lf lf lf";        :REM  PRINT DOWN WINGS
40 FOR T=1 TO 200:NEXT T      :REM  DELAY LOOP
45 PRINT"sp sp sp lf lf up";:REM  ERASE BIRD, MOVE UP
50 PRINT" lf lf lf";        :REM  PRINT UP WINGS
60 FOR T=1 TO 200:NEXT T
65 PRINT"sp sp sp lf lf up";
70 GOTO 30
```

Assignment 16:

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

INSTRUCTOR NOTES 17 FOR-NEXT LOOPS

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

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

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

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

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

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

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

Jumping into the middle of a loop is usually a disaster. Jumping out of a loop before NEXT causes an exit is commonly done, but in some cases (especially where subroutines are involved) may give hard to find bugs.

QUESTIONS:

1. Write a loop that prints the numbers from 0 to 20.
2. Write a loop that prints the numbers from 30 down to 20, by twos.
3. Write a pair of nested loops to print the numbers 100, 200, 300 and between them, the letters A, B, C, D, E on separate lines.

LESSON 17 FOR-NEXT LOOPS

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

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

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

Run this:

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

The loop can start on any number and end on any higher number.

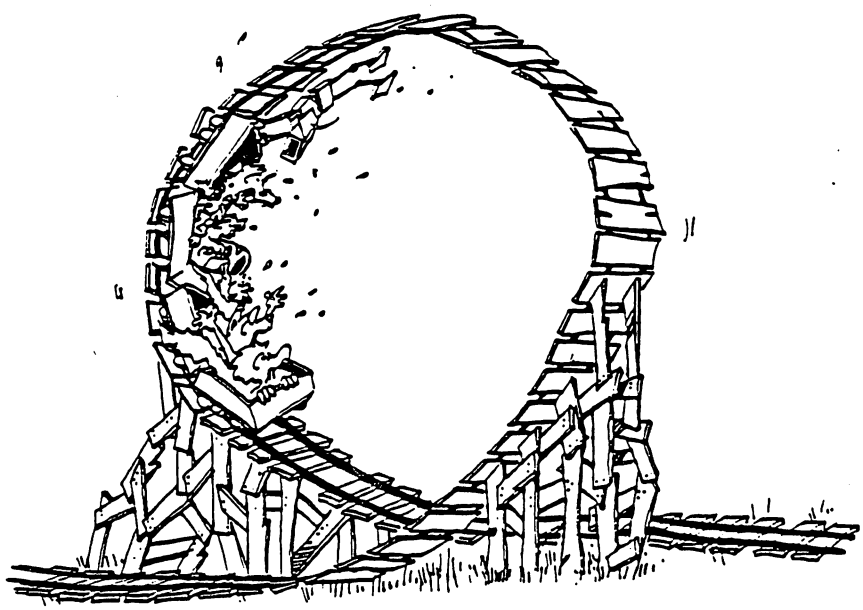
Try changing line 30 in these ways:

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

MARK UP YOUR LISTINGS

Show where the loops are by arrows:

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



THE STEP COMMAND

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

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

Assignment 17A:

1. Have the computer count by five's from 0 to 100.

COUNT DOWN LOOPS

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

Try this:

```
10 REM **APOLLO 11**
20 PRINT"clr"
30 PRINT"clr T MINUS 12 SECONDS dn dn dn sp sp sp
       sp sp AND COUNTING"
35 FOR T=1 TO 2000:NEXT T
40 FOR I=11 TO 0 STEP -1
50 PRINT "clr dn";I
60 FOR T=1 TO 2000:NEXT T:REM TIMING LOOP
70 NEXT I
80 PRINT "clr dn red ALL ENGINES RUNNING. dn dn dn
        sp sp sp LIFT OFF."
81 FOR T=1 TO 2000:NEXT T
82 PRINT "clr dn grn WE HAVE A LIFT OFF."
83 FOR T=1 TO 2000:NEXT T
84 PRINT "clr dn yel sp sp 32 MINUTES dn dn dn
         sp sp . . .sp                PAST THE HOUR."
85 FOR T=1 TO 2000:NEXT T
86 PRINT "clr dn cyn LIFT OFF ON APOLLO 11."
```

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

NESTED LOOPS

In this program, we have one loop inside another.

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

The inside loop is in line 60.

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

LOOP VARIABLES

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

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

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

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

I is the loop variable.

BADLY NESTED LOOPS

The inside loop must be all the way inside:

Right:

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

Wrong:

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

Assignment 17B:

1. Write a program that prints your name 15 times.
2. Now make it indent each time by 2 spaces more. It will go diagonally down the screen. Use TAB in a loop.
3. Now make it write your name 23 times, starting at the bottom of the screen and going up. Use the "up" keys in a PRINT and put it all in a loop.
4. Now make it write your name on one line, your friend's name on the next and keep switching until each name is written 5 times.

INSTRUCTOR NOTES 18 EDIT AND RUN MODES, THE CALCULATOR

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

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

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

```
PRINT and RUN
```

Other names for these modes are:

Edit mode:	Direct mode Calculator mode	Immediate mode Command mode
Run mode:	Program mode	Deferred Execution mode

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

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

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

QUESTIONS:

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

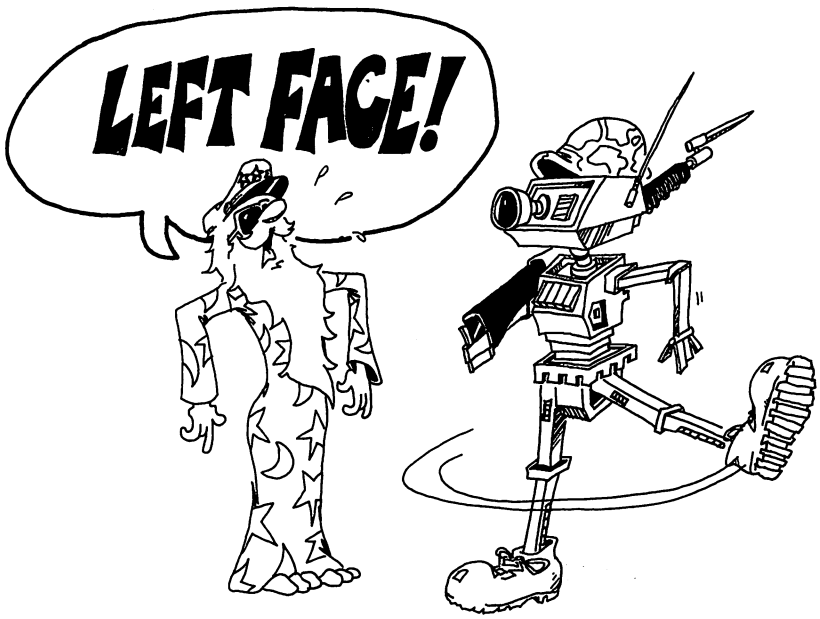
LESSON 18 EDIT AND RUN MODES, THE CALCULATOR

Enter NEW
Press the SHIFT and CLR keys
You are ready to begin the lesson.

EXECUTION AND RUNNING

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

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



DEFERRED EXECUTION

Enter and run this program:

```
10 PRINT "HI"
```

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

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

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

IMMEDIATE EXECUTION

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

```
PRINT "HI"
```

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

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

Try this longer example:

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

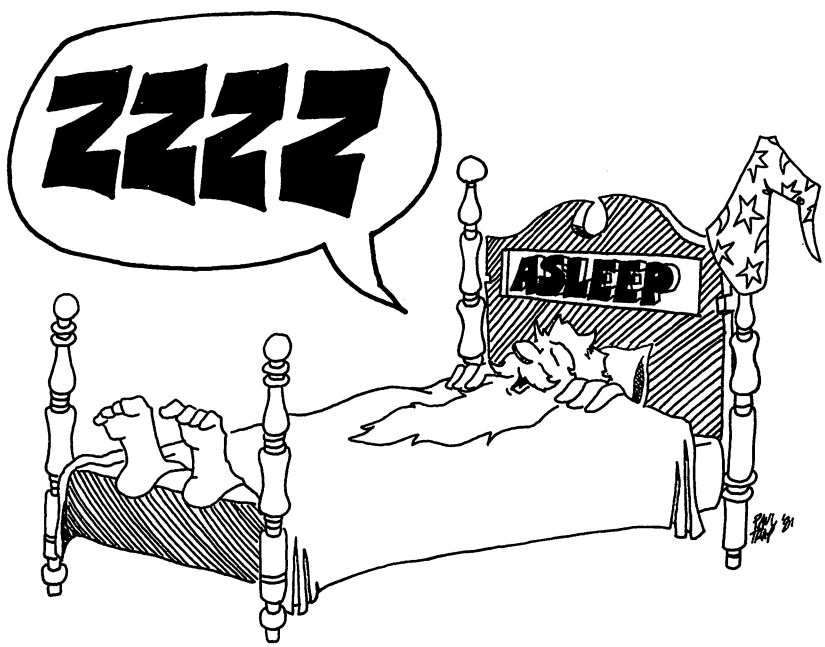
Rule: In immediate execution you can run a one line program that has several statements separated by colons. (But the computer forgets the program as soon as it is done running.)

ASLEEP OR AWAKE?

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

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

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



THE EDIT MODE

Press the STOP and RESTORE keys.

The computer says READY. This word is called a "prompt" and the computer prints it each time it enters the "Edit mode" of BASIC. "READY" is the "snoring" of the computer when it is in the Edit mode.

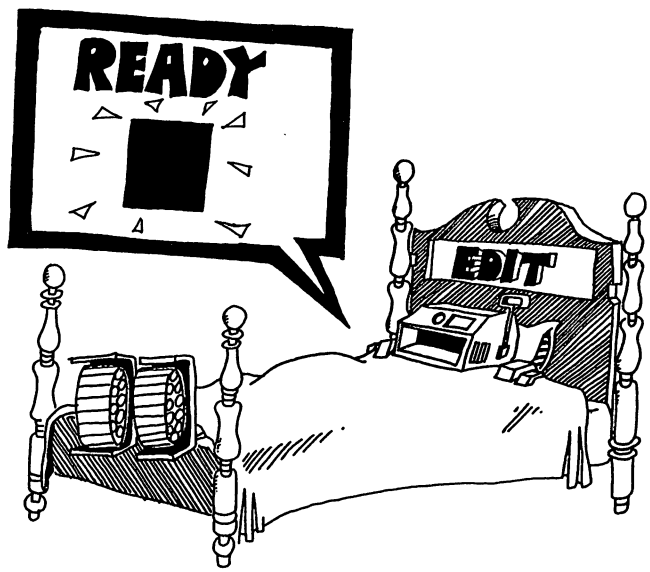
The flashing square is called the "cursor." It tells us that the computer is waiting for us to type something. The next letter we type will be put on the screen at the flashing cursor.

While the computer is in the Edit mode:

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

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

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



THE RUN MODE

Enter RUN to leave the Edit mode and go to the RUN mode.

While the computer is in the RUN mode:

The program in memory runs.

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

Assignment 18:

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

INSTRUCTOR NOTES 19 SOUND

This lesson shows how to produce sound using the POKE command and the built in hardware of the VIC.

The VIC has four voices. The first three produce musical notes. The fourth produces a "shh" noise.

Sound is sent to the TV speaker as part of the composite signal. If you use a color monitor, you need to consult the VIC manual to see how to send the sound signal to an amplifier and speaker.

A nice aspect of the VIC sound is that after a tone is turned on, it continues without requiring any more computing. This is in contrast to many other computers where the sound is produced by the central processor, and to have sound and moving graphics at the same time requires very sophisticated programming.

QUESTIONS:

1. What two things must you POKE to get a sound?
2. How do you turn off all the voices at once?
3. How do you turn on voice 1 and 2 and turn off voices 3 and 4?
4. Why must you be careful of the box number that you POKE?
5. What number gives a high pitch? What gives a low pitch?
6. What number gives the loudest sound?
7. Which voice has the highest pitch? Which the lowest pitch?

LESSON 19 SOUND

If your VIC computer is hooked up to a TV, you can make interesting sound effects and music. Turn up the sound on the TV now.

If your VIC is hooked up to a video monitor, you can make sounds by connecting your computer to an amplifier and speaker.

SOLO VOICE

Enter this:

```
10 POKE 36874,200  :REM VOICE 1, PITCH 200
20 POKE 36878,15   :REM VOLUME 15
85 FOR T=1 TO 1000 :NEXT T
90 POKE 36874,0    :REM VOICE OFF
```

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

Run it. You should hear a low tone.

The number 36874 is the name on the front of a special memory box. The computer goes to that box to see what tone to have voice number 1 make.

You can hear tones for numbers 128 to about 255. Tone 0 turns off the voice.

The number 36878 is the name of a box that tells how loud the sound should be. Put 1 for softest, 15 for loudest, and 0 for all 4 voices off.

POKE is a command that puts a number between 0 and 255 in a memory box.

`POKE 36874,200` means put number 200 in box 36874

Line 85 tells how long the tone is on.

FOUR SOLO VOICES

Voice		box number	contents
1	lowest	36874	0, 128 to 255
2	middle	36875	0, 128 to 255
3	highest	36876	0, 128 to 255
4	raspy	36877	0, 128 to 255
	Volume	36878	0 to 15

Run:

```
10 REM SOLO VOICES
20 SO=36873          :REM BASE ADDRESS OF VOICE
30 V =36878          :REM VOLUME CONTROL
40 INPUT"WHICH VOICE? <1-4> ";S
42 IF S< 1 THEN GOTO 40
44 IF S> 4 THEN GOTO 40
46 SV=SO+S           :REM CHOOSE VOICE
50 PRINT:INPUT"HOW LOUD? <0-15> ";L
55 PRINT"WHAT PITCH? <128-255> "
60 INPUT P
70 POKE  V ,L        :REM CHOOSE LOUDNESS
80 POKE  SV,P        :REM CHOOSE TONE
85 FOR T=1 TO 1000:NEXT T
90 POKE SV,0         :REM TURN OFF VOICE
95 GOTO 40
```

This program lets you choose which voice to play, how loud, and what pitch.

Line 40 lets you pick a voice number 1 to 4.

Line 46 adds this voice number to the base number 36873 to get a box number between 36874 and 36878.

Lines 42 and 44 help make sure you do not accidently choose the wrong number to POKE.

Because you put the box numbers in lines 20 and 30 just once, CAREFULLY, you are less likely to make a mistake in a POKE command.

It is also quicker to type just the variable names V and SV, than long numbers like 36874.

FOUR VOICE CHOIR

Remove line 90 from the above program and you can turn on each voice, one after another.

All four voices can play at once.

You can turn all four off by choosing 0 for the loudness.

Or you can turn off any voice by choosing 0 for its tone.

MUSIC

The computer can play music if you choose the tones for each voice correctly. (However, it will not win any prizes for perfect pitch. Each note is only approximately correct.)

C	135	G	215
C#	143	G	217
D	147	A	219
D#	151	A#	221
E	159	B	223
F	163	C	225
F#	167	C#	227
G	175	D	228
G#	179	D#	229
A	183	E	231
A#	187	F	232
B	191	F#	233
C	195	G	235
C#	199	G#	236
D	201	A	237
D#	203	A#	238
E	207	B	239
F	209	C	240
F#	212	C#	241

The lowest octave of these three is likely to be most nearly in tune.

SOUND EFFECTS

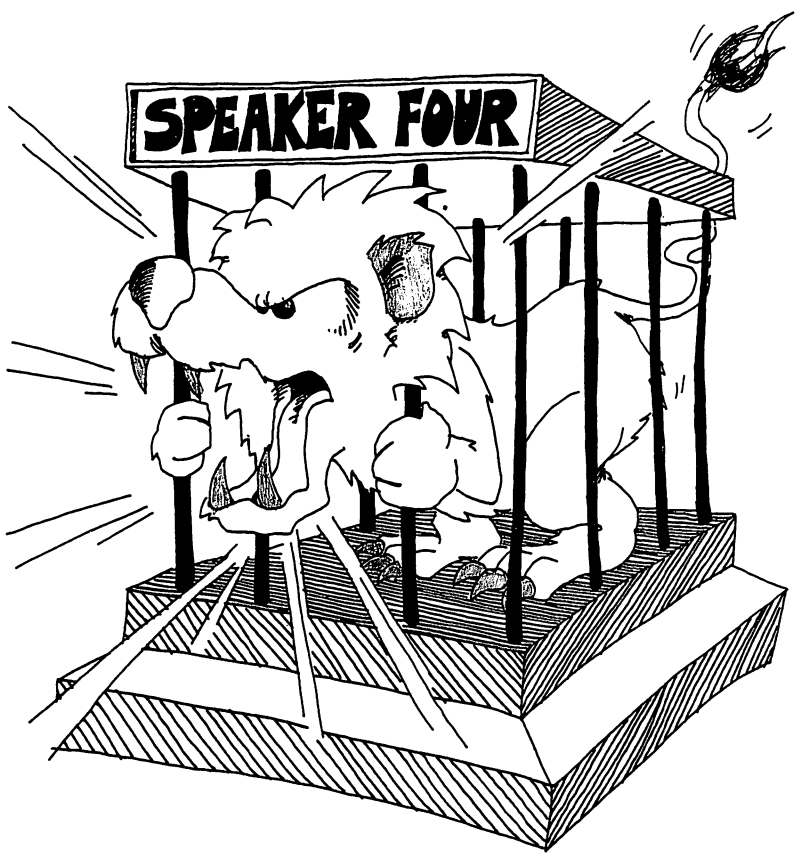
The manual that came with your VIC computer has 20 examples of sound effect programs. You can start with these and modify your programs to get other effects.

Try this:

```
10 REM GOING UP
20 S=36876            :REM VOICE 3
30 V=36878
40 POKE V,15
50 FOR P=128 TO 254
60 POKE S,P
70 FOR T=1 TO 100:NEXT T
80 NEXT P
```

Assignment 19:

1. Try your own version of these sound effects: laser gun, wind (use voice 4), siren, and explosion.
2. Write a program to show a flying bird; make the bird chirp as it goes.
3. Make the computer play a simple tune, like Row, Row Your Boat, or Mary Had A Little Lamb. You may want to look up the notes in a music book, or just "wing it."



INSTRUCTOR NOTES 20 VARIABLE NAMES

This lesson treats the rules for naming variables.

Descriptive variable names help clarify programs. On the other hand, they take more typing and there are two hidden "gottcha's."

One "gottcha" is that BASIC only records the first two letters of a name. This means that those beautiful long descriptive names may not be unique. Even experienced programmers occasionally get caught here, and the bug may be quite hard to detect.

The other is that "reserved words" may not be included in the name anywhere: start, middle, or end. The reserved words are the BASIC commands and functions.

When you execute a line that has a variable containing a reserved word, you will get a ?SYNTAX ERROR IN XX message. It is not obvious from this message what is wrong. The only correction that works is choosing a new name.

It is recommended that only one and two letter names be used.

A list of reserved words in VIC BASIC is presented in the Appendix.

QUESTIONS

1. Names longer than two characters may give you trouble. Why?
2. Names that have numbers in them may be good names or wrong names. How can you tell if they are good?
3. Names cannot have punctuation marks in them, except in one case. What punctuation mark is allowed, where do you put it, and what does it tell you?
4. Long names may have "reserved words" at the front, inside, or at the back. Which words are reserved?" Where is there a list of reserved words?

LESSON 20 VARIABLE NAMES

OLD RULES

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

MORE RULES

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

RIGHT AND WRONG NAMES

Some correct names:

```
A77
LEFTSIDE
X3AB
APE
NAME$
D5$
```

Some wrong names:

```
2X
X#
S%$
$NAME
3RDNAME$
LIST
COLOR          (has OR in it!)
GIFT           (has IF in it!)
TOM            (has TO in it!)
```

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

```
10 TOM = 1
```

and then

```
RUN
```

You will see

```
?SYNTAX ERROR IN 10
```

DIFFERENT NAMES THAT ARE THE SAME

Breaking the rule that:

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

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

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

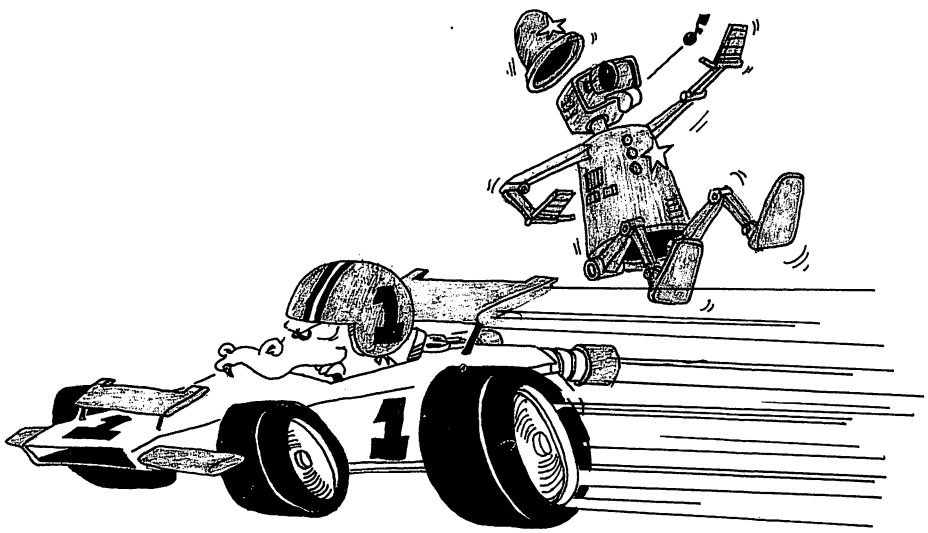
Try this program:

```
10 HOP$ = "BUNNY"
20 HOSE$ = "LONG AND NARROW"
30 PRINT HOP$
```

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

Assignment 20:

1. Read the rules numbered 1 through 8. For each rule make up two names, one that is correct, and one that disobeys the rule. (Rule number 6 has no wrong name.) Try each name in a line like 10 NAME=1 or 10 NAME$="A" to see if it is a legal name.



INSTRUCTOR NOTES 21 COLOR GRAPHICS

This lesson runs through most of the color features of the VIC.

The next lesson finishes with the POKE commands that put graphics characters on the screen in color.

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

Screen border and background color are controlled by a POKE to the same memory address: 36879. Not all numbers in the range 0 through 255 give acceptable results. Consult your VIC manual, page 134, for a table of good numbers. These numbers can be calculated by the formula given in the lesson.

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

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

If your program has POKEs, it is wise to save it to tape First, before running it. Then if you have a system crash, you can cold start and load the program and be exactly at the point before the crash.

QUESTIONS:

1. How many different colors can the border have?
2. How many colors for the screen background?
3. How many colors for the printing?
4. What does memory box number 646 control?
5. What two things does box number 36879 control?
6. What danger is there in using POKE?

LESSON 21 COLOR GRAPHICS

ADJUST YOUR COLOR TV OR MONITOR

Put all colors on screen like this:

1. First press CTRL and RVS ON. (Now "space" will be a colored spot.)
2. Press CTRL and RED key; then hold down SPACE BAR to make a red strip on screen.
3. Repeat with the rest of the color keys; observe 6 colored stripes on the screen.
4. Adjust color controls on TV or monitor until each color looks correct.

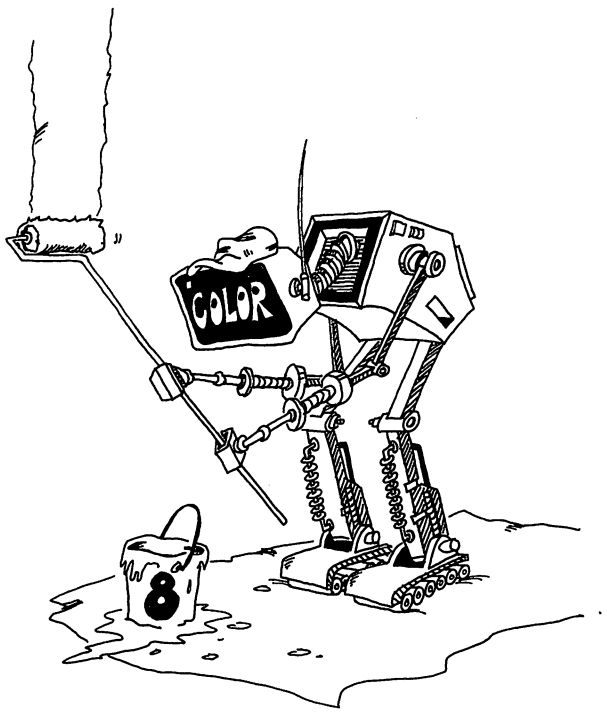
COLOR SCREEN AND BOARDER COMBINATIONS

There is a special memory box which tells the VIC what colors to show on the border and on screen background.

Its box number is 36879. (Yes, it is a neighbor of the box that tells how loud the sound should be.)

Run:

```
10 REM COMBINATIONS
20 BOX = 36879
25 PRINT"clr dn dn dn"
30 PRINT"WHAT COLOR SCREEN? <0 TO 15>"
35 INPUT CS
37 POKE BOX,8+16*CS+CB
38 PRINT"clr dn dn dn"
40 PRINT"WHAT COLOR BORDER? <0 TO 7> "
45 INPUT CB
50 POKE BOX,8+16*CS+CB
90 GOTO 25
```



SCREEN COLORS

0 BLACK	8 ORANGE
1 WHITE	9 LT. ORANGE
2 RED	10 PINK
3 CYAN (BLUE-GREEN)	11 LT. CYAN
4 PURPLE	12 LT. PURPLE
5 GREEN	13 LT. GREEN
6 BLUE	14 LT. BLUE
7 YELLOW	15 LT. YELLOW

The screen can have any of these 16 colors.

The border can only have the colors from 0 through 7.

(Note that these numbers are one less than the numbers shown on the color keys.)

You can change the printing color, even while the above program is running, by pressing the CTRL key and a color key! Try it!

CHARACTER COLOR

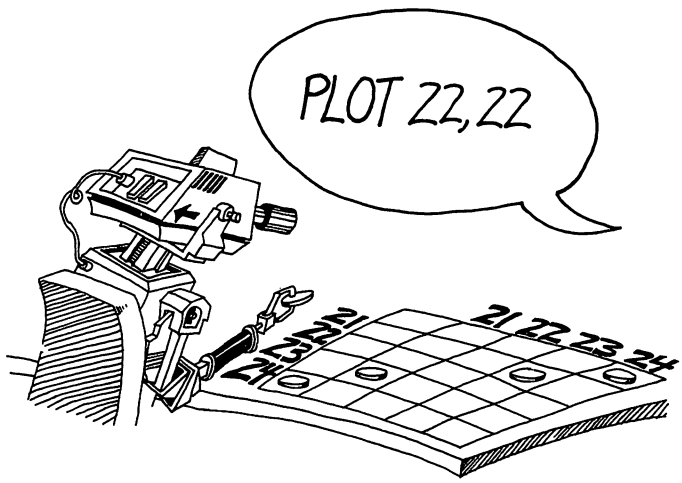
There is a box that tells the computer what color you have chosen for printing characters. Its box number is 646.

You can change the color of the letters that are printed while the program is running by poking a number from 0 through 7 in that box.

Add these lines to the above program:

```
60 PRINT"WHAT COLOR LETTERS? <0 TO 7>"
62 INPUT CL
64 POKE 646,CL
```

Run it.



Careful! If you choose the same color for the screen and letters, you cannot see the printing! Don't give up. Just enter different numbers to the blank screen anyway, the printing will appear again in some color or another!

JUMPING NAME

```
10 REM JUMPING NAME
15 PRINT "clr dn dn dn"
20 CB=646                    :REM CHAR. COLOR BOX
30 INPUT "NAME";N$
32 PRINT "HOW MANY LETTERS IN IT?"
34 INPUT N
39 PRINT "clr";
40 FOR I=1 TO RND(1)*22      :REM MOVE DOWN
45 PRINT "dn";:NEXT I
50 FOR I=1 TO RND(1)*22-N    :REM MOVE ACROSS
55 PRINT "rt";:NEXT I
57 NC=INT(RND(1)*8)          :REM RANDOM COLOR
58 IF NC=1 THEN GOTO 57      :REM NOT WHITE
60 POKE CB,NC                :REM PUT IN BOX
70 PRINT N$;                 :REM PRINT NAME
80 FOR T=1 TO 1000:NEXT T
90 GOTO 39
```

What happens: Line 39 clears the screen and homes the cursor.

Lines 40 and 45 run the cursor down the screen a random number of lines.

Lines 50 and 55 run the cursor across the screen a random number of spaces.

Line 57 picks a color for printing letters.

Line 58 picks another color if the first choice was "white." We do not want white letters on a white screen.

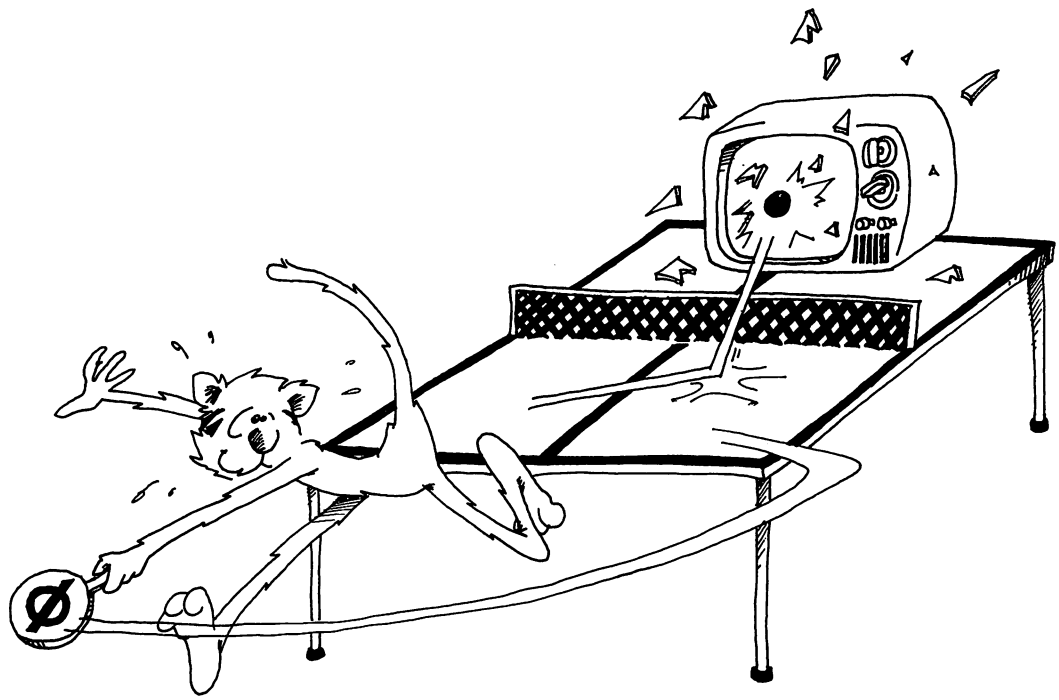
Line 60 puts the color choice in box 646. This tells the computer what color to print your name.

Line 70 prints your name in the new color.

Line 90 says: "Go do it over again."

Assignment 21:

1. Add lines to the "JUMPING NAME" program so that a new screen background color is chosen for each time through. Line 58 must compare the background color and the printing color. It must change the printing color if they are the same.
2. Write a program that uses two nested loops to go through all possible screen and border colors in order.
3. Add lines to the "BIRD" program so that the user is asked what color bird she wants. Then poke the color into the box number 646.



INSTRUCTOR NOTES 22 POKING GRAPHICS

The POKEing of characters and colors to the screen is explained.

The character is poked to one address and its color to an entirely different address. This complication may confuse your student.

The screen cells are addressed from 7680 continuously to 8185, and the color cells from 38400 to 38905. A row, column method of addressing is clearer, and a program using addressing in that form is included in the lesson.

The advantage of making graphics using POKE is that it may be clearer than the method using CRSR keys. It is also faster. The most important reason is that the PEEK instruction allows an easy way to see what character is in any given cell on the screen. This is useful for games to detect collisions, etc. This procedure is taught in lesson 28.

QUESTIONS:

1. The "home" spot on the screen has address 7680. Where on the screen is the character at address 7683 shown?
2. What address is the color box for this character?
3. What color numbers may be put in this box?

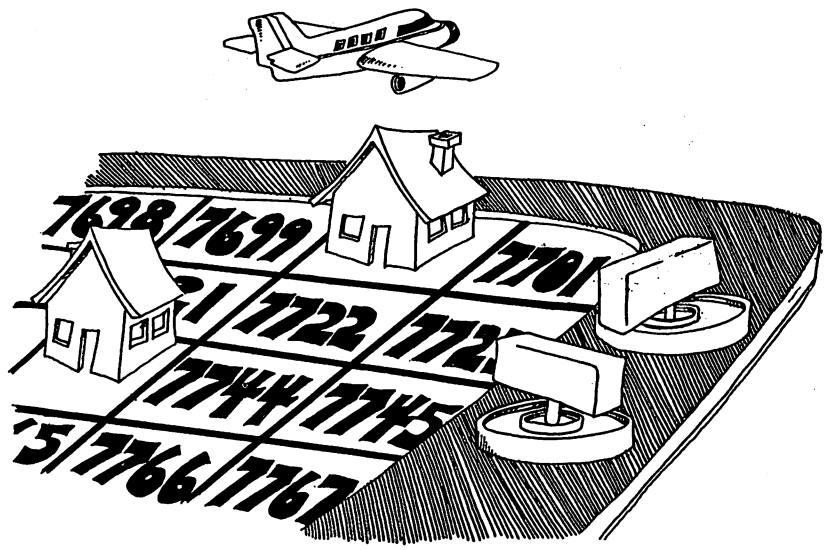
LESSON 22 POKING GRAPHICS

Make colored stripes and adjust the TV or monitor color knobs.

THE SCREEN IS A MAP OF MEMORY BOXES

Imagine a little town with 23 streets. Each street has 22 houses on it. Each house is a box in memory and can hold one character. The streets are the 23 lines, counting down the screen. There are 22 boxes counting across a line.

The computer flies over the town like an airplane. It looks down on the boxes and draws a map which it shows on the screen. The map shows which character is in each house.



Remember, most of the map is blank screen, but "blank" or "space" is a character too!

Run this:

```
10 REM POKE A HOUSE
20 POKE  7680,102          :REM
CHAR.102 INTO HOME
30 POKE 38400, 7           :REM
YELLOW COLOR
```

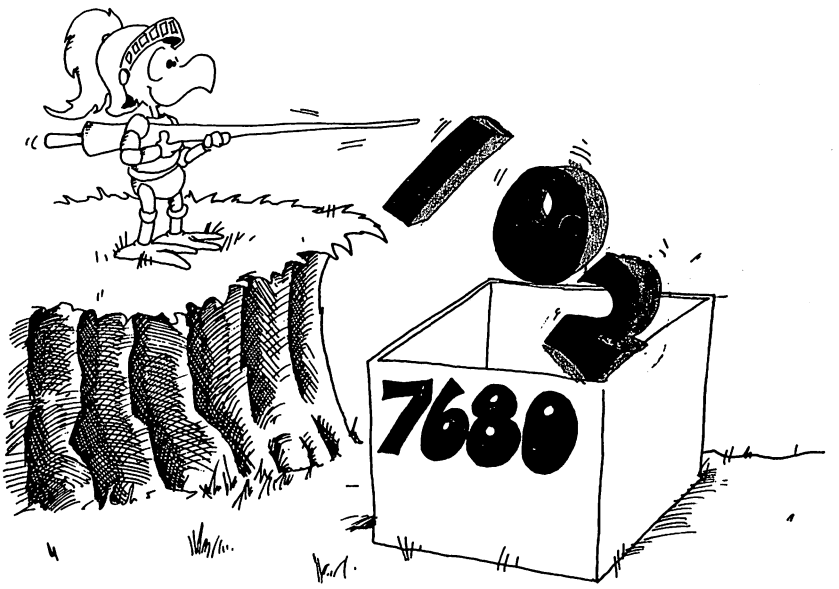
You will see a yellow square in the "home" position on the screen, (upper left corner).

WHAT HAPPENED? We POKEd the checkered box (character number 102) into the memory box that shows in the home position on the screen.

POKE means you put the number directly in the memory box.

Then we POKEd a color, yellow, into another memory box that gives the color of the character in the home position.

Try this: Change the number 102 to any other number from 64 through 127 and run it again. You will get another character in the home position. Change to another color number from 2 to 7.



ADDRESSES ON THE SCREEN

Each house has an address.

The box we call the "home" of the cursor has the lowest address. It is 7680. The label on the front of the box is 7680.

The next box to the right is 7681, and the numbers increase across the screen and down.

The highest number belongs in the lower right corner, address 8185.

THE WHOLE TOWN

Enter:

```
10 REM EVERY 3rd HOUSE
15 PRINT"clr";                    :REM
CLEAR SCREEN
20 POKE 36879,106          :REM BLUE
SCREEN
25 FOR I=7680 TO 8185 STEP 3   :REM
WALK ALONG SCREEN
30 CH=INT(RND(1)*64)+64        :REM
RANDOM CHARACTERS
60 POKE I,CH                   :REM
POKE CHARACTER
70 FOR T=1 TO 100:NEXT T       :REM
PAUSE
80 NEXT I                      :REM
END OF LOOP
```

Save to tape before running this program.

This time all the characters were white. White is the color that is usually in "color map" boxes 38400 to 38905.

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

Add this to the above program.

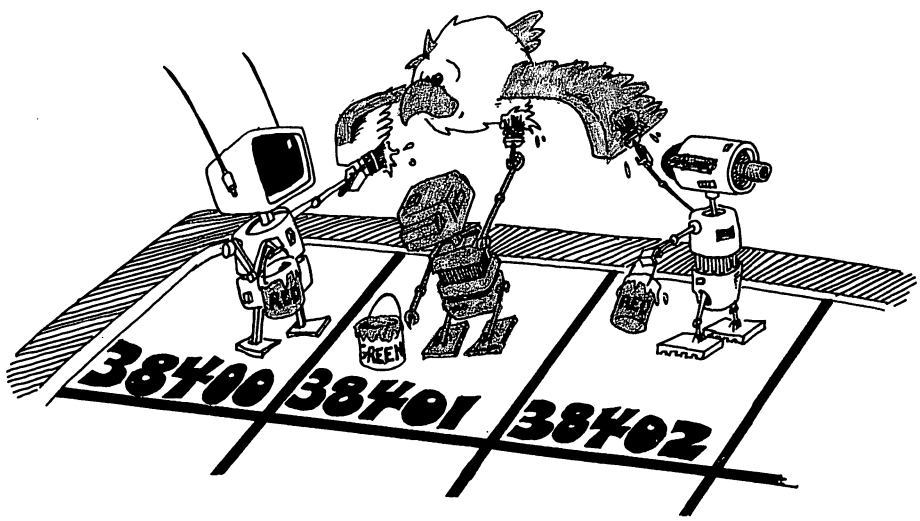
```
85 FOR I=38400 TO 38905 STEP 3   :REM COLOR BOXES
90 POKE I,RND(1)*8               :REM RANDOM COLORS
95 NEXT I
```

Try these: Change each STEP to 2. Change the screen and border number. (Look in the VIC manual to see what numbers give nice color combinations.) Add a delay loop at line 92.

Change line 30 to :

```
30 CH=INT(RND(1)*64)
```

and you will get just letters and punctuation on the screen.



COORDINATE NUMBERS

Numbers like 7680 and 38400 are confusing and hard to remember.

We can number the streets and houses like those in real towns:

Street 0 at the top of the screen
Street 1 next below
. . .
Street 22 at the bottom of the screen

We number the houses on each street:

House 0 on the left of each street
House 1 next to it

House 21 at the right end of the street

Then we use a little formula to get the address of each house and its color box.

screen address = 7680 + street number * 22 + house number
color address = 38400 + street number * 22 + house number

Enter:

```
10 REM TOWN MAP
15 SC= 7680              :REM SCREEN
CORNER
16 CC=38400               :REM COLOR
CORNER
17 B$="hm sp sp ... sp" (16 spaces)
20 PRINT"clr";
30 PRINT"hm WHICH STREET";
35 INPUT S               :REM STREET
NUMBER
38 PRINT B$;             :REM ERASE
WRITING
40 PRINT"hm WHICH HOUSE";
42 INPUT H               :REM HOUSE
NUMBER
43 PRINT B$;
45 C=5                   :REM COLOR
NUMBER 5
50 AD = 22*S + H         :REM ADDRESS
55 POKE AD+SC,102        :REM POKE
SCREEN CELL
56 POKE AD+CC,C          :REM POKE
COLOR CELL
57 FOR T=1 TO 2000:NEXT T
80 GOTO 30
```

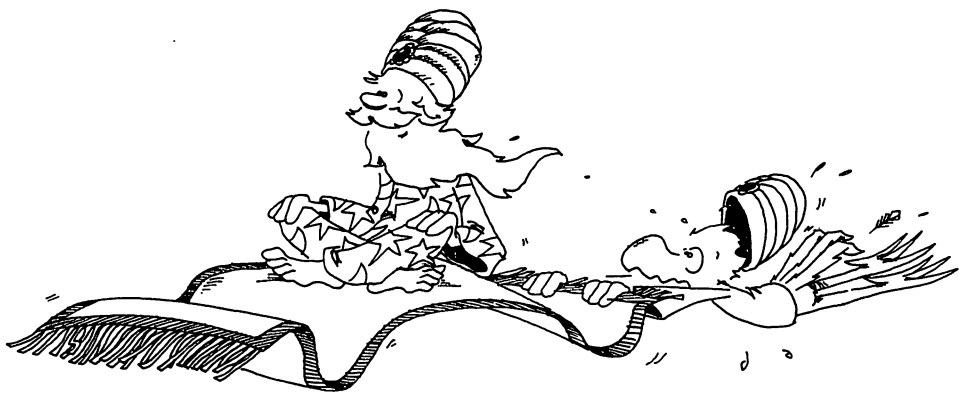
(Omit the REM statements when you type this in.)

Save to tape before you run this. (If you make a mistake in what you POKE, the computer may "crash").

Assignment 22:

1. Write a program that draws a square. Let the user choose what color it will be. Save it to tape.
2. Add to the program so that it has 1 to 6 spots on it like dice.
3. SINBAD'S MAGIC CARPET program is shown below. You prove it.

Changing the formula in line 215 gives different carpets. Add a "super" outside loop that draws one carpet after another, each changed a little in color design. Do this by putting variables in line 215 that are changed in the super loop.



INSTRUCTOR NOTES 23 SECRET WRITING AND THE GET COMMAND

This lesson concerns the GET command.

GET is a method of requesting a single character from the keyboard.

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

Another advantage over INPUT is that no RETURN key pressing is required. This makes GET useful in "user friendly" programming for menus and in games for moves, etc.

When the GET command is executed, the computer looks in the input buffer for a character. If it finds one, it returns it to the variable. (If the variable is numerical and the keystroke was not, a ?SYNTAX ERROR is printed.)

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

For most situations where you want to think, then press a key, you must put GET in a loop, and keep looking at the buffer until a non-empty string is present.

The input buffer has a "queue" up to 10 characters long. So in some cases, you can "type ahead" up to 10 characters.

QUESTIONS:

1. Compare INPUT and GET. For each question reply "GET" or "INPUT."
   a) Gets whole words and sentences at once.
   b) Shows a cursor.
   c) Gets one character at a time.
   d) Shows the keystrokes on the screen.
   e) Does not need a RETURN keypress.
2. Why do you usually need to put GET in a short loop?

LESSON 23 SECRET WRITING AND THE GET COMMAND

THE INPUT COMMAND

There are two ways to use INPUT.

Without a message:

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

With a message:

```
10 INPUT "NAME,AGE";NAME$,AGE
10 INPUT "HOW ARE YOU";FEEL$
```

Either way, the computer waits for you to type in a word, sentence or number, and for you to press the RETURN key.

THE GET COMMAND AND SECRET WRITING

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

```
10 GET A$
```

When the program reaches GET in line 10, the computer looks to see if any key has been pressed. If so, it reads the key and puts the character into memory box A$.

Nothing shows on the screen:

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

THE COMPUTER IS IMPATIENT

The computer does not wait. If you have not pressed a key by the time the computer reaches GET, it fills the variable A$ with "" (the empty string) and goes on!

This is a bad feature if you want to think before you press any key. The computer does not wait for you to think.

MAKING THE COMPUTER WAIT

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

```
35 GET L$:IF L$="" THEN GOTO 35
```

This is a good way to use GET in guessing games for entering the word or number to be guessed without the other player being able to see it.

Run this program:

```
10 REM -----GET-----
20 PRINT "clr dn dn dn"
30 PRINT "PRESS ANY KEY"
40 GET K$:IF K$=""THEN 40
45 PRINT "dn dn red WAITING"
47 FOR T=1 TO 1000:NEXT T
50 PRINT "blk THE KEY YOU PRESSED
WAS ";K$
55 FOR T=1 TO 1000:NEXT T
60 GOTO 20
```

MAKING WORDS OUT OF LETTERS

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

```
10 REM ## BACKWARDS ##
20 PRINT "clr dn dn dn"
25 W$ = ""
30 PRINT "A 5 LETTER WORD dn"
40 FOR I = 1 TO 5
42 GET L$:IF L$="" THEN GOTO 42
44 W$ = L$+W$
46 NEXT I
47 FOR T=1 TO 1000: NEXT T
50 PRINT "clr dn BACKWARDS dn"
60 PRINT W$
```

This program makes a word of any length you want.

```
10 REM GET A WORD
20 PRINT "clr dn dn dn"
30 PRINT"TYPE A WORD dn"
31 PRINT"END IT WITH A PERIOD"
```

```
35 W$=""          :REM WORD STRING IS
EMPTY
40 GET L$:IF L$=""THEN GOTO 40
50 IF L$="." THEN GOTO 80:REM TO
TEST FOR END
60 W$=W$+L$       :REM ADD LETTER TO
END OF WORD
65 GOTO 40        :REM TO GET ANOTHER
LETTER
80 REM WORD IS FINISHED
85 PRINT "clr dn pur";W$;"blk"
```

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

THE GET COMMAND FOR NUMBERS

The GET command can be used to input numbers. If no key has been pressed, the number will be zero. Try this:

```
10 GET N
15 PRINT N
20 GOTO 10
```

Assignment 23:

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

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

2. Write a sentence making game. Each sentence has a noun subject, a verb, and an object. The first player types a noun (like "The donkey"). The second player types a verb (like "sings"). The third player types another noun (like "the toothpick."). Use GET so no player can see the words of the others. You may expand the game by having adjectives before the nouns.

INSTRUCTOR NOTES 24 PRETTY PROGRAMS, GOSUB, RETURN, END

This lesson covers subroutines. The END command is also treated here because the program will usually have its subroutine at high line numbers and so must END in the middle line numbers.

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

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

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

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

QUESTIONS

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

LESSON 24 PRETTY PROGRAMS, GOSUB, RETURN, END

Run this program then save it to tape:

```
100 REM MAIN PROGRAM
101 :
110 PRINT "clr READY TO GO dn"
120 GOSUB 200
130 PRINT "grn BACK FROM THE SUBROUTINE dn"
135 PRINT "yel BACK AGAIN dn blk"
190 END
199:
200 REM SUBROUTINE
201 :
210 PRINT "red      IN THE SUBROUTINE dn"
215 FOR T=1 TO 1000 :NEXT T
290 RETURN
```

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

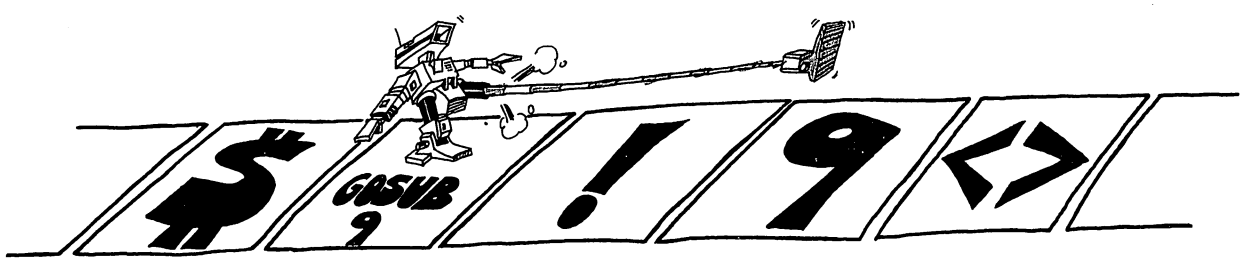
Where there are PRINT commands, you may put many more program lines.

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

Line 120 and line 140 "call the subroutine." This means the computer goes and does the commands in the subroutine, then comes back.

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

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

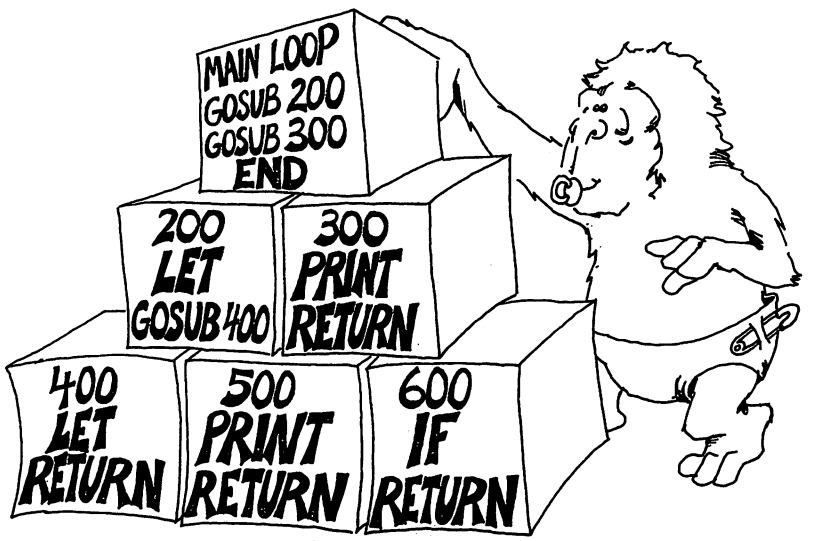


WHAT GOOD IS A SUBROUTINE?

In a short program, not much good.

In a long program, it does two things:

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



THE END COMMAND

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

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

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

MOVING PICTURES

```
10 REM  JUMPING J
15 PRINT "red clr'
20 X=10:Y=10:D=1
25 FOR J=1 TO 6
26 FOR I=1 TO 6
29 A$="rev sp off"
30 GOSUB 100:REM DRAW
34 A$=" "
35 GOSUB 100:REM ERASE
40 Y = Y - D
50 NEXT I
55 D = -D
60 NEXT J
90 END
100 REM
101 REM DRAW THE J
102 REM
104 PRINT"hm"
105 FOR L = 1 TO Y:PRINT:NEXT
109 FOR K = 1 TO 5
110 PRINT TAB(X);A$"    "A$
111 NEXT K
120 PRINT TAB(X);A$;" ";A$
130 PRINT TAB(X);" ";A$
190 RETURN
```

The picture is the letter "J." The subroutine starting in line 100 draws the "J." Before you GOSUB 100 you pick what character you want the "J" to be by setting A$. Look at line 29 and at line 34. If you pick A$ to be " ", a single space, then the subroutine erases a "J" from that spot.

The subroutine draws the "J" with its upper-left corner at the spot X,Y on the screen. When you change X or Y (or both) the 'J' will be drawn in a different spot. Line 20 says that the first "J" will be drawn near the middle of the screen.

The letter "D" tells how far the "J" will move from one drawing to the next. Line 20 makes "D" equal to 1, but line 55 changes D to -1 after 6 pictures have been drawn.

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

Assignment 24A:

1. Enter the JUMPING J program and run it. Then make these changes:
   Change the subroutine so it prints your own initial.
   Change the color of your initial to blue.
   Change the "jumping" to "sliding" (so the J moves horizontally instead of vertically).
   Change the starting point to the lower right-hand corner instead of the middle of the screen.
   Change the distance the slide goes to 8 steps instead of 6.
   Change the size of each step from 1 to 2.
   Change the "sliding" so it slides uphill. Use

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

   Change the program so the letter changes color from red (color 2) through all the colors to yellow (color 7) as it jumps.

HOW TO WRITE A LONG PROGRAM

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

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

The outline of could be:

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

After making this outline, I filled in more details.

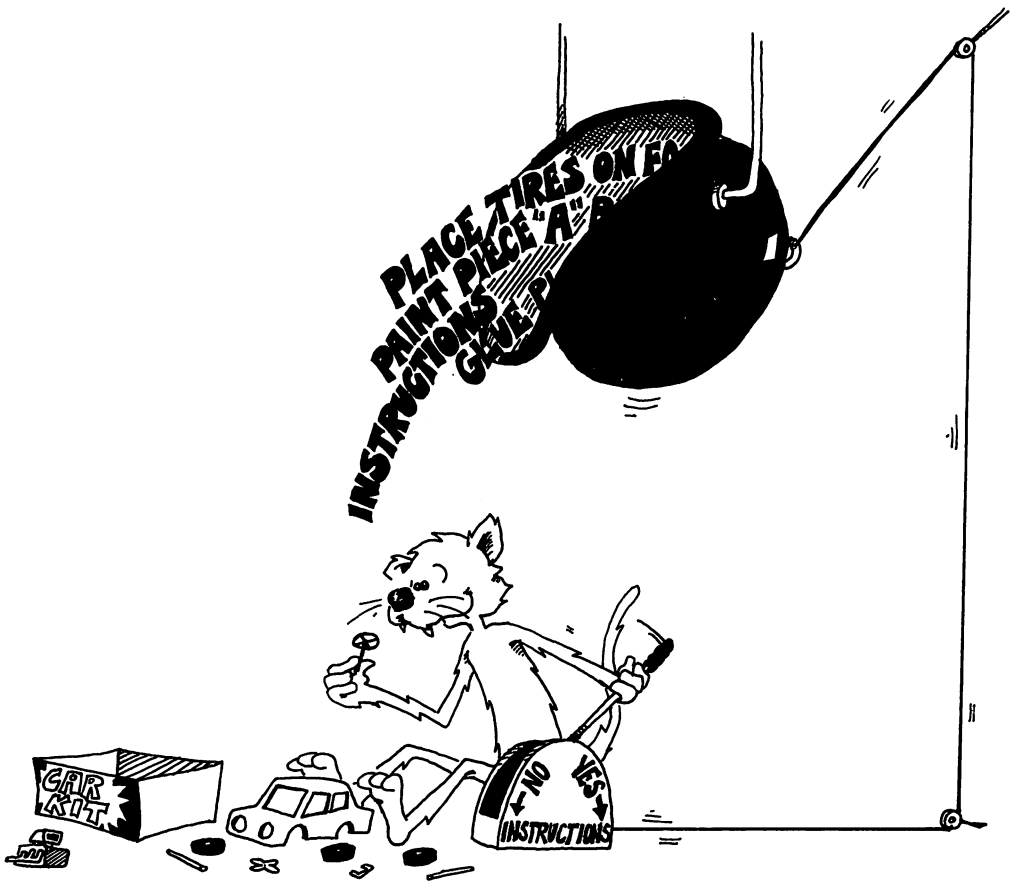
```
10 REM *** HANGMAN GAME ***
99 :
100 REM MAIN LOOP
101:
120 INPUT" NEED INSTRUCTIONS? <Y_N> "; Y$
122 IF Y$="Y" THEN GOSUB 200
130 GOSUB 300:REM GET WORD
132 STOP
135 GOSUB 400:REM MAKE GUESS
140 GOSUB 500:REM TEST GUESS
145 GOSUB 700:REM TEST IF GAME IS OVER
190 GOTO 135:REM MAKE ANOTHER GUESS
199:
```

```
200 REM INSTRUCTIONS
...  write the instructions last
290 RETURN
299:
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:
400 REM MAKE A GUESS
...  player 2 guesses a letter
490 RETURN
499:
500 REM TEST IF GUESS IS RIGHT
...  if wrong, GOSUB 600:REM draw hangman part
...  if right, GOSUB 700:REM see if game is over
590 RETURN
599:
600 REM ADD TO THE DRAWING
...  add to the hangman drawing
...  test if drawing is done
...  if so, then GOSUB 800
690 RETURN
699:
700 REM TEST IF GAME IS OVER
...  see if all letters have been guessed
...  if yes, GOSUB 900
790 RETURN
799:
800 REM END GAME MESSAGE
...  message for when guesser loses
890 RETURN
899:
900 REM END OF GAME MESSAGE
...  message for when guesser wins
990 RETURN
```

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

Assignment 24B:

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



ADVANCED PROGRAMMING

INSTRUCTOR NOTES 25 DATA, READ, RESTORE

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

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

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

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

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

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

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

QUESTIONS:

1. What happens if you try to READ more data items than are in the DATA statements?
2. What rule tells you where to put the DATA statements in the program? How about where to put the READ statements?
3. Can you put numerical data and string data in the same DATA statement?
4. Can you change the items in a DATA statement while the program runs?

LESSON 25 DATA, READ, RESTORE

TWO KINDS OF DATA

There are two kinds of data in your programs:

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

```
10 REM FIRST KIND OF DATA
20 PRINT"clr dn dn dn"
30 INPUT"YOUR PET PEEVE";P$
40 PRINT"dn REALLY!"
50 PRINT"dn red YOU DON'T LIKE dn grn"
60 PRINT P$;"blk"
```

   In this program, P$ is data entered by the user as the program runs.
2. The kind that is stored in the program at the time it is written.

```
10 REM THE SECOND KIND
20 PRINT"clr dn dn dn"
30 X=2
40 Y=3
50 PRINT X+Y
```

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

STORING LOTS OF DATA

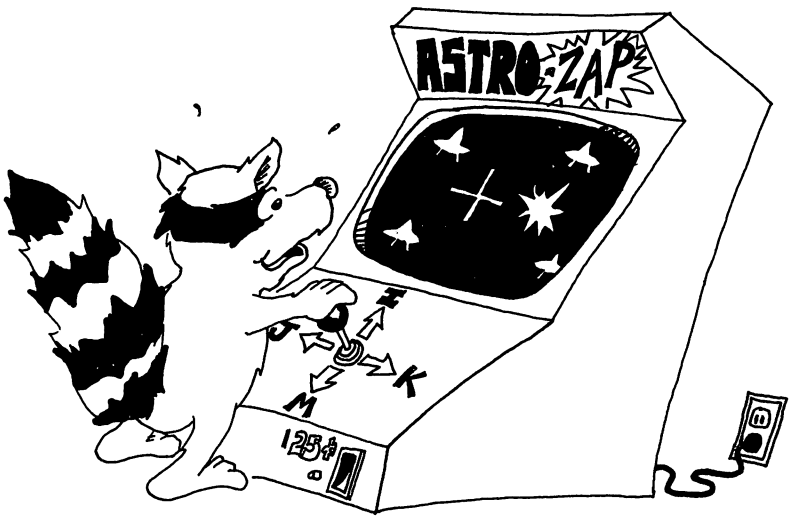
It is OK to store small amounts of data in LET statements. But it is awkward to store large amounts of data that way.

Use the DATA statement to store large amounts of data.

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

```
10 REM LOTS OF DATA
20 PRINT"clr dn dn dn"
30 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,
FRIDAY,SATURDAY
40 READ D1$
42 READ D2$
44 READ D3$
46 READ D4$
60 PRINT D1$,D2$
```

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



STRANGE RULES

1. It doesn't matter where the DATA statement is in the program.
   Do this: Change line number 30 in the above program to line number 90. Run the program. It works just the same.
2. It doesn't matter how many DATA statements there are.
   Do this: Break the DATA statement into two statments:

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

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

IT IS POLITE FOR THE "READ" COMMAND TO POINT

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

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

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

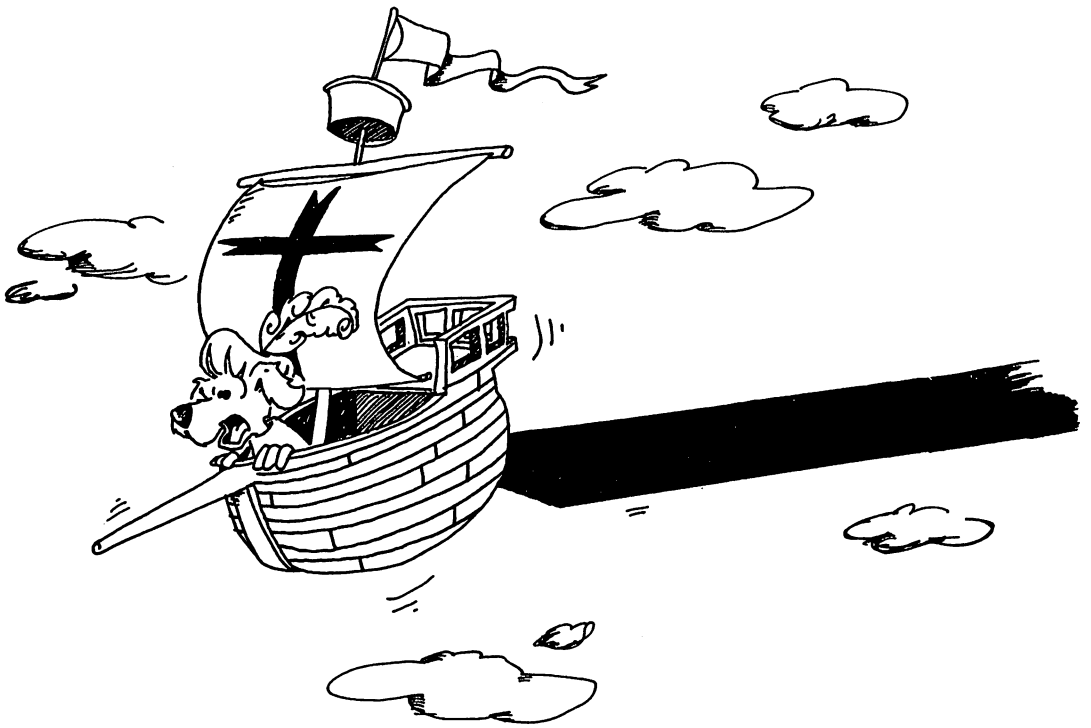
If the pointer gets to the end of one DATA statement, it automatically goes to the next DATA statement (that is, to the DATA statement with the next higher line number).

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

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

```
30 DATA SUNDAY,MONDAY,TUESDAY
. . .
91 DATA WEDNESDAY,THURDSAY,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
```

BACK TO SQUARE ONE

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

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

MIXTURES OF DATA

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

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

Correct:

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

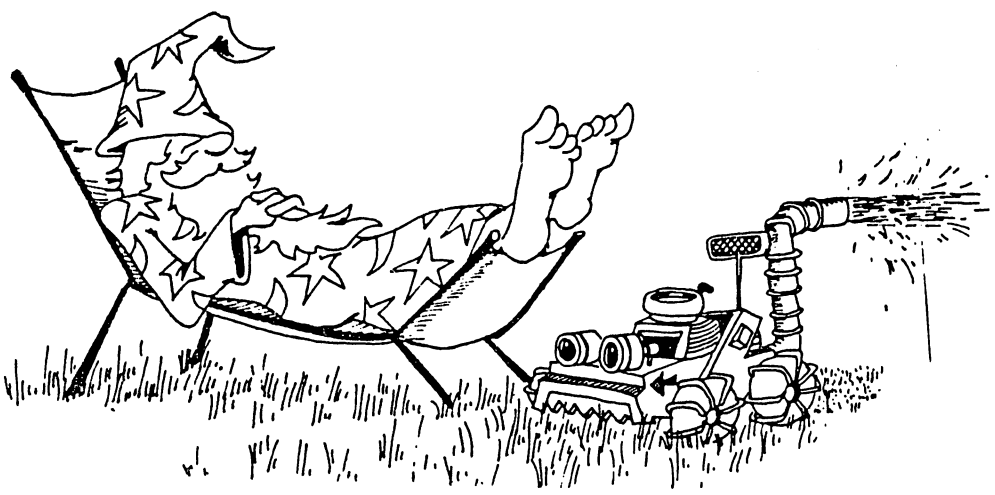
Wrong:

```
70 DATA 77,FUZZ
75 READ B$              ok, B$ box holds
"77"80 READ N               TYPE MISMATCH
ERROR
```

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

Assignment 25:

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



INSTRUCTOR NOTES 26 SNIPPING STRINGS: LEFT$, MID$, RIGHT$, and LEN

In this lesson the functions:

```
LEFT$     MID$     RIGHT$
LEN
```

are demonstrated. The use of MID$() with 3 arguments is shown, but not that with the third argument omitted.

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

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

QUESTIONS:

1. If you want to save the "STAR" from "STARS AND STRIPES," what function will you use? What arguments?
2. If you want to save "AND", what function and arguments?
3. If you want to count the number of characters in the string PQ$, what function do you use? What argument?
4. What is wrong with each of these lines?

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

5. What two arguments does the RIGHT$() function need?
6. What command will snip the third and fourth letters out of a word?
7. Write a short program that takes the word "computer" and makes it into "putercom."

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

GLUING STRINGS

You already know how to glue strings together:

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

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

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

SNIPPING STRINGS

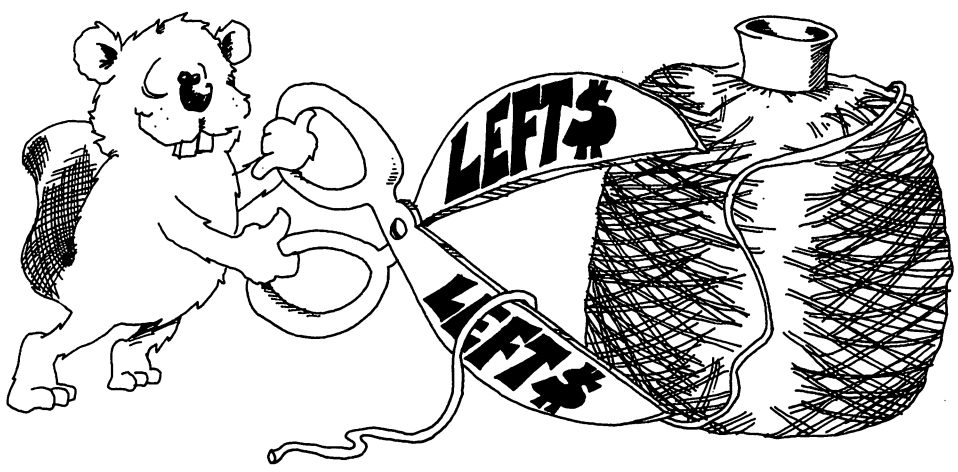
Let's cut a piece off a string. Enter and run:

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

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

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

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



Try another. Change line 40 to:

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

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

```
789
```

RIGHT$() is like LEFT$() except the characters are saved off the right end of the string. (But the order of letters is still "reading order", left to right.)

MORE SNIPPING AND GLUING

Run:

```
10 REM SCISSORS & GLUE
20 PRINT"clr"
30 N$="123456789ABCDEF"
35 FOR I=1 TO 9
40 L$=LEFT$(N$,I) : R$=RIGHT$(N$,I)
50 PRINT I;L$;" ";R$
60 NEXT I
```

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

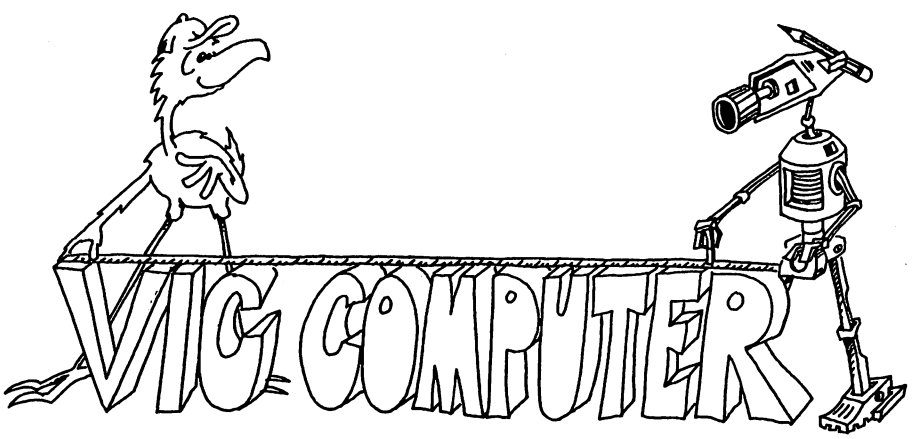
```
55 IF I=4 THEN PRINT: PRINT R$ + L$ :PRINT
```

HOW LONG IS THE STRING?

Run:

```
10 REM : LONG ROPE :
20 PRINT"clr"
30 INPUT"GIVE ME A STRING: ";N$
35 PRINT"clr"
40 L=LEN(N$)
50 PRINT "THE STRING: dn"
52 PRINT "'";N$;"' dn"
55 PRINT "IS";L;"CHARACTERS LONG"
```

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



CUTTING A PIECE OUT OF THE MIDDLE

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

Run:

```
10 REM ### MIDDLE ###
20 PRINT"clr"
30 N$="123456789"
40 P$=MID$(N$,3,4)
50 PRINT P$
```

The line:

```
40 P$= MID$(N$,3,4)
```

LOOK MA, NO SPACES

Enter:

```
10 REM > NO SPACES >
20 PRINT"clr dn dn GIVE ME dn"
21 PRINT"A LONG SENTENCE dn"
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$: REM
SAVE ONLY LETTERS
80 NEXT I
90 PRINT"clr dn dn";T$;"dn dn dn"
```

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

Assignment 26:

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

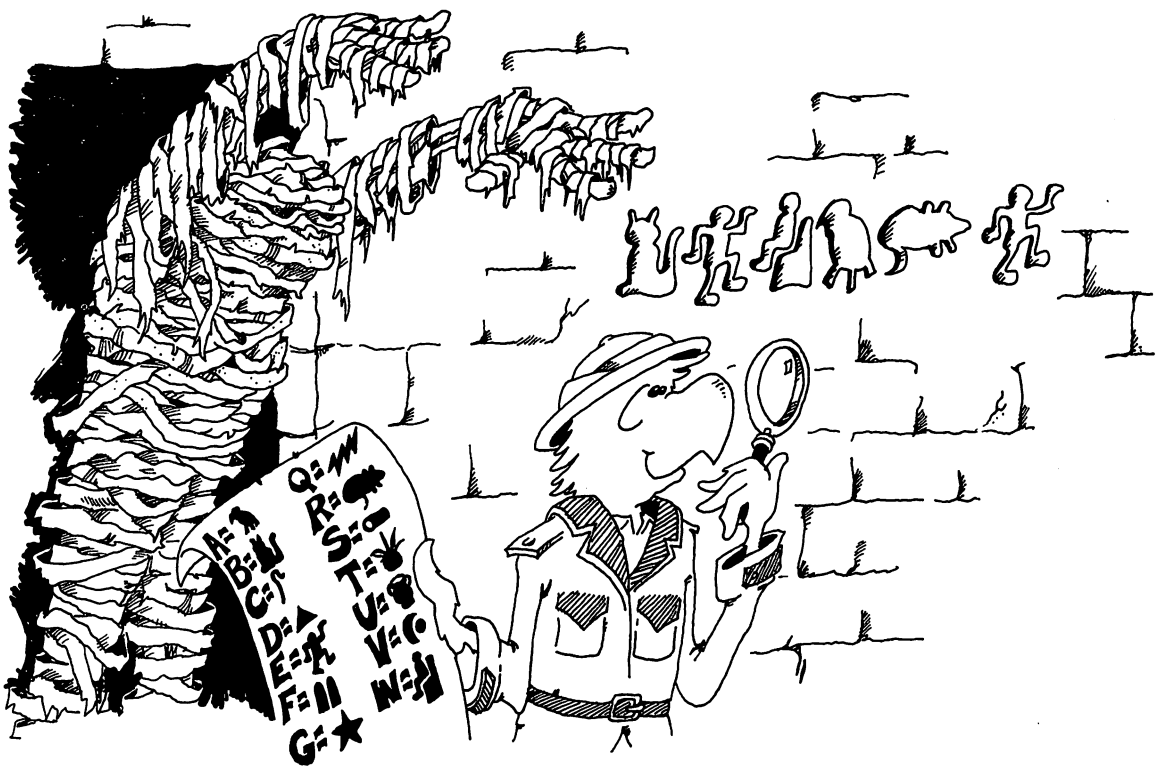
```
THIS IS A TEST.    becomes:
HTSII S AETTS .
```

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

```
ARE YOU A TURKEY?
YOU ARE A TURKEY.
```

3. Write a PIG LATIN program. It asks for a word. Then it looks for the first vowel in the word. It takes all the letters up to the vowel and puts them on the end of the word, and adds "AY." (If the word starts with a vowel, the program just adds "LAY" to the end of the word. Examples:

```
TURKEY    becomes   URKEYTAY
ADAM      becomes   ADAMLAY
```



INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS

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

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

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

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

QUESTIONS:

1. If your number "marches" too quickly in program 2 of the assignment, how do you slow it down?
2. If your program has the string "GEORGE WASHINGTON WAS BORN IN 1732." write a few lines to answer the question "How long ago was Washington born? (You need to get the birthdate out of the string and convert it to a number.)
3. What is a "value." What is meant by "a function returns a value?" What are some of the things you can do with the value?
4. What is an "argument" of a function? How many arguments does the MID$() function have? How many for the CHR$() function?
5. Where in the line do commands always go? Can you put a function at the start of a line?

LESSON 27 SWITCHING NUMBERS WITH STRINGS

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

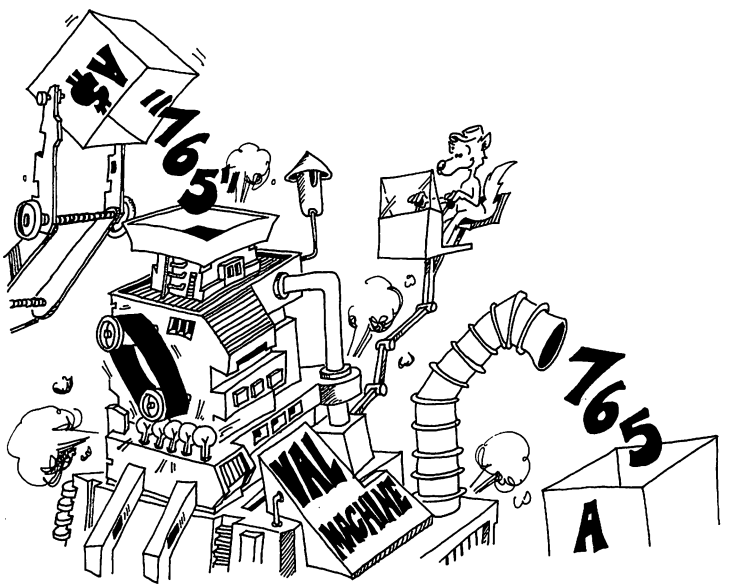
MAKING STRINGS INTO NUMBERS

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

Run:

```
10 REM STRINGS INTO NUMBERS
20 PRINT"clr"
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 a string to a number, if it can.



MAKING NUMBERS INTO STRINGS

Run:

```
10 REM NUMBERS INTO STRINGS
20 PRINT "clr"
25 INPUT"GIVE ME A NUMBER ";NB
30 N$=STR$(NB)
35 L=L EN(N$)
40 FOR I=L TO 1 STEP -1
```

```
45 B$=B$+MID$(N$,I,1)
50 NEXT I
60 PRINT"red HERE IT IS BACKWARDS dn
blk"
65 PRINT B$
```

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

FUNCTIONS AGAIN

In this book we use these functions:

```
RND(), INT(), LEFT\(), RIGHT\(), MID\(), LEN(),
VAL(), STR\(), ASC(), CHR\(), PEEK()
```

Rules about functions:

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

```
MID$(D$,5,J) has 3 arguments: D$, 5, and J
```

The arguments may be numbers or strings or some of each.

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

```
right:          10 LET D=LEN$(CS$)
wrong:          10 LEN(CS$)=5
```

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

The arguments tell which value is returned.

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

PRACTICE WITH FUNCTIONS

For each function in the list below:

Tell how many arguments it has, and give their names. Tell whether the value of the function is a string or a number.

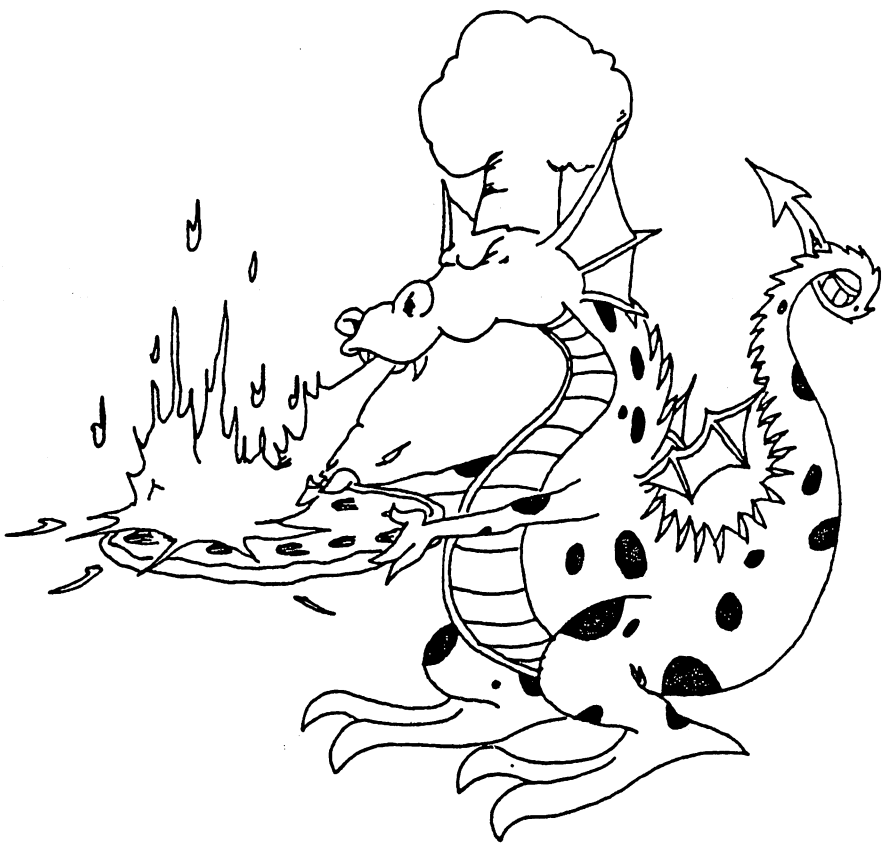
```
RND(8) _____
INT(Q) _____
LEFT$(U$,Y) _____
MID\(RIGHT$,E,2) _____
VAL(ER$) _____
STR$(INT(RND(8))) _____
```

Each line below has errors. Explain what is wrong.

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

Assignment 27:

1. Write a program that asks for a number. Then make another number that is backwards from the first, and add them together. Print all three numbers like an addition problem (with "]" sign and a line under the numbers).
2. Make a number "march" slowly across the screen. That is, write it on the screen, then take its left digit and move it to the right. Keep repeating. Don't forget to erase each digit when you move it.



INSTRUCTOR NOTES 28 ACTION GAMES AND THE FUNCTION KEYS

This long lesson introduces the function keys and the "keyboard's box." It shows how to control two simultaneously moving objects on the screen, and how to use PEEK to detect collisions.

We develop the CUPID game in this lesson, one piece at a time.

It serves as a prototype for many different games. This game uses an elegant method of moving the arrow shot by a "diamond" while the diamond can still move and be controlled from the keyboard. The student may profit from help in understanding how the movements are achieved.

When drawing moving objects, you need to erase each old image before the next image is drawn.

A "hit" on a target is detected by using PEEK to test the square in front of the projectile. If there are several kinds of targets, it is better to test if the square is "background." If not, then jump to a subroutine that asks the ASCII number of the target type so appropriate action can be taken.

Graphics games may grow to be rather long. BASIC is a little slow for such games. Maximum speed can be obtained if the "working" part of the program is first, and the "initialization" part is at the end, reached by a call from early in the program. This format is discussed in lesson 32.

For maximum speed, avoid repeatedly converting numerical constants to floating point as the program runs. That is why lines 55, 60, and 65 compare K$ to variables F1, F2, and Z0 rather than to 39, 47, and 48. This practice is probably the most important single factor in obtaining fast programs.

QUESTIONS:

1. What ASCII numbers do the function keys have?
2. How can you detect that a function key has been pressed?
3. How can you detect that a key is being held down?
4. What number box holds the key being pressed?
5. What number is in the box if no key is being pressed?

LESSON 28 ACTION GAMES AND THE FUNCTION KEYS

THE FUNCTION KEYS

The 4 brown keys on the right side of the VIC are called "function keys."

They have these ASCII numbers:

f1	133		
f2		137	(shifted f1)
f3	134		
f4		138	(shifted f3)
f5	135		
f6		139	(shifted f5)
f7	136		
f8		140	(shifted f7)

How can you find out if a function key is being pressed?

You cannot use the INPUT command. It ignores the function keys. You use the GET command.

Enter:

```
10 REM FUNCTION KEYS
20 F1$=CHR$(133)
30 GET K$:IF K$="" THEN 30
40 IF K$=F1$ THEN PRINT "YOU PRESSED
F1"
45 PRINT"rvs sp sp sp lf lf lf off";
50 PRINT K$
60 GOTO 30
```

(Remember, "rvs" means push CTRL and RVS ON, "sp" means push the space bar, and "off" means push CTRL and RVS OFF.

Run it. Press any key and see that it is printed to the screen. But when you press f1, the special message gets printed.

Notice that trying to print character number 133 gives nothing. The eight characters that are functions keys do not print.

USING THE FUNCTION KEYS IN PROGRAMS

First you GET a one character string from the keyboard. If it has ASCII number 133 through 140, it is one of the function keys. The little program above shows how to do this.

Make these changes in the above program:

```
40 IF K$=F1$ THEN GOSUB 200
200 REM SCREEN COLOR CHANGE
210 PRINT"SCREEN COLOR <R/Y>"
215 GET C$:IF C$="" THEN 215
220 IF C$="R" THEN POKE 36879,168
230 IF C$="Y" THEN POKE 36879,253
299 RETURN
```

Now the f1 key sends you to a subroutine to choose screen colors.

THE TROUBLE WITH "GET"

With GET, the computer can't tell if you just gave a key a quick press or are holding the key down.

Suppose you want to control a car on the screen by pushing the "s" (for "speed") key. As long as the "s" key is down, the car goes. When you let up on the "s" key, the car stops. The GET command is not good for doing this.

LOOKING IN THE KEYBOARD'S BOX

Here is how to tell if a key is being pressed.

There is a memory box that tells if a key is being pressed. Its address is 197.

The box holds the number 64 if no key is being pressed.

If a key is being pressed, it holds a number from 0 to 63 to tell which one. (If two keys are being pressed, it holds the higher number of the two.)

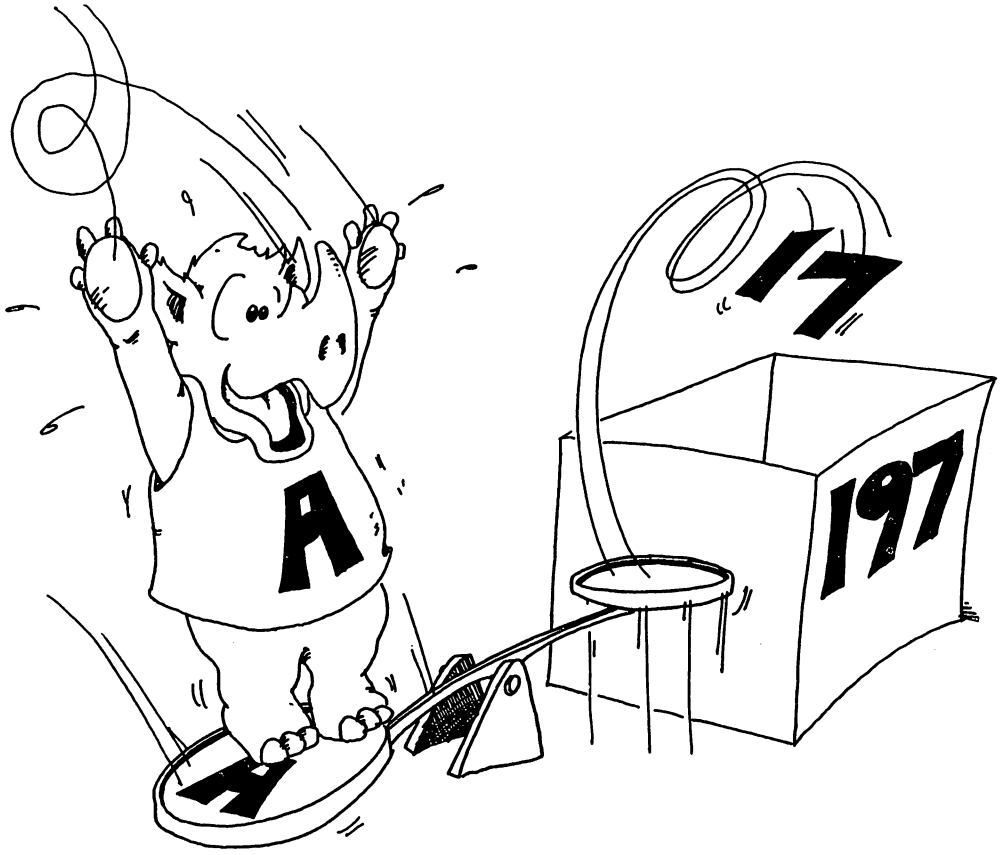
Try this:

```
10 REM KEYBOARD'S BOX
20 PRINT"clr"
30 PRINT"PRESS ANY KEY"
40 N=PEEK(197)
50 PRINT N
60 GOTO 40
```

Notice that the number keeps printing as long as you hold the key down.

Try holding two keys down at once. Compare the number to the one for holding each of the keys by itself.

The only keys that do not put numbers in the box are CTRL, RUN STOP, SHIFT, SHIFT LOCK, and the COMMODORE FLAG.



Assignment 28A:

1. Make a table showing the number in box 197 for each key on the keyboard. (Note: there is no key giving numbers 16, 25, 38, and 40.) This table will be useful when you design games.

MOVING A DIAMOND

Make the f1 and f2 keys control a moving diamond.

It moves up while you press the f1 key.

It moves down while you press the f2 key.

It stops if you don't press either key.

```
10 REM CUPID
11 PRINT" clr"
12 POKE 36879,43          :REM RED SCREEN
20 F1=39:F3=47:Z0=64      :REM KEY NUMBERS
24 SC=7700                :REM SCREEN CORNER
```

```
26 Y =SC+220                    :REM START SPOT OF DIAMOND
50 K =PEEK(197)                 :REM LOOK FOR A KEY PRESS
55 IF K=Z0 THEN D=  0           :REM DON'T MOVE?
60 IF K=F1 THEN D=-22           :REM MOVE UP?
65 IF K=F3 THEN D= 22           :REM MOVE DOWN?
90 POKE Y,32                    :REM ERASE OLD DIAMOND
120 Y=Y+D                       :REM SPOT FOR NEW ONE
130 IF Y>8184 THEN Y=8184       :REM OFF BOTTOM?
140 IF Y<7700 THEN Y=7700       :REM OFF TOP?
170 POKE Y,90                   :REM POKE NEW DIAMOND
199 GOTO 50                     :REM DO AGAIN
```

Save to tape before running.

Run it. It puts a white diamond on a red screen. The diamond runs up and down the right side of the screen. It goes when you press f1 or f3. Otherwise it stops.

Line 50 looks in the keyboard's box.

Line 55 says: "if no keys are being pressed, set D to zero." D is how much the diamond's screen address will change before the diamond is next put on the screen.

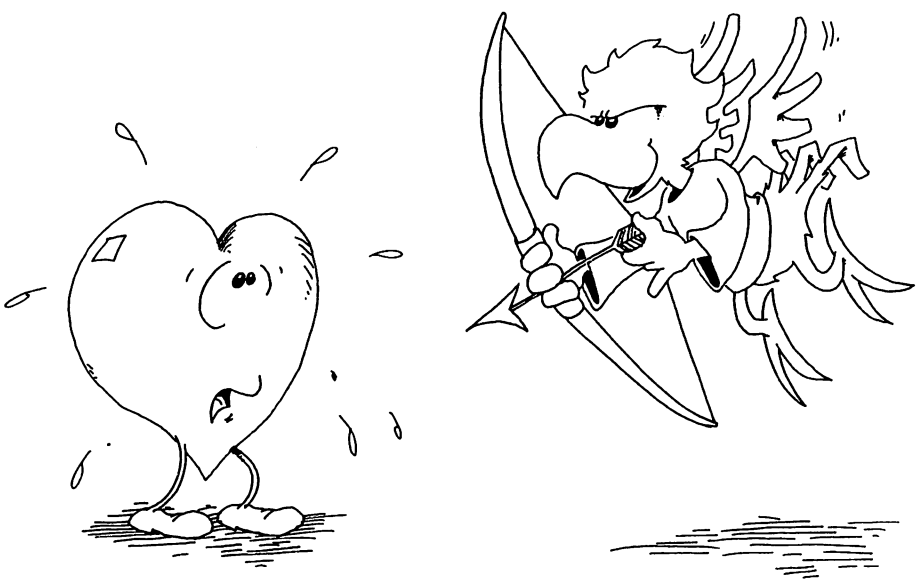
Line 60 tests to see if the f1 key is being pressed. If it is, D is set to -22. This means the next diamond will be put one line higher than the present diamond.

Line 65 sets D to 22 if f3 is being pressed. It means the diamond will be POKEd one line lower next time.

Line 90 erases the old diamond by putting a space (character 32) on its screen spot.

Lines 130 and 140 perform another bit of housekeeping. They check the value of Y (the address of the diamond on the screen) to make sure that the diamond will not try to move off the top or bottom of the screen.

Line 170 POKEs the new diamond onto the screen, and the cycle starts again.



SHOOTING ARROWS

We want the diamond to shoot arrows when we press the "Q" key.

Why pick "Q"? Two reasons:

First, "Q" is in a comfortable spot for the left hand to control shooting while the right hand controls motion.

Second, "Q" has number 58, higher than f1 or f3, so even if two keys are pressed at once, an arrow will shoot. (Be careful to not hold the "Q" key down, or else the f1 and f2 keys will not control the diamond.)

Add these lines:

```
28 S=0:X=Y
70 IF K=48 AND S=0 THEN S=S+1
75 IF S<>0 THEN GOSUB 300
300 REM SHOOT
310 IF S=1 THEN X=Y-1:POKE X,31:S=2:RETURN
315 POKE X,32
316 IF S=21 THEN S=0:RETURN
320 X=X-1:S=S+1
330 POKE X,31
399 RETURN
```

THE RIGHT WAY TO SHOOT

It's no good if the diamond has to stop moving while the arrow moves.

So when line 70 sees that the "shooting key" is pressed, it doesn't just jump to a subroutine that shoots the arrow across the screen. Instead, it sets a "flag." The flag is the variable "S." The flag is "set" when S<>0.

When the flag is set, the computer takes turns moving the arrow one space, then the diamond, then the arrow again, etc.

The "move arrow" subroutine has to recognize two things:

First, is this a new arrow, or just one that already started across?

Line 310 does this. S equals 1 only for a new arrow.

After each move of the arrow, S gets 1 bigger and X, the position of the arrow, gets 1 smaller so the arrow goes to the right.

Second, the arrow must be stopped when it gets to the other side of the screen, and the diamond must be told that it can shoot another arrow.

Line 316 does this. When S=21, the arrow has moved 21 spaces and is at the other side of the screen. S is set back to 0 meaning that the diamond is allowed to shoot again. (The arrow was erased in line 315 and no new arrow has been written so the screen is empty of arrows.)

THE TARGETS ARE HEARTS

Add these lines:

```
30 GOSUB 400
400 REM HEARTS
405 FOR L=1 TO 20
410 J=7680 + INT(RND(1)*23)*22
420 I=INT(RND(1)*15)
430 POKE I+J,83
450 NEXT L
499 RETURN
```

Save to tape and then run it.

The arrows hit the hearts and erase them.

POPPING THE TARGETS

Add this line:

```
325 IF PEEK(X)=83 THEN S=0:POKE X,91:RETURN
```

PEEK to see if the arrow is about to hit a target.

PEEK(X) tells the number of the character at the spot the arrow is about to move on to. If the spot has number 83, it is a heart. Then pop it by poking a cross there. Set S equal to zero because that arrow is finished moving.

Assignment 28B:

1. You can change the CUPID program in many ways. Change which characters are used. Change colors. Change the column which the diamond moves in.
2. Add a "laser" sound which starts when the arrow is shot and stops when the arrow ends its flight.
3. Why does the diamond dodge around while the arrow flies? It only makes sense if the hearts are shooting back at the diamond! Add missiles that try to hit the diamond.

INSTRUCTOR NOTES 29 ASCII CODE, KEYBOARD, ON...GOTO

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

The ASCII code is primarily intended to standardize signals between hardware pieces such as computers with printers, terminals, other computers, etc. But within programs the ASCII numbers also are useful. The letters are numbered in increasing order. You can alphabetize a list by putting the ASCII numbers in order. The numerical digits are also in order; the punctuation marks and graphics characters also have ASCII numbers.

Some of the signals sent between hardware, such as "bell", "line feed", and "carriage return" also have ASCII numbers which can be used inside of PRINT statements for control of the TV screen and the VIC's loud speaker.

QUESTIONS:

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

```
10 PRINT CHR$(13);
```

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

LESSON 29 ASCII CODE, KEYBOARD, ON...GOTO

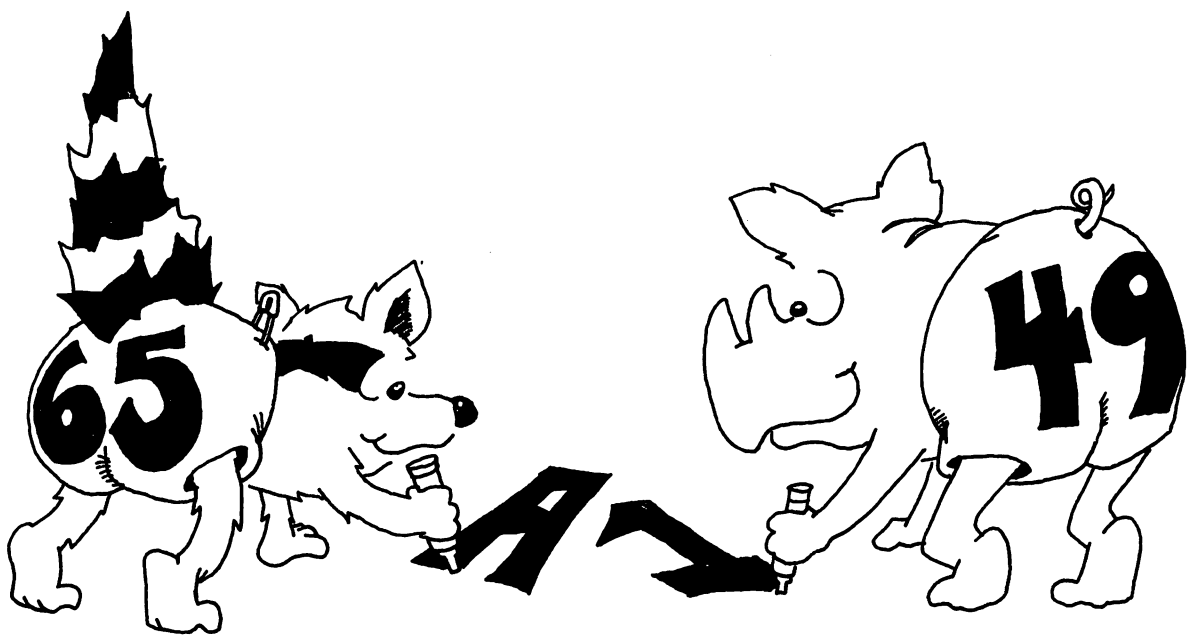
NUMBERING THE LETTERS IN THE ALPHABET

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

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

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

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



ASC() CHANGES CHARCTERS INTO NUMBERS

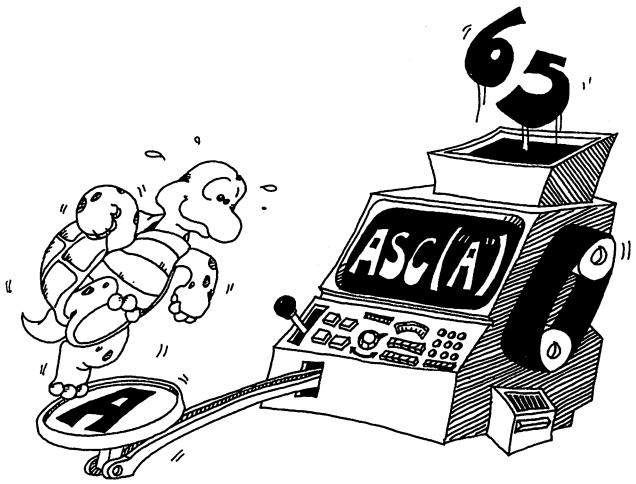
Use the ASC() function to change characters into ASCII numbers.

Run:

```
10 REM * KEY NUMBER? *
25 PRINT "clr PRESS KEY dn"
26 PRINT "TO SEE ASCII NUMBER dn"
30 GET C$:IF C$ = "" THEN GOTO 30
40 PRINT C$;TAB(5);ASC(C$)
50 GO TO 30
```

Try out some letters, digits, punctuation, and graphics characters. Try also the RETURN, INST DEL, CLR HOME keys, and other keys.

Press STOP to end the program. Then SAVE it to tape.



ALPHABETICAL LIST

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

Run:

```
10 REM ALPHABETIZE
30 PRINT"clr GIVE ME A LETTER dn"
31 INPUT A$
40 PRINT"dn GIVE ME ANOTHER dn"
42 INPUT B$
45 A=ASC(A$):B=ASC(B$)
47 REM PUT IN ORDER BY
48 REM LOOKING AT
49 REM ASCII NUMBERS
50 IF A>B THEN X=A:A=B:B=X:REM SWAP
THEM
60 PRINT"dn IN ORDER dn"
65 PRINT CHR$(A);TAB(5);CHR$(B)
```

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

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

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

THE ASCII NUMBERS FOR CHARACTERS

Briefly:				
	65	to	90	letters
	48	to	57	numbers
	32	to	47	punctuation
	58	to	64	punctuation
	91	to	95	punctuation
	96	to	127	graphics characters
	161	to	191	graphic characters

SPACE is 32 and RETURN is 13.

CHANGING NUMBERS INTO CHARACTERS

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

This program prints all the ASCII characters. Some of them make changes on the screen such as color changes or clearing the screen or line feeds. Follow the changes by looking at Appendix J in your VIC 20 Manual.

Run:

```
10 REM DISPLAY ASCII
20 PRINT"clr"
30 FOR I=0 TO 191
32 IF I=5 THEN POKE 36879,8: REM
BLACK SCREEN
34 IF I=14 THEN PRINT CHR$(142):REM
UPPER CASE
36 IF I=28 THEN POKE 36879,24: REM
WHITE SCREEN
40 PRINT I, CHR$(I)
50 FOR T=1 TO 500:NEXT T
60 NEXT I
```

Save the program to tape.

Nothing gets printed for numbers 0 through 31. But some numbers make changes in the screen. Number 5 changes the printing to white, 13 is RETURN and causes a line to be skipped, 19 homes the cursor, etc.

THE ON...GOTO COMMAND

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

This means that:

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

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

Assignment 29:

1. Write a program which asks for a word. Then it rearranges all the letters in alphabetical order.
2. Write a program that speaks "double dutch." It asks for a sentence, then removes all the vowels and prints it out.
3. Write a program that uses GET to get a letter (A to C) to use in a menu. Change the letter to a number (1 to 3). Then use the ON...GOTO command to pick which menu item to do.

INSTRUCTOR NOTES 30 ARRAYS AND THE DIM COMMAND

This lesson introduces arrays. The DIM() statement is described.

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

Two-dimensional arrays are compared to the rectangular array of cells on the TV screen.

Higher dimensional arrays are just mentioned, with no examples given.

QUESTIONS:

1. What does the DIM AD$(5) command do?
2. Where do you put the DIM command in the program?
3. What two kinds of array families are there?
4. What is the "index" or "subscript" of an array?
5. What does the command DIM SR(5,9) do?

LESSON 30 ARRAYS AND THE DIM COMMAND

MEET THE ARRAY FAMILY

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

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

Here is a family of numerical variables:

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

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

Instead of "family" we should say "array."

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

THE DIM() COMMAND RESERVES BOXES

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

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

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

saves four memory boxes, one each for the variables A(0), A(1), A(2) and A(3). These boxes are for numbers and contain the number "0" to start with. Another example:

```
30 DIM A(3),B$(4)
```

This time, DIM reserves 4 boxes for the A() array and 5 for the string array B$(). The boxes named B$(0) through B$(4) are for strings and are empty to start with.

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

MAKING A LIST

Enter:

```
10 REM ++ IN A ROW ++
30 DIM A$(5)
35 PRINT"clr dn dn ENTER A WORD dn"
40 FOR N=0 TO 5
45 IF N>0 THEN PRINT"ANOTHER dn"
50 INPUT A$(N)
55 PRINT
60 NEXT N
100 PRINT"IN A ROW dn dn"
110 FOR I=0 TO 5
120 PRINT A$(I);TAB(10);I
130 NEXT I
```

Run and save to disk.

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. Lines 50 and 120 in the program "IN A ROW" are in loops where the index (N or I) keeps changing.

MAKING TWO LISTS

Enter:

```
10 REM PHONE LIST
30 DIM NAME$(20), NUMBER$(20)
35 I=0
40 PRINT "clr dn ENTER DATA dn"
```

```
50 INPUT"NAME? ";N$
55 IF N$="END" THEN END
56 NA$(I)=N$
60 INPUT"NUMBER? ";NU(I)
65 PRINT
70 I=I+1:GOTO 50
```

Run. Press STOP key to stop program. Save to tape.

ONE DIMENSION, TWO DIMENSION, ...

The arrays that have one index are called "one dimensional arrays." But arrays can have 2 or more indices. Two-dimensional arrays have their "family members" put in a rectangle like the days in a month on a calender.

```
10 REM +++ TWO-DIM ARRAY +++
20 DIM T(4,5)
30 FOR X=0 TO 4
40 FOR Y=0 TO 5
50 T(X,Y)= X+Y
60 NEXT Y,X
70 REM PRINT OUT ARRAY
75 PRINT"clr dn dn dn dn"
80 FOR J=0 TO 5
82 PRINT"dn"
85 FOR I=0 TO 4
87 PRINT TAB(3+2*I) T(I,J);
90 NEXT I,J
```

Assignment 30:

1. Finish the PHONE LIST program so that it prints out the list of names with the telephone numbers beside them.
2. Use a two dimensional array to make a "weekly calendar" program. It could use an array made by DIM AR$(7,24) so that each day of the week could have an entry for each hour.

INSTRUCTOR NOTES 31 LOGIC: AND, OR, NOT

This lesson treats the AND, OR and NOT relations and the numerical values for TRUE and FALSE. These are important for some types of IF statements.

The TEENAGER program in lesson 12 used a nested IF to print out "You are a teenager." A more concise logic uses the OR relation.

There are several abstract ideas in this lesson that are difficult to grasp. The fact that TRUE and FALSE have numerical values of -1 and 0 is bad enough. But in addition, the computer sometimes treats any number that is not zero as being TRUE.

QUESTIONS:

For each IF statement, tell if anything will be printed:

1.
```
10 IF 3=3 THEN PRINT "HI"
10 IF NOT(3=3) THEN  PRINT "HI"
10 IF 3=3 OR 0=2 THEN PRINT "HI"
10 IF 3=3 AND 0=2 THEN PRINT "HI"
10 IF "A"="B" THEN PRINT "HI"
10 IF NOT("A"="B") THEN PRINT "HI"
```

2. What number will each of these lines print?

```
10 A=-1 PRINT A, NOT A
10 A= 0 PRINT A, NOT A
10 A=-1:B=-1 PRINT A AND B
10 A=0:B=-1 PRINT A AND B
10 A= 0:B= 0 PRINT A AND B
10 A= 0:B=-1 PRINT A OR B
10 A= 0:B= 0 PRINT A OR B
10 PRINT NOT 0
```

LESSON 31 LOGIC: AND, OR, NOT

ANOTHER TEENAGER PROGRAM

Enter:

```
10 REM < AND, OR, NOT >
20 PRINT"clr dn"
30 INPUT"NAME";N$
35 PRINT
40 INPUT"AGE";A
45 PRINT
50 IF (A>12) AND (A<20) THEN PRINT
N$;" IS A TEENAGER.dn"
55 NFLAG = (A<13) OR (A>19)
60 IF NFLAG THEN PRINT N$;" IS NOT
A TEENAGER!dn"
65 PRINT
70 IF (NOT NFLAG) AND (A=16) THEN
PRINT "AND ";N$;
:PRINT"dn IS red SWEET SIXTEEN!blk
dn "
```

Run and save to tape.

WHAT DOES "AND" MEAN?

Two things are true about teenagers: They are over 12 years old and they are less than 20 years old. Look at line 50.

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

WHAT DOES "OR" MEAN?

In line 55 the OR is used. Two things are said: "age is under 13" and "age is over 19."

Only one of them needs to be true for you to be "not a teenager."

IF (you are under 13) OR (you are over 20) THEN (you are not a teenager).

TRUE AND FALSE ARE NUMBERS

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

Rule:

TRUE is the number -1
FALSE is the number 0

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

To see these numbers, enter this in the Edit mode:

```
PRINT 3=7
```

The computer checks to see if 3 really does equal 7. It doesn't so it prints a "0" meaning FALSE.

and this:

```
PRINT 3=3
```

The computer checks to see if 3=3. If it does, the computer prints "−1" meaning "TRUE."

PUTTING TRUE AND FALSE IN BOXES

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

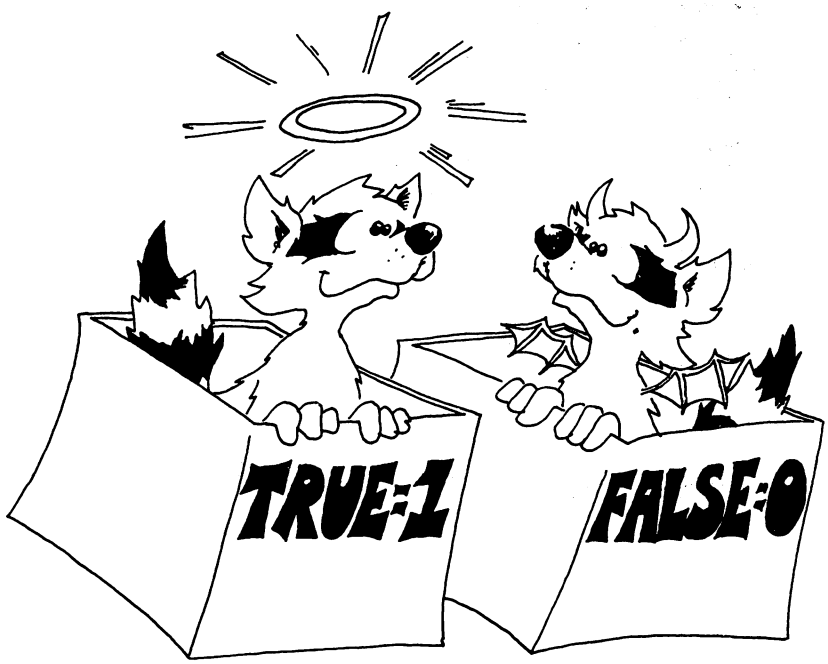
```
10 N= (3=22)
20 PRINT N
```

The number 0 is stored in the box N because 3=22 is FALSE.

And this:

```
10 N= "B"="B"
20 PRINT N
```

The number -1 is stored in the box N because the two letters in the quotes are the same so the statement "B"="B" is TRUE.



THE IF COMMAND TELLS LITTLE WHITE LIES

The IF command looks like this:

```
10 IF (something A) THEN (command C)
```

Try these in the edit mode:

```
IF 0 THEN PRINT "TRUE"
IF -1 THEN PRINT "TRUE"
```

Now try this:

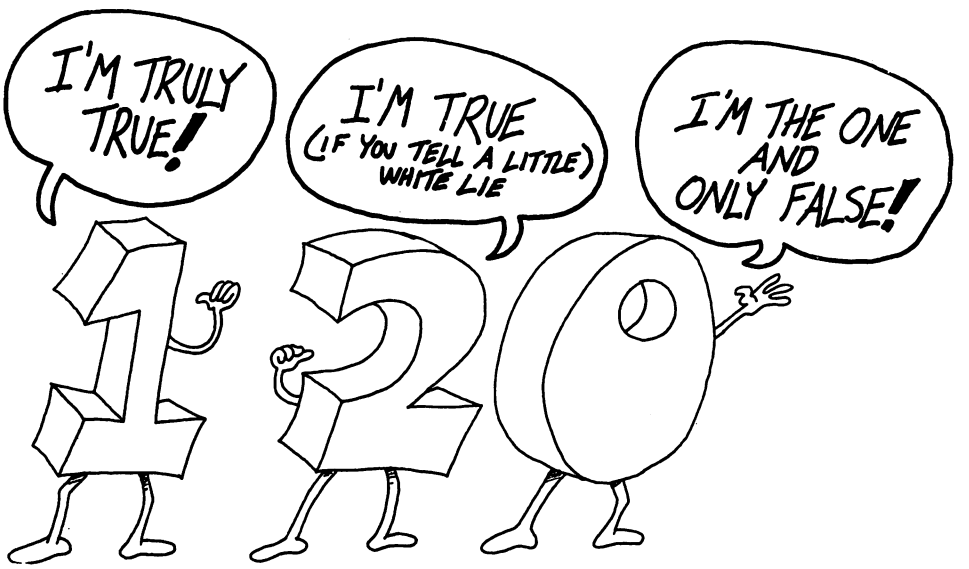
```
IF 22 THEN PRINT "TRUE"
```

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

If it is zero, the computer says "something A is FALSE", and skips what is after THEN.

If it is not zero, the computer says "something A is TRUE", and obeys the commands after THEN.

The IF command tells little white lies. TRUE is supposed to be the number "-1", but the IF stretches the truth to say "TRUE is anything that is not FALSE", that is, any number that is not zero is TRUE.

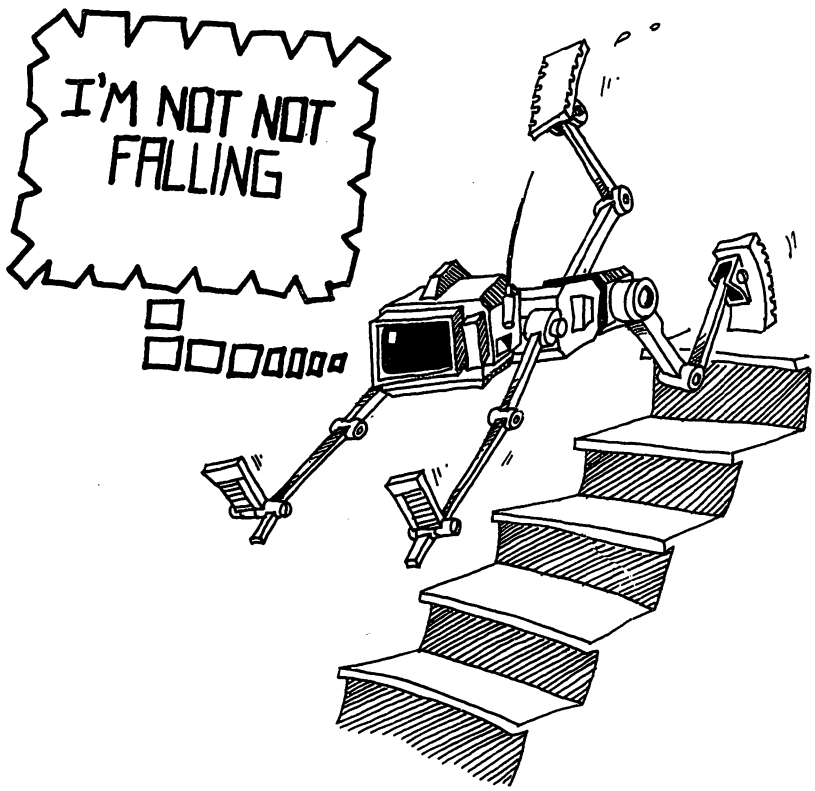


WHAT DOES "NOT" MEAN?

NOT changes FALSE to TRUE and TRUE to FALSE. Try this:

```
10 REM DOUBLE NEGATIVE
20 N=-1
30 PRINT "N "; TAB(10); N
40 PRINT "NOT N"; TAB(10); NOT N
50 PRINT "NOT NOT N ";TAB(10); NOT (NOT N)
60 REM The computer knows that "I don't have no...'
61 REM means ""I do have ....''
```

Save to tape.



Careful! `NOT (TRUE) = FALSE`

and `NOT (FALSE) = TRUE`

only works for real TRUE, the one where

`TRUE = -1`

It doesn't work for the little white lies where

`TRUE = any number except zero.`

Try this. Put N = 3 in the above program and see that that (NOT 3) doesn't give 0.

THE LOGICAL SIGNS

You can use these 6 symbols in the "something A" phrase:

```
=       equal
<>      not equal
<       less than
>       greater than
<=      less than or equal
>=      greater than or equal
```

You have to press two keys to make the ⟨⟩ sign, the ⟨= sign and the ⟩= sign.

The last two are new so look at this example to see the difference between ⟨ and ⟨⟩:

```
2<=3 is TRUE          2<3 is TRUE
3<=3 is TRUE          3<3 is FALSE
4<=3 is FALSE         4<3 is FALSE
```

These two "something A" phrases mean the same:

```
2<=Q          (2<Q) OR (2=Q)
```

Assignment 31:

1. Tell what will be found in the box N if:

```
N=4=4
N="G"<>"S"
N=5>7
N=3>2 AND 3<2
N=4=3 OR 4=4
N=NOT Ø
N=5>=4
```

2. Tell if the word "JELLYBEAN" will be printed:

```
IF Ø THEN PRINT "JELLYBEAN"
IF -1 THEN PRINT "JELLYBEAN"
IF 9 THEN PRINT "JELLYBEAN"
IF 3<>Ø THEN PRINT "JELLYBEAN"
IF Ø OR -1 THEN PRINT "JELLYBEAN"
IF NOT -1 THEN PRINT "JELLYBEAN"
IF "A"="Z" THEN PRINT "JELLYBEAN"
IF NOT(Ø) AND 2 THEN PRINT "JELLYBEAN"
IF NOT(Ø) OR -1 THEN PRINT "JELLYBEAN"
IF 4<=5 THEN PRINT "JELLYBEAN"
```

3. Write a program to detect a double negative in a sentence. Look for negative words like not, no, don't, won't, can't, nothing and count them. If there are two such words there is a double negative. Test the program on the sentence "COMPUTERS AIN'T GOT NO BRAINS."

INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS

This lesson concerns clear programs which interact with the user in a "friendly" way.

The "spaghetti" program should be discouraged. A format for writing programs is presented in this lesson. While methods of imposing order on the task are largely a matter of taste, the methods used in this lesson can serve to introduce the ideas.

"User friendly" means that the screen displays are easy to read, keyboard input is "RETURN key free" as much as possible, and errors are "trapped." Ask if entries are OK. If not, give an opportunity to fix things.

Instructions and "HELP" should be available. Prompts need be given. Beginners need complete prompts, but experienced users would rather have curt prompts.

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

Most young students will not progress very far toward fully "friendly" programming. To be acquainted with the desirability of "friendly" programming and to use some simple techniques toward accomplishing it are satisfactory achievements.

QUESTIONS:

1. Should your program give instructions whether the user wants them or not?
2. What is a "prompt?" Give two examples.
3. What is "scrolling?" How can you write to the screen without scrolling?
4. If you want the user to enter a single letter from the keyboard, what command is best (avoids using the RETURN key)?
5. What is an "error trap?" How would you trap errors if you asked your user to enter a number from 1 to 5?
6. In what part of the program are most of the GOSUB commands found?
7. Why put the "STARTING STUFF" section of the program at the end of the program (at high line numbers)?

LESSON 32 USER FRIENDLY PROGRAMS

There are two kinds of users:

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

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

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

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

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

PROGRAMS HAVE THREE PARTS

"STARTING STUFF": at the beginning of the program run.

give instructions to the user
draw a screen display
set variables to their starting values
ask the user for starting information

MAIN LOOP:

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

SUBROUTINES:

do parts of the program

PROGRAM OUTLINE

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

PUT THE MAIN LOOP AT THE BEGINNING OF THE PROGRAM

Put the MAIN LOOP near the front because it will run faster there.

PUT STARTING STUFF AT THE END OF THE PROGRAM

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

PUT SUBROUTINES IN THREE PLACES

between line 2 and line 99 for subroutines that must run fast
after line 2999 for starting stuff subroutines
between 200 and 999 for the rest of the subroutines

INFORMATION PLEASE

```
280 PRINT "DO YOU WANT INSTRUCTINS <Y/N> "
```

This lets a beginner see instructions and lets others say "no."

TIE A STRING AROUND THE USER'S FINGER

Use a "prompt" to remind users what choices they have. Example: `<Y/N>` where the choice is Y for "yes" or N for "no"

Beginners need long prompts. Other users like short prompts.

DON'T GIVE THE USER A HEADACHE

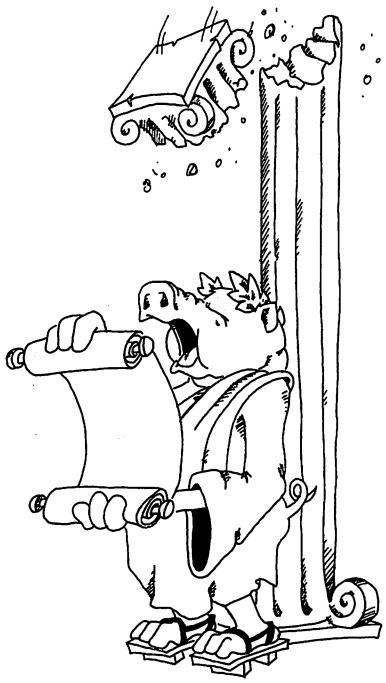
SCROLLING gives headaches!

BASIC usually scrolls. It writes new lines at the bottom of the screen and pushes old lines up.

It is like the scrolls the Romans used for writing. They unwound from the bottom and wound up at the top.

Avoid scrolling. Use CRSR arrows in PRINT to print just where you want. Erase by printing a string of blanks to the same spot.

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



OUCH! MY FINGERS HURT

Use the GET command to enter single letters. This saves having to press RETURN.

```
380 PRINT "DO YOU NEED INSTRUCTIONS? <Y/N> "
382 GET R$:IF R$="" THEN GOTO 382
384 IF R$="Y" THEN GOSUB 3400
```

SET TRAPS FOR ERRORS

Example: Add this line to the above lines:

```
386 IF R$<>"N" THEN GOTO 380
```

Line 380 asked for only two choices (Y or N). If the user presses some other key, line 386 sends him back to line 380.

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

Assignment 32:

1. Look at the EATER program. Add REM's to explain the lines in the program. Make changes in the colors and characters used.
2. Write a secret cipher program. The user chooses a password and it is used to make a cipher alphabet like this:

   if the password is DRAGONETTE
   remove the repeated letters, get DRAGONET
   put it at the front of the alphabet and the rest of
   the letters after it in normal order

   ```
   DRAGONETBCFHIJKLMPQSUVWXYZ
   ```

   The user chooses to code or decode from a menu.

INSTRUCTOR NOTES 33 DEBUGGING, STOP, CONT

It is difficult to drill systematically on debugging, unless you are NASA with NASA's budget and time scale.

We present a series of small techniques and a description of how to put them together in a debugging scheme. Only practice will serve to make debugging a chore that the student approaches with some confidence.

QUESTIONS:

1. What two ways can you make the computer print

   ```
   BREAK IN LINE 55
   ```

   while the program is running?
2. How are the STOP and the END commands different?
3. How is the STOP command different from the STOP key?
4. What does the CONT command do?
5. Why would you put STOP commands in your program?
6. How do delay loops help you debug a program?
7. How do extra PRINT commands help you debug a program?
8. Why do you take the STOP and extra PRINT commands out of the program after you have fixed the errors?
9. Can you pick in what line the STOP key will stop the program? Can you pick using the STOP command?

LESSON 33 DEBUGGING, STOP, CONT

THE STOP COMMAND

Enter and run:

```
10 REM SECRET STOP
20 PRINT"clr"
25 R=INT(RND(1)*200)
30 FOR I=0 TO 200
40 IF I=R THEN STOP
50 NEXT I
```

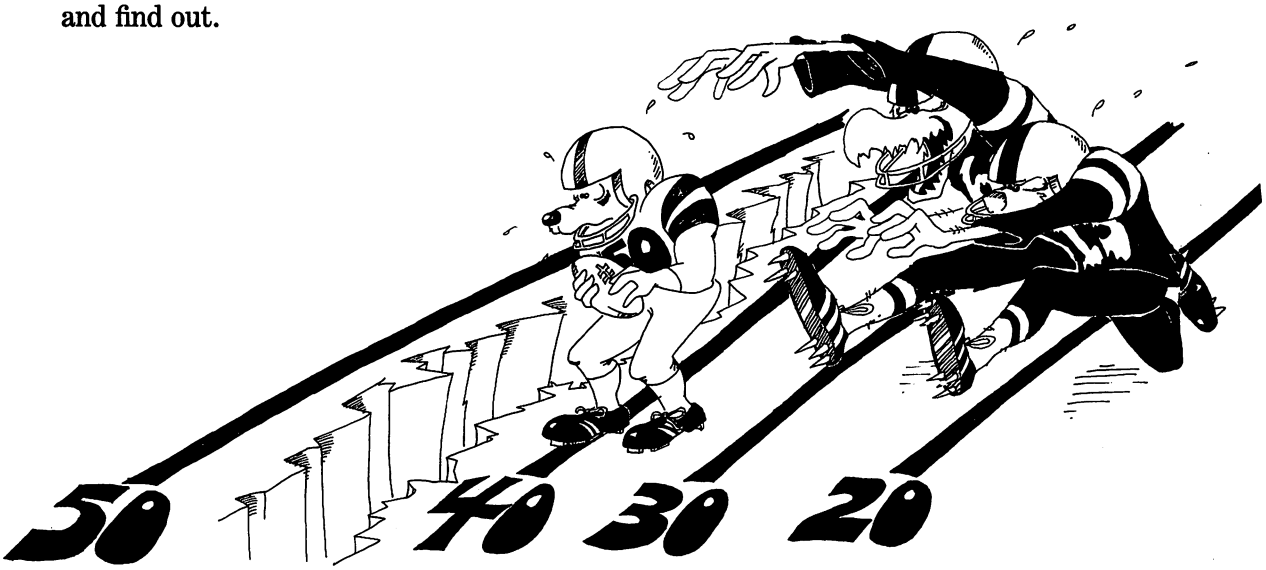
The program will stop, and the computer will peep and print a message:

```
BREAK IN LINE 40
```

What do you suppose the secret value of I was?

Enter: `PRINT I` (No line number)

and find out.



HOW TO START IT AGAIN

Enter the command CONT. Try it!

"STOP" IS LIKE "END"

STOP makes the computer stop and enter the Edit mode.

It is like END except it peeps and prints the number of the line that the STOP is in.

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

STOP is used for debugging your program.

ANOTHER WAY TO STOP RUNNING THE PROGRAM

You can stop running the program by pressing the STOP key.

Try it:

```
10 REM GO FOREVER
15 PRINT"clr"
16 FOR T=1 TO 1000:NEXT
20 PRINT "clr dn MUD dn"
22 PRINT "  TURTLES dn"
24 PRINT "    OF dn"
26 PRINT "     THE dn"
28 PRINT "      WORLD dn"
30 PRINT "       UNITE! dn"
40 FOR T=1 TO 2000:NEXT T
99 GOTO 10
```

Pressing the STOP key stops the program at its present spot. It prints:

```
BREAK IN LINE XX
```

peeps and enters the Edit mode

(where XX is the line number where it stops.)

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

The above program usually stops in line 40. Why? Try to make the above program stop in some other line.

WHAT DO YOU DO AFTER YOU STOP?

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

(Or you use the STOP key to stop the program, but it may not stop in the trouble spot.)

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

You are in the Edit mode. You can:

List parts of the program and study them.
Use the PRINT command to look at variables. Do they have the values you expected?
Do little calculations on the computer in the "calculator mode" (another name for the "Edit mode") to check what the computer is doing.

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

STARTING THE PROGRAM AGAIN

There are four ways to start a program. They are:

`CONT`	if you have not changed the program
`GOTO XX`	where XX is a line number
`RUN XX`	where XX is a line number
`RUN`	your old friend

You may use the CONT command if you have not:

added a line
deleted a line
or changed a line by editing it

Or you may start running the program at a different spot by entering (without line number) the command:

```
GOTO XX
```

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

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

What is the difference between these four ways?

`CONT` `GOTO XX`
These two ways use the values in the variable boxes left over from the last time you ran.

`CONT`	starts at the line where the BREAK occurred.
`GOTO XX`	starts at line XX

`RUN` `RUN XX`
These two ways throw away all the variable boxes made the last time, then execute the program.

`RUN`	starts at the first line of the program
`RUN XX`	starts at line XX

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

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

DEBUGGING

Little errors in your program are called "bugs."

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

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

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

Assignment 33:

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

RESERVED WORDS

ABS	AND	ASC	ATN		
CHR$	CLR	CMD	CLOSE	CONT	COS
DATA	DEF	DIM			
END	EXP	FN	FOR		
GET	GOSUB	GOTO			
FRE					
IF	INPUT	INPUT #	INPUT	INT	
LEFT$	LEN	LET	LIST	LOAD	LOG
MID$					
NEW	NEXT	NOT			
ON	OR PRINT	OPEN PRINT #	PEEK PRINT	POKE	POS
READ	REM	RESTORE	RETURN	RIGHT$	RND
	RUN				
SAVE	SGN	SIN	SPC(	SQR	STEP
	STOP	STR$	SYS		
TAB(	TAN	THEN	TO		
USR					
VAL	VERIFY				
WAIT					

ANSWERS TO ASSIGNMENTS

```
A 1-3
10 REM HI THERE
20 PRINT "HI THERE,"
30 PRINT "COMPUTER!"

A 2-1
10 REM COLORED NAMES
20 PRINT "■"
30 PRINT "■■MINDA"
40 PRINT "■■ANNE"
50 PRINT "■■CARLSON■"

A 2-2
10 REM COLORED NAMES
20 PRINT "■"
30 PRINT "■■M■I■N■D■A■"
40 PRINT "■■ANNE"
50 PRINT "■■CARLSON■"

A 3-5
10 REM BIRDS
20 PRINT "■"
30 PRINT
40 PRINT
50 PRINT "  ◜●◝"
60 PRINT
70 PRINT
80 PRINT "■◟◯◞"
90 PRINT
100 PRINT "■—●—"
110 PRINT "■"


A 4-2
10 REM SMILE
15 PRINT "■"
20 PRINT
25 PRINT
30 PRINT
35 PRINT "■"
40 PRINT "      00     00"
45 PRINT "      00     00"
50 PRINT
55 PRINT
60 PRINT
65 PRINT
70 PRINT
75 PRINT
80 PRINT " *                    *"
85 PRINT "    *                *"
90 PRINT "       *          *"
95 PRINT "          ****"
```

```
A 5-1
10 REM SILLY GOOSE
15 PRINT "■"
20 PRINT "■"
25 PRINT
30 PRINT
35 PRINT
40 PRINT "HELLO"
45 PRINT
50 PRINT "WHAT IS YOUR NAME?"
55 PRINT
60 INPUT N$
65 PRINT "■■"
70 PRINT
75 PRINT "WELL,"
80 PRINT
85 PRINT N$
90 PRINT
95 PRINT "IT IS SILLY TO TALK"
100 PRINT
105 PRINT "TO A COMPUTER!"
110 PRINT "■"

A 5-2
10 PRINT "■■"
20 PRINT "■■WHAT IS YOUR FAVORITE COLOR?"
25 PRINT
30 INPUT C$
35 PRINT "■■I PUT THAT IN BOX C$."
40 PRINT "NOW YOUR FAVORITE ANIMAL?■"
45 INPUT C$
50 PRINT "■■I PUT THAT IN BOX C$ TOO"
55 PRINT "NOW LET'S SEE WHAT IS IN BOX C$"
60 PRINT "IT IS:"
65 PRINT
70 PRINT C$

A 6-1
10 PRINT "■"
20 PRINT "NAME A MUSICAL GROUP"
25 INPUT A$
30 PRINT
35 PRINT "NAME ONE OF THEIR SONGS"
40 PRINT
45 INPUT B$
50 PRINT
55 PRINT A$;" PLAYS ";B$
```

```
A 6-2
10 REM 3 PRINTS
15 PRINT "■"
20 PRINT "GIVE ME THE NAME OF A MUSICAL GROUP"
25 INPUT G$
30 PRINT
35 PRINT "WHAT IS ONE OF THEIR TUNES"
40 INPUT T$
45 PRINT "■"

50 PRINT G$;
55 PRINT " PLAYS ";
60 PRINT T$

A 6-3
10 REM TUNES
15 PRINT "■"
20 POKE 36878,3
25 POKE 36875,135
30 FOR T=1 TO 250:NEXT T
35 POKE 36875,147
40 FOR T=1 TO 250:NEXT T
45 POKE 36875,159
50 FOR T=1 TO 500:NEXT T
55 POKE 36875,0

A 7-2
10 REM GLUING STRINGS
15 PRINT "■■■■"
20 PRINT "YOUR FAVORITE DESSERT?"
25 INPUT D$
30 PRINT "■■■■"
35 PRINT "YOUR FAVORITE TV STAR"
40 INPUT S$
45 PRINT "■■■■"
50 P$=S$+" EATS "+D$
55 PRINT P$

A 8-3
10 REM COLORFUL FRIENDS
15 PRINT "■■■■"
20 PRINT "■BRIAN"
25 PRINT
30 FOR T=1 TO 200:NEXT T
35 PRINT "■VIC"
40 PRINT
50 FOR T=1 TO200:NEXT T
```

```
A 9B-1
10 REM PIZZA
15 PRINT "■■■"
20 PRINT "HALLO. AY AM MARIO, YOUR PIZZA MAN."
25 PRINT "■JUST TELL ME ZE GORY DETAILS AND I'LL "
30 PRINT "DO ZE REST."
35 PRINT "■WHAT SIZE SHOULD ZIS PIZZA BE (S/M/L/)?"
40 INPUT S$
45 IF S$="S" THEN PRINT "■ON A DIET? HO HO!"
50 IF S$="M" THEN PRINT "■GOOD CHOICE-NOT TOO BIG, BUT
FILLING!"
55 IF S$="L" THEN PRINT "■YOU MUST HAVE A BIG BUNCH AT
HOME!"
60 PRINT "■NOW, YOUWANT DOUBLE CHEES ON ZIS (Y/N)?"
65 INPUT CH$
70 REM ETC.
75 REM MUSHROOMS, ETC.
80 REM ANCHOVIES, ETC.
85 REM PEPPERS, ETC.
90 REM MEAT, ETC.
95 PRINT "■■HOKAY, HERE IS YOUR PIZZA!"
100 PRINT "■■"
105 IF S$="S" THEN PRINT "WAN SMALL PIZZA WITH ";
110 REM ETC.
120 IF BASE$="P" THEN PRINT "PEPPERONI"
125 REM ETC., , ETC.
130 PRINT
135 FORJ=1 TO 2000
140 NEXT J

A 9 B-2
10 REM COLOR GUESSER
15 PRINT"■■■■"
20 PRINT "PLAYER 2 TURN YOUR BACK ■"
25 PRINT "PLAYER 1 ENTER A COLOR■"
30 INPUT C$
35 PRINT "■■■■"
40 PRINT "PLAYER 2 TURN AROUND AND GUESS ■■■"
45 INPUT G$
50 PRINT "■■■"
55 IF G$ <>C$ THEN PRINT "■WRONG! ■■"
60 IF G$=C$ THEN GOTO 70
65 GOTO 55
70 PRINT "■■■RIGHT! ■"
```

```
A 10-1
10 REM BIRTH YEAR
15 PRINT "■■■HOW OLD ARE YOU?■"
20 INPUT A
25 PRINT "AND WHAT YEAR IS IT NOW?■"
30 INPUT Y
35 B=Y-A
40 PRINT "HAS YOUR BIRTHDAY COME YET THIS YEAR?■"
45 PRINT "<Y/N>■"
50 INPUT Y$
55 IF Y$="N" THEN B=B-1
60 PRINT "■YOUWERE BORN IN";B;"■"

A 10-2
10 REM MULTIPLICATION
15 PRINT "■■■"
20 PRINT "GIVE ME A NUMBER ■"
25 INPUT A
30 PRINT "■■GIVE ME ANOTHER■"
35 INPUT B
40 C=A*B
45 PRINT "■■THEIR PRODUCT IS ■";C

A 11 A-1
10 REM PARTY GAME
15 PRINT "■■■THIS IS A PARTY GAME■■"
20 PRINT "■WHAT IS YOUR LAST NAME? ■■"
25 INPUT L$
30 PRINT "■■PLEASE TURN YOUR BACK ■■"
35 FOR T=1 TO 999:NEXT T
40 PRINT"■■■"
45 PRINT "SOMEONE PRINT IN A NICKNAME■■"
50 INPUT N$
55 FOR T=1 TO999:NEXT T
60 PRINT "■■■"
65 PRINT L$;" IS CALLED: ■■■"
70 POKE 35875,200
75 POKE 35878,15
80 FOR T=1 TO 999:NEXT T
85 POKE 35875,0
90 PRINT TAB(10);"'■";N$;"'■"

A 11 A-2
10 REM %#! INSULTS !#%
15 PRINT "■■■HEY YOU! WHAT IS YOUR NAME?■■"
20 INPUT N$
25 REM DELAY LOOP
30 FOR T=1 TO 2000:NEXT T
35 PRINT "■■■"
40 PRINT "■BAH!!!■■"
45 PRINT N$;TAB(5);"■YOUR FATHER EATS LEEKS!!!■"
```

```
A 11 B-1
10 REM MODIFIED FOR VIC
15 PRINT "■■■■"
20 PRINT "YOUR ... ■"
25 PRINT
30 FOR I=1 TO 2000:NEXT I
35 PRINT "  ... WAIT'S ...■"
40 PRINT
45 FOR I=1 TO 2000:NEXT I
50 PRINT "                OVER!!"

A 11 B-2
10 PRINT "■■■CLOCK PROGRAM■"
15 PRINT "PRESENT TIME* HR,MIN,SEC?■"
20 INPUT H,M,S
25 MUL=1000
30 FOR I=1 TO MUL:NEXT
35 PRINT H":"M":S"
40 S=S+1
45 IF S<60 GOTO 60
50 S=0
55 M=M+1
60 GOTO 60
65 REM SAME FOR HOURS

A 12 B-3
10 REM I GOT YOUR NUMBER
15 PRINT "■ ■■■"
20 PRINT "GIVE ME A NUMBER ■"
25 PRINT "BETWEEN ZERO AND TEN. ■"
30 INPUT N
35 IF N=0 THEN PRINT "■■I GOT PLENTY OF NOTHING!■"
40 IF N=1 THEN PRINT "■■I'M NUMBER ONE! ■"
45 IF N=2 THEN PRINT "■■TWO IS COMPANY.■"
50 REM ETX.
55 IF N>10 THEN GOTO 99
60 FOR T=1 TO 2000:NEXT T
65 GOTO 20
99 PRINT "■ THAT'S ALL FOLKS!■"

A 13-1
10 REM PAIR OF DICE
15 PRINT "■■■"
20 LET D1=1+INT(RND(8)*6)
25 LET D2=1+INT(RND(8)*6)
30 LET D=D1+D2
35 PRINT "■"TAB(5);"THE FIRST DIE ";D1
40 PRINT "■"TAB(5);"THE SECOND DIE";D2
45 PRINT "■"TAB(5);"THE DICE      ";D
50 PRINT "■■AGAIN? ";
55 INPUT Y$
60 IF Y$="Y" THEN GOTO 15
```

```
A 13-2
10 REM PAPER, SCISSORS AND ROCK
15 PRINT "■■■■"
20 PRINT "PLAY THE ■■■"
25 PRINT TAB(5)"P A P E R ■"
30 PRINT TAB(7)"S C I S S O R S"
35 PRINT TAB(9)"R O C K ■■■"
40 PRINT "          GAME"
45 PRINT "■■■ENTER <P,S,R> ■■■"
50 C=INT(RND(1)*3)+1
55 PRINT "■"
60 IF C=1 THEN C$="P"
65 IF C=2 THEN C$="S"
70 IF C=3 THEN C$="R"
80 INPUT Y$
85 IF C$<>Y$ THEN GOTO 100
90 PRINT "■"TAB(5)"■TIE"
95 GOTO 50
100 REM WHO WINS?
105 IF C$="P" THEN IF Y$="S" THEN GOTO 135
110 IF C$="S" THEN IF Y$="R" THEN GOTO 135
115 IF C$="R" THEN IF Y$="P" THEN GOTO 135
120 REM COMPUTER WINS
125 PRINT "■"TAB(5)"■■COMPUTER WINS"
130 GOTO 50
135 REM YOU WIN
140 PRINT "■"TAB(5)"YOU WIN"
145 GOTO 50

10 REM !!VACATION!!
12 PRINT "■■■■"
20 REM HEADING
21 PRINT " VACATION■"
22 PRINT "■   CHOOSING ■"
23 PRINT "■       PROGRAM "
25 PRINT "■■PICKS YOUR VACATION■"
26 PRINT "■BY THE AMOUNT"
27 PRINT "■ YOU WANT TO SPEND■"
30 REM INSTRUCTIONS
31 FOR T=1 TO 2000:NEXT T
33 PRINT "■■■ENTER THE AMOUNT■"
34 PRINT "IN DOLLARS"
35 REM GET DOLLAR AMOUNT
37 INPUT D
38 PRINT:PRINT
40 M$="FLIP PENNIES WITH YOUR KID BROTHER"
41 P$="■SPEND THE AFTERNOON IN BEAUTIFUL HOG WALLOW,
MICH."
42 Q$="ENTER A PICKLE EATING CONTEST IN SCRATCHYBACK,
TENN."
48 Z$="■BUY  A COSY YACHT AND CRUISE THE CARIBBEAN SEA."
```

```
70 IF D<.50 THEN PRINT M$:GOTO 90
71 IF D< 1 THEN PRINT P$:GOTO 90
72 IF D<3 THEN PRINT Q$:GOTO 90
73 REM ETC.
74 REM ETC.
77 IF D<1000000 THEN PRINT Z$:GOTO 90
78 PRINT "TREAT YOUR WHOLE SCHOOL TO A 'ROUND THE WORLD
TRIP."
90 REM ENDING TO PROGRAM
```

A 16-1

```
10 REM BALL
20 PRINT "□[graphics]"
30 B$="□"
40 PRINTB$;
50 FOR T=1 TO 100:NEXT T
60 B$=" "+B$
70 PRINT "[graphics] ";
99 GOTO 40
```

A 16-2

```
10 REM BLINKING
25 PRINT "□[graphics] "
30 INPUT "YOUR NAME";N$
40 PRINT "□[graphics] "
45 PRINT "[graphics]□          ";N$
50 FOR T=1 TO 500:NEXT T
55 PRINT "[graphics]□       ";N$
60 FOR T=1 TO 500:NEXT T
99 GOTO 45
```

A 17 A-1

```
10 REM COUNTING BY 5'S
15 PRINT "□[graphics]"
20 FOR I-5 TO 100 STEP 5
30 PRINT I
35 FOR T=1 TO 300:NEXT T
40 NEXT
```

A 17 B-3

```
10 REM CLIMBING NAME
12 PRINT "□[graphics]"
15 N$-"STANISLAUS MAZURSKY"
20 FOR I=21 TO 1 STEP -1
30 PRINT N$
32 FOR T=1 TO 200:NEXT T
35 PRINT "□                         "
36 PRINT "□□□"
40 NEXT I
```

```
A 17 B-2
10 REM SLIPPING NAME
15 N$="BRIAN"
20 FOR I=1 TO 8
22 FOR T=1 TO 300:NEXT T
25 PRINT TAB(I*2);N$
30 NEXT I

A 17 B-4
10 REM FRIEND'S NAMES
15 PRINT "♥"
20 INPUT "NAME ";N$
22 PRINT
25 INPUT"OTHER NAME ";M$
30 FOR I=1 TO 5
36 PRINT "▣"
40 PRINT▣N$
41 FOR T=1 TO 200:NEXT T
46 PRINT "%"
50 PRINT M$
51 FOR T=1 TO 200:NEXT T
60 NEXT I

A 19-2
10 REM CHIRPING
15 PRINT "♥▣▣▣▣▣▣▣▣▣▣▣ ▣"
16 I=200
17 POKE 36878,15
20 PRINT "▣       ╰●╯▌▌▌"
21 FOR T=220 TO 230
22 POKE 36876,T
23 NEXT T
24 POKE 36876,0
30 I=I+1:IF I>245 THEN I=200
40 V=0:IF RND(8)<.2 THEN V=15
41 POKE 36878,V
50 PRINT "▣       ╭●╮▌▌▌"
51 FOR T=1 TO200:NEXT T
99 GOTO 20
100 POKE 36876,0

A 21-2
10 REM HEART
20 PRINT "♥";
30 NEXT
40 FOR S=8 TO 248 STEP 16
45 FOR B=0 TO 7
50 POKE 36879,S+B
60 FOR T=1 TO 500:NEXT
65 NEXT B
70 NEXT S
80 GOTO 40
```

```
A 22-3
10 REM SINBAD'S CARPET
20 PRINT "■"
100 CC=38400
105 C=CC+252
110 SC=7680
115 CH=96+128
150 FOR I=SC TO SC+505
155 POKE I,CH
160 NEXT I
210 FOR I=0 TO 11:FOR J=0 TO 10
212 R=)I+J)*7/22
215 K=K+(R+3)/(R+5)
216 IF K>7 THEN K=0
220 POKE C+I*22+J,K
222 POKE C-I*22+J,K
224 POKE C+I*22-J,K
226 POKE C-I*22-J,K
250 NEXT J,I

A 23-1
10 REM MENU MAKER
12 PRINT "■■■■"
20 PRINT "    WHICH COLOR:"
21 PRINT
22 PRINT "     <R> RED"
24 PRINT "     <Y> YELLOW"
26 PRINT "     <G> GREEN"
28 PRINT "     <B> BLUE"
29 PRINT
30 FOR T=1 TO 300:NEXT T
31 GET C$:IF C$="" THEN 31
35 IF C$="R" THEN C=2
36 IF C$="Y" THEN C=7
37 IF C$="G" THEN C=5
38 IF C$="B" THEN C=6
40 POKE 646,C
50 PRINT "COLOR"
60 PRINT "■"
99 REM MAKE A STAR OF THIS COLOR

A 22-1
10 REM DRAW A BOX
12 PRINT "■■■■"
20 PRINT "WHAT COLOR SQUARE?■"
21 PRINT "<0-7> ■"
25 INPUT C
30 SC=7680
31 CC=38400
40 X=8
41 Y=8
```

```
42 BC=22*Y+X
46 Q=SC+BC
56 POKE Q    ,79
57 POKE Q+1  ,99
58 POKE Q+2  ,99
59 POKE Q+3  ,80
60 POKE Q+22 ,101
61 POKE Q+44 ,101
62 POKE Q+66 ,76
65 POKE Q+25 ,103
66 POKE Q+47 ,103
67 POKE Q+69 ,122
68 POKE Q+67 ,100
69 POKE Q+68 ,100
70 Q=CC+BC
71 POKE Q    ,C
72 POKE Q+1  ,C
73 POKE Q+2  ,C
74 POKE Q+3  ,C
75 POKE Q+22 ,C
76 POKE Q+44 ,C
77 POKE Q+66 ,C
78 POKE Q+25 ,C
79 POKE Q+47 ,C
80 POKE Q+69 ,C
81 POKE Q+67 ,C
82 POKE Q+68 ,C

A 23-2
10 REM SILLY SENTENCES
15 PRINT "■■■■■"
16 PRINT "SILLY SENTENCES■"
17 PRINT "WANT INSTRUCTIONS?■
18 GET Y$:IF Y$="" THEN 18
19 IF Y$="Y" THEN GOSUB 100
20 PRINT "■THE SUBJECT:"
21 PRINT "END WITH A PERIOD"
22 GET L$:IF L$="" THEN 22
24 IFL$="." THEN 30
28 S$=S$+L$
29 GOTO 22
30 S$=S$+" "
32 PRINT "■■THE VERB:"
34 GET L$:IF L$="" THEN 34
36 IF L$="." THEN 42
38 S$=S$+L$
40 GOTO 34
42 S$=S$+" "
50 PRINT "■■THE OBJECT:"
52 GET L$:IF L$-"" THEN 52
54 IF L$="." THEN 70
```

```
56 S$=S$+L$
58 GOTO 52
70 S$=S$+L$
80 PRINT
81 PRINT "■ ■■■ "
85 PRINT S$
99 END
100 REM
105 PRINT "■■■■ "
110 PRINT "THREE PLAYERS, ENTER"
111 PRINT " PARTS OF A SENTENCE"
115 PRINT "■NO PLAYER CAN SEE"
116 PRINT "■WHAT THE OTHERS ENTER"
120 PRINT "■THE FIRST ENTERS THE SUBJECT"
130 PRINT "■THE SECOND ENTERS THE VERB"
140 PRINT "■THE THIRD ENTERS THE OBJECT"
148 FOR T=1 TO 5000:NEXT T
190 PRINT "■■■■"
199 RETURN

A 24-1
10 REM SUBROUTINES
15 PRINT "■■■■ "
20 GOSUB 100
22 GOSUB 200
24 GOSUB 300
26 IF RND(8)<.5 THEN GOSUB 400
99 END
100 REM
101 REM THE FIRST
102 REM
105 PRINT "NOW IS THE TIME"
150 GOSUB 900
199 RETURN
200 REM
201 REM SMOKE
202 REM
210 PRINT "■■FOR RED SMOKE"
250 GOSUB 900
251 PRINT "■■ "
299 RETURN
300 REM
301 REM SORRY
302 REM
310 PRINT "TO POUR OUT"
311 PRINT "■OF YOUR ■VIC ■2
350 GOSUB 900
399 RETURN
400 REM
401 REMPERHAPS
402 REM
```

```
410 PRINT "■■KI DON'T MEAN IT!>"
450 GOSUB 900
451 PRINT "■"
499 RETURN
900 REM
901 REM TIMER
910 FOR T=1 TO999:NEXT T
999 RETURN

A 25-1
10 REM COUSINS
14 FLAG=1
15 PRINT "■■■■"
16 IF FLAG=0 THEN GOSUB 300
20 RESTORE
30 INPUT"WHICH RELATIVE";R$
33 FLAG=0
35 READ F1$:READ N$
40 IF F1$=R$ THEN GOSUB 200
50 IF F1$="END" THEN GOTO 15
55 GOTO 35
200 REM
201 REM PRINT IT
202 REM
210 PRINT R$;TAB(12);N$
250 FOR T=1 TO2000:NEXT T
260 FLAG=1
299 RETURN
300 REM
301 REM NO ONE
302 REM
310 PRINT "■YOU DON'T HAVE A ";R$
320 PRINT
399 RETURN
400 DATA FATHER,HARRY
401 DATA MOTHER,DOLLY
402 DATA SISTER,ALICE
403 DATA SISTER,LYN
405 DATA GRANDFATHER,ROGER
406 DATA GRANDMOTHER,ADA
407 DATA GRANDMOTHER,SUZAN
420 DATA COUSIN,JOHN
421 DATA COUSIN,MARY
422 DATA COUSIN,FRANK
430 DATA UNCLE,BOB
431 DATA UNCLE,RICHARD
432 DATA UNCLE,VINCENT
440 DATA AUNT,HATTIE
441 DATA AUNT,GRETCHEN
499 DATA END,END
```

```
A 26-1
10 REM CIPHER MAKER
20 PRINT "■■■■"
25 PRINT "CODE MAKER■"
30 PRINT "ENTER A SENTENCE■"
32 INPUT S$
33 PRINT
34 S$=S$+" "
35 L=LEN(S$)
40 FOR I=1 TOL STEP 2
45 P$=MID$(S$,I,2)
50 Q$=RIGHT$(P$,1)+LEFT$(P$,1)
55 L$=L$+Q$
60 NEXT I
66 PRINT "■■HERE IS THE CODED "
67 PRINT "■SENTENCE:■■"
70 PRINT "■■";L$;"■"

A 26-2
10 REM ANSWERER
15 PRINT "■■■■"
20 PRINT "ENTER A QUESTION ■"
25 INPUT Q$
28 L=LEN(Q$)
31 REM TAKE OFF "?"
33 Q$=LEFT$(Q$,L-1)+","
36 REM LOOK FOR WORD END
39 FOR I=1 TO L
40 C$=MID$(Q$,I,1)
45 IF C$=" " THEN S1=I:I=L
46 NEXT
48 REM LOOK FOR WORD END
50 FOR I=S1+1 TO L
52 C$=MID$(Q$,I,1)
54 IF C$=" " THEN S2=I:I=L
56 NEXT I
58 REM TURN THE WORDS
60 S$=MID$(Q$,S1+1,S2-S1)
62 V$=LEFT$(Q$,S1)
63 PRINT
65 PRINT S$+V$+RIGHT$(Q$,L-S2)

A 26-3
10 REM * PIG LATIN *
12 PRINT "■■■■"
20 PRINT"PIG LATIN"
25 PRINT "■"
30 INPUT "■■GIVE ME A WORD:";W$
35 L=LEN(W$)
40 FORI=1 TOL
42 L$=MID$(W$,I,1)
```

```
44 T=(L$="A" OR L$="E" OR L$="I" OR L$="O" OR L$="U")
50 IF T THEN GOTO 60
55 NEXTI
60 P=I-1
65 PL$=RIGHT$(W$,L-P)+LEFT$(W$,P)+"AY"
70 IF I=1 THEN PL$=W$+"LAY"
80 PRINT "[illegible]";PL$
90 PRINT "[illegible]ANOTHER?"
91 GET Y$:IF Y$="" THEN 91
92 IF Y$="Y" THEN 25
```

```
A 27-1
10 REM ADDING BACKWARD
15 PRINT "[illegible]"
20 INPUT "GIVE ME A NUMBER";N
30 N$=STR$(N)
35 L=LEN(N$)
40 FOR I=1 TO L
45 B$=B$+MID$(N$,L+1-I,1)
50 B=VAL(B$)
55 NEXT I
57 PRINT "[illegible]"
60 PRINT "    ";N
61 PRINT "  +";B
65 PRINT "    ";LEFT$("------",L)
70 A=N+B
72 A$=STR$(A)
75 IF LEN(A$)=L THEN PRINT "    ";A
76 IF LEN(A$)=L+1 THEN PRINT "   ";A
79 END
```

```
A 27-2
10 REM MARCHING
12 PRINT "[illegible]"
20 INPUT "GIVE ME A NUMBER";N
22 D$=" [illegible] "
23 B$="                    "
25 N$=STR$(N)
27 L=LEN(N$)-1
28 N$=RIGHT$(N$,L)
30 REM LOOP
40 FORI=1TO22-L
50 PRINT D$;LEFT$(B$,I);N$
55 N$=RIGHT$(N$,L-1)+LEFT$(N$,1)
65 FOR T=1 TO 300:NEXT T
70 NEXT I
```

```
A 29-1
10 REM ALPHABETICAL
15 PRINT "■■■"
20 PRINT "IN ALPHAETICAL ORDER"
30 PRINT "■GIVE ME A WORD: ■"
31 INPUT W$
35 L=LEN(W$)
38 REM  TEST LETTERS
40 FOR I=65 TO 65+26
42 REM SEE IF IN WORD
45 FORJ=1 TO L
50 G=ASC(MID$(W$,J,1))
55 IF G=I THEN H$=H$+CHR$(G)
60 NEXT J-NEXT I
70 PRINT "■■HERE IT IS"
71 PRINT "■IN ALPHABETICAL ORDER:■■"
75 PRINT "      ";H$
80 PRINT "■"

A 29-2
10 REM DOUBLEDUTCH
15 PRINT "■ ■■■"
20 PRINT "GIVEME A SENTENCE:■"
30 INPUT S$
35 L=LEN(S$)
40 PRINT "■■■"
47 REM CHECK LETTERS
50 FORI=1 TO L
52 L$=MID$(S$,I,1)
60 IF L$="A" THEN 72
61 IF L$="E" THEN 72
62 IF L$="I" THEN 72
63 IF L$="O" THEN 72
64 IF L$="U" THEN 72
66 SS$=SS$+L$
72 NEXT I
75 PRINT "IN DOUBLE DUTCH"
70 PRINT "■■■";SS$
80 PRINT "■"

A 29-3
10 REM MENU PROGRAM
12 PRINT "■■■"
21 REM MAKE A MENU
30 PRINT "■MAKE YOUR CHOICE:"
31 PRINT "■■■<A> TAKE A NAP"
32 PRINT "<B> EAT AN APPLE"
33 PRINT "<C> CALL A FRIEND"
35 PRINT "■■"
40 INPUT X$
41 X=ASC(X$)-64
```

```
45 PRINT "■ "
50 ON X GOTO 60,70,80
52 GOTO 30
60 PRINT "YOUR BED IS NOT MADE."
61 END
70 PRINT "YOUR SISTER ATE THE LAST ONE."
71 END
80 PRINT "YOUR FATHER IS ON THE PHONE."
81 END

A 31-3
10 REM "AIN'T GOT NO"
12 PRINT "■■■■"
20 PRINT "ENTER A SENTENCE■■"
22 PRINT "NO PUNCTUATION ■ "
24 PRINT "EXCEPT APOSTROPHY ■■■ "
32 INPUT S$
33 S$=S$;" "
35 L=LEN(S$)
40 NN=0
42 S1=1:S2=1
45 FORI=1 TO L
50 L$=MID$(S$,I,1)
55 IF L$=" " THEN S1=S2:S2=I+1:GOSUB 200
60 NEXT I
65 PRINT "■■"
70 IF NN=0 THEN PRINT "NO NEGATIVE WORDS"
72 IF NN=1 THEN PRINT "A NEGATIVE SENTENCE"
74 IF NN=2 THEN PRINT "■DOUBLE NEGATIVE ■"
76 IF NN>2 THEN PRINT "MANY NEGATIVES"
80 FORT=1 TO 2000:NEXT T
198 GOTO 12
200 REM TEXTWORD
205 LW=S2-S1-1
210 W$=MID$(S$,S1,LW)
220 READ NW$
222 IF NW$="END" THEN RESTORE:GOTO 295
224 IFW$=NW$ THEN NN=NN+1
230 GOTO 220
295 RETURN
300 DATA
NO,NOT,NEVER,NEGATIVE,DON'T,AIN'T,DOESN'T,NOTHING,SHOULDN'T,
WOULDN'T,NONE,END

10 REM ****SYMBOLS****
11 REM
20 REM CLEAR     ■ "
21 REM
22 REM HOME      ■ "
23 REM
30 REM UP        ■ "
```

```
31 REM
32 REM DOWN      "
33 REM
34 REM LEFT      "
35 REM
36 REM RIGHT     "
37 REM
40 REM BLK       "
41 REM
42 REM WHT       "
43 REM
44 REM RED       "
45 REM
46 REM CYN       "
47 REM
48 REM PUR       "
50 REM
60 REM GRN       "
61 REM
62 REM BLU       "
63 REM
64 REM YEL       "
65 REM
66 REM RVS ON    "
67 REM
68 REM RVS OFF   "

10 REM SNAKE
20 GOTO 1000
100 REM MAIN LOOP
105 GETG$
110 IF G$="," THEN H=H-1:IF H<1 THEN H=4
112 IF G$="," THEN H=H+1:IF H=5 THEN H=1
114 FOR T=1 TO 50:NEXT T
115 ON H GOTO 120,122,124,126
120 Y=Y-1:GOTO 127
122 X=X-1:GOTO 127
124 Y=Y+1:GOTO 127
126 X=X+1
127 IF Y>22 THEN Y=22
128 IF X>21 THEN X=21
129 IF X<0  THEN X= 0
130 IF Y<0  THEN Y= 0
131 Z=X+22*Y
132 POKE SC+Z,81
133 POKE CC+Z,W
135 W=W+1:IF W>7 THEN W=2
140 A=B:B=C:C=D:D=E:E=F:F=X
142 L=M:M=N:N=O:O=P:P=Q:Q=Y
145 Z=A+22*L
```

```
146 POKE CC+Z,1
999 GOTO 100
1000 REM
1001 REM ** SNAKE**
1002 REM
1500 SC=7680
1501 CC=38400
1510 X=11:Y=12
1520 W=2
1600 PRINT "■"
2000 REM BORDER
2099 GOTO 100

A 32-2
10 GOTO 1000
100 REM MAIN LOOP
110 GOSUB 400
115 PRINT "■■■CODE OR DECODE?<C/D>"
116 GET Y$:IF Y$="" THEN 116
120 IF Y$="C" THEN 500
130 IF Y$=D" THEN 600
140 GOTO 115
400 REM GET PASSWORD
405 PRINT "INPUT PASSWORD ■"
406 INPUT PW$
408 F$=LEFT$(PW$,1)
410 FOR I=2 TO LEN(F$)
411 L1$-MID$(PW$,I,1)
412 FOR J=1 TO LEN(F$)
415 L2$=MID$(F$,J,1)
420 IF L1$=L2$ THEN 430
421 NEXT J
422 F$=F$+L1$
430 NEXT I
432 PW$=F$
433 PRINT "■SHORTENED: ■"
434 PRINT "   ";PW$
440 FOR J=1 TO LEN(PW$):L2$=MID$(PW$,J,1)
441 IF L2$=LEFT$(A$,1) THEN A$=MID$(A$,2):GOTO 460
442 FOR I=2 TO LEN(A$):L1$=MID$(A$,I+1)
445 IF L1$-L2$ THEN A$=LEFT$(A$,1)+MID$(A$,I+1)
455 NEXT I
460 NEXT J
462 REM FORM CODE ALPHABET
470 PRINT "■■CIPHER ALPHABET: ■"
471 PRINT A$
499 RETURN
500 REM CODE MESSAGE
505 PRINT "■■INPUT MESSAGE■"
510 GET L$:IF L$="" THEN 510
511 L=ASC(L$)
```

```
515 IF L$="*" THEN GOTO 590
520 IF L<65 ORL>91 THEN D$=D$+L$:GOTO 540
530 P$=P$+MID$(A$,L-64,1)
540 PRINT L$
589 GOTO 510
590 PRINT:PRINT P$
600 REM DECODE MESSAGE
610 PRINT "■TYPE IN MESSAGE"
612 PRINT "■END WITH A '*'"
615 GET L$:IF L$="" THEN 615
617 IF L$="*" THEN 690
620 FOR I=1 TO 26
625 IF L$=MID$(A$,I,1) THEN PRINT MID$(B$,I,1);:GOTO615
630 NEXT I
635 PRINT L$;
640 GOTO 615
690 END
1000 REM STARTING STUFF
1010 PRINT "■■■■"
1020 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
1030 B$=A$
1999 GOTO 100

10 REM COLOR EATER
12 GOTO 1000
15 REM EAT
20 POKE CC+R,1
21 POKE CC+Q,6
22 X=I:Y=J:Q=X+22*Y
24 PP=SC+Q:POKE PP,87
26 PP=CC+Q:POKE PP,5
28 POKE 36877,235
30 FOR T=1 TO 100:NEXT T
32 POKE 36877,0
35 N=0
39 GOTO 100
45 REM LOOK FOR FOOD
47 POKE CC+Q,6
48 POKE CC+R,1
51 X=X+INT(RND(1)*3)-1
52 Y=Y+INT(RND(1)*3)-1
53 IF X<0 THEN X=0
54 IF X>21 THEN X=21
55 IF Y<0 THEN Y=0
56 IF Y>22 THEN Y=22
57 R=X+22*Y
58 POKE SC+R,87
59 POKE CC+R,5
62 POKE 36876,220
64 FOR T=1 TO 50:NEXT T
65 POKE 36876,0
```

```
66 N=N+1
70 IF N<15 THEN 100
80 X=INT(RND(8)*22)
85 Y=INT(RND(8)*23)
95 N=0
100 REM MAIN LOOP
101 FOR I=X-1 TO X+1
102 IF I>21 THEN 116
103 IF I<0 THEN 116
104 FOR J=Y-1 TO Y+1
106 IF J<0 THEN 115
106 IF J>22 THEN 115
110 C1=PEEK(SC+P)
111 IF C1=C THEN GOTO 68
115 NEXT J
116 NEXT I
119 GOTO 35
1000 REM
1001 REM COLOR EATER
1002 REM
2000 REM STARTING STUFF
2010 PRINT "Ω"
2011 SC=7680
2012 CC=38400
2014 C=81
2020 FOR I=0 TO 21
2030 FOR J=0 TO 23
2040 IF RND(8)>.5 THEN 2050
2042 POKE SC+I+22*J,C
2044 POKE CC+I+22*J,2
2050 NEXT J,I
3000 X=8:Y=8
3010 POKE 36878,15
3999 GOTO 100
```

GLOSSARY

argument
The variable, number or string that appears in the parentheses of a function. Like:

INT(N)	has	N as an argument
LEFT$(W$,3)	has	W$ and 3 as arguments

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

A(0)	is the first member of the array A
B$(7)	is the eighth member of the array B$
CD(3,I)	is a member of the array CD

arrow keys
There are two CRSR keys on the VIC. Each has two arrows. There are two other keys that have arrows and print them as characters.

ASCII
Stands for American Standard Code For Information Interchange. Each character has an ASCII number. Some actions like line feed, carriage return and bell also have numbers.

assertion
The name of a phrase that can be TRUE or FALSE. The phrase we called "something A" in an IF statement is an assertion. An assertion is also a numerical expression. See expression, TRUE, FALSE, logic, IF, something A. Example:

the assertion	`"A"<>"B"`	is TRUE
the assertion	`3 = 4`	is FALSE

beep
A tone you may want to put in a program to call attention to something.

bell
The early teletype machines had a bell (like the bell on a typewriter). Computers make 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
(From "pull yourself up by your bootstraps.") Means to start up the computer from scratch. An easy thing to do with modern computers that have start up programs stored permanently in read-only memory (ROM). It was an involved procedure in the early days. Now it usually means to read in the disk operating system programs (DOS) from a disk.

branch
A point in a program where there is a choice of which statement to execute next. An IF statement is a branch. So is an ON...GOTO statement. A branch is not the same as a jump where there is no choice. See jump, IF.

buffer
A storage area in memory for temporary storage of information being inputed or outputed from the computer.

call
Using a GOSUB calls the subroutine. Putting a function in a statement calls the function. Call means the computer does what commands are in the subroutine or does the calculation that the function is for.

carriage return
On a typewriter, you push the lever that moves the carriage carrying the paper so a new line can begin. In computing, it means the cursor is moved to the start of the line, but not down to the next line. See line feed, CRLF.

character
Letters, digits, punctuation marks and the space are characters. So are the graphics characters.

cassette
The rectangular holder of magnetic recording tape.

clear
Means erase. Used in "clear the screen" and "clear memory."

column
Things arranged vertically. See row.

command
In BASIC, a command makes the computer do some action, such as list the program for the LIST command. See statement, expression. Some commands need expressions to be complete. Example:

```
POKE 2*J+1,3
```

concatenation
Means sticking two strings together by using a "+" sign. For example:

```
"HI"+"THERE"
```
gives the string "HI THERE"

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

CRLF
Short for "carriage return followed by line feed." This is what is called just a "carriage return" on a typewriter. See carriage return, line feed.

cursor
A marker that shows where the next character on the screen or in a storage buffer will be placed. Cursor means "runner." The cursor runs along the screen as you type. There are two kinds of cursors in the VIC:

INPUT cursor	a flashing square on the screen
PRINT cursor	invisible, "shows" where next character will be printed

data
BASIC has two kinds of data: numerical and string. Logical data (TRUE, FALSE) are types of numerical data.

debug
Means to run a program to find the errors and fix them. You fix the errors by editing the program. See edit.

deferred execution
Means run a stored program. See immediate execution

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

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

disk
Short for "diskette." Someone called it "a cross between a 45 RPM record and a magnetic tape." Used to store information in a permanent form. Like a magnetic tape, the information can be erased and new information can be recorded.

edit
There are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

enter
To put information into the computer by typing, then pushing the RETURN key. The information goes into the input buffer as it is typed. When RETURN is pushed, the computer uses the information.

erase
To destroy information in memory or write blanks to the screen. See clear.

error trap
Part of a program that checks for mistakes in information that the user has entered, or checks to see if computed results make sense.

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

expression
a portion of a statement that has a single value, either a number or a string. See value. Examples:

```
7*X=1                    3*LEN(D$)=RND(8)
GT$=LEFT$(D$,2)          "DOPE"<> N$
```

False
the number 0. See logic, assertion.

file
A collection of information that is on a tape, or is in the computer ready to put on a tape.

file name
The name of a file. See lesson 14 for legal names.

flag
A number or string used to mark that a condition is holding. Example: you set the variable S=0 if the arrow can be shot. You set S<>0 if an arrow is flying.

flashing box
The input cursor.

fork in the road
Describes a branch point in the program. See branch.

function
BASIC has a number of functions built in. Each function has a name followed by parentheses. In the parentheses are one or more arguments. The function has a single value (numerical or string) determined by its arguments. See value, argument. The functions treated in this book are:

```
ASC, CHR$, INT, LEFT$, LEN, MID$, PEEK, RIGHT$,
RND, STR$, VAL
```

graphics
Means picture drawing.

immediate execution
When a line that does not start with a number is entered from the edit mode, it is executed as soon as the RETURN key is pressed. If the line has only one command, you usually think of it as "entering a command." But the line may have several statements separated by colons, and thus it is a little program. Then you think of it as "executing a program in the immediate mode."

index
An array name is followed by one or more numbers or numerical variables in parentheses. Each number is an index. Another word for index is "subscript."

integers
The whole numbers, positive, negative and zero.

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

line
There are two kinds of lines in BASIC: Lines that start with a number are stored in the program in memory. Lines that do not start with a number are executed right away (see immediate execution). A line contains one or more statements, separated by colons.

Parts of a line:

`16`	line number
`IF A<=7 THEN PRINT Q$+LEFT$(...)`	first statement
`GOTO 40`	second statement
`IF A<=7 THEN`	a command
`PRINT Q$+LEFT$("RAT",K)`	a command
`GOTO 40`	a command
`A<=7`	an assertion
`A<=7`	an expression
`Q$+LEFT$("RAT",K)`	an expression

```
LEFT$("RAT",K)          a function
"RAT"                   an argument
K                       an argument
A, Q$, K                variables
7, "RAT"                constants
<=                      an operation
```

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

line editing
Retyping parts of a line to correct it. You do this by moving the cursor to the wrong part, and then typing the correct characters.

line feed
Moving the cursor straight down to the next line. See CRLF, carriage return.

line numbers
The number at the beginning of a program line. The line number tells the computer where to store the line. Some lines don't have numbers (the ones that will be executed in the immediate execution mode).

listing
A list of all the lines in a program.

load
To transfer the information in a file on tape to the memory of the computer by using the LOAD command.

logic
The part of a program that compares numbers or strings. The relations AND, OR, and NOT, and =, <>, <, >, <=, and >= are used. See assertion, IF, something A.

loop
A part of the program that is done over and over again. There are two kinds of loops: FOR...NEXT loops, and "home made" loops that use IF... commands with GOTO commands.

loop variable
Is the number that changes as the loop is repeated. For example:

```
FOR I=1 TO 5:NEXT I     I is the loop variable
```

memory
The part of the computer where information is stored. Memory is made of semiconductor chips, but we think of it as "boxes" with labels on the front and the information inside.

menu
A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

message
The string in quotation marks after an INPUT command. Example:

```
INPUT"TEXT, NUMBER ";T$,N
```

monitor
Has two meanings. We use it to mean a box with a TV type screen that is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

nesting
When one thing is inside of another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

```
L=LEN(MID$(A$,3,J))      nested functions
X=5*(6+(7*(8+K)))        nested parentheses
```

number
Is one type of information in BASIC. The numbers are generally decimal numbers. See integer, strings.

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

pointer
In a DATA statement, a pointer "shows" which item is next to be read. It is not visible or accessible to the programmer, but keeping it mentally "in view" helps in understanding the program.

program
There are two kinds. The usual program is a list of numbered lines containing statements. The computer executes the statements (commands) in order when the RUN command is entered. The program is stored in a special part of memory, and only one program can be stored at a time.

A one line program can be entered when the computer is in the edit mode. It does not start with a line number and runs as soon as you press the RETURN key. This one line program does not get mixed with the stored program. But when it runs, it may read or change the variables (if any) that the stored program made when it ran.

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

pseudo-random
A number that is calculated in secret by the computer using the RND() function. It is usually called a "random number." Pseudo-random emphasizes that the number really is not random (since it is calculated by a known method) but just is not predictable by the computer user.

punctuation
The characters like period, comma, /, ?, !, $, etc.

random
Numbers that cannot be predicted, like the numbers that show after the roll of dice, or the number of heads you get in tossing a coin 10 times.

remark
A comment you make in the program by putting it in a REM statement. Example:

```
REM THE NEXT 7 LINES ARE A SUBROUTINE
```

reserved words
A list of words and abbreviations that BASIC recognizes as commands or functions. The reserved words cannot be used in variable names. See the appendix.

return a value
When a function is used (called), its spot in the expression is replaced with a value (a number or a string). This is called "returning a value."

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

row
Things arranged horizontally (across).

save to tape
The program in the computer's memory is written (in magnetic code) on a tape cassette by the recorder when the SAVE command is executed:

```
SAVE "filename"
```

where "filename" stands for a name you choose to identify the program while it is on the tape.

screen
The TV screen or a similar one in a monitor that is hooked up to the computer. See monitor.

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

something A
Is a phrase in this book that stands for an assertion in an IF statement. See assertion, IF. Example:

```
IF B>4 THEN GOSUB 500
```

B>4 is "something A"

stack
Is a data type used in machine language programming. The 6502 processor has a stack in page $01 and it holds information about loops, subroutines, and nesting.

starting stuff
Is the name given in this book to initialization material in a program. It includes REM's for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things that need to be done just once at the beginning of a program run.

statement
The smallest complete section of a program. It starts with a command. The command may have expressions in it.

store
To put information in memory or to save a file on a disk.

string
A type of data in BASIC. It consists of a set of characters. See number.

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

subscript
A number in the parentheses of an array. It tells which member of the array is being used. See index.

syntax
Means the way a statement in BASIC is spelled. SYNTAX ERROR means the spelling of a command or variable name is wrong, the punctuation is wrong or the order of parts in the line is wrong."

timing loop
A loop that does nothing except use up time. See delay loop.

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

True
Has the value -1. See logic, FALSE, assertion.

typing
Pressing keys on the VIC. It is different from "entering." See enter.

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

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

variable name
A variable is either a string variable or a numerical variable. The name tells which. The rules for naming variables are given in lesson 20. The most important rule is that string variables have names ending in a "$" sign. Numerical variables do not have a dollar sign on the end.

INDEX OF COMMANDS AND FUNCTIONS EXPLAINED IN THIS BOOK

KEYS EXPLAINED IN THIS BOOK

ERROR MESSAGES

BAD DATA

You read a tape file which was supposed to have numerical data. But it found some string characters.

BAD SUBSCRIPT

You made an error using an array. Like:

DIM A(5,5):A(1,1,1)77 wrong number of subscripts

or DIM A(5): A(14)7 subscript was larger than 5.

CAN'T CONTINUE

You used the CONT command when it was not allowed.

DEVICE NOT PRESENT

Asked for an I/O device (tape, disk, printer, etc.) that was not present. Commands OPEN, CLOSE, CMD, PRINT, INPUT, or GET may give this. (We have not treated these commands in this book.)

DIVISION BY ZERO

You divided by zero. Or you divided by a variable whose value was zero.

EXTRA IGNORED

A comma was found on an INPUT command. Program continues anyway.

FILE NOT FOUND

You were looking for a file on tape, and the END-OF-TAPE marker was found.

FILE NOT OPEN

Must open a file before you use it with CLOSE, CMD, PRINT#, INPUT#, or GET#.

FILE OPEN

You tried to OPEN a file using a number of a file already open.

FORMULA TOO COMPLEX

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

ILLEGAL DIRECT

You used INPUT or GET in the edit mode. (Or you used DEF FN or DATA in the edit mode.)

ILLEGAL QUANTITY

You made one of these errors:

You used a negative number as an array subscript, like

LET A(−1)= 34

You used a function with the wrong kind of argument.

Like:

wrong: L=VAL(R)	correct L~AL(R$)
wrong: TAB(−3)	correct TAB(3)
wrong: SPEED=400	correct SPEED255

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

LOAD

Something is wrong with the program on tape.

NEXT WITHOUT FOR

You used a NEXT before the computer reached a FOR... statement. Or you used the wrong name for the variable. Like:

```
FOR I= 1 TO 5
NEXT M
```

NOT INPUT FILE

You opened a file for output but tried to get data from it with a GET or an INPUT.

NOT OUTPUT FILE

You opened a file for input but tried to PRINT to it.

OUT OF DATA

You tried to READ after you had already read all the data in the DATA statements in the program.

OUT OF MEMORY

Usually it means you have nested too many FOR...NEXT loops or too many subroutines inside each other, or an expression has too many parentheses.

OVERFLOW

You did a calculation which had a very large answer, too big for the computer to handle.

REDIM'D ARRAY

You made one of these errors: You used an array before you did the DIM command for it. like:

```
LET A(3)=7:DIM A(20)
```

or you executed DIM twice for the same array, like going through the DIM line twice.

```
2010 DIM B$(7)
```

REDO FROM START

On an INPUT command, the computer found letters or punctuation when it expected only numbers. Start entering data again from the beginning of the INPUT list.

RETURN WITHOUT GOSUB

You let the computer reach a RETURN command before it went through a GOSUB... command. This usually happens when the program accidently "runs into" a subroutine at the end.

STRING TOO LONG

You used concatenation to make a string longer than 255 characters.

SYNTAX

You "spelled" the line wrong. Maybe you forgot a (or a ; or put a # in a name, etc.

TYPE MISMATCH

You mixed numbers and strings, like:

```
LET A=""9'' or LET A$=33 or A$=LEFT(A$,1)
```

UNDEF'D FUNCTION

You forgot to use a DEF FN... command before using a user defined function.

UNDEF'D STATEMENT

You used a GOTO or a GOSUB to a line number that is not in your program.

VERIFY

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

ALPHABETICAL INDEX

V

Z

KEY SYMBOLS

```
10 REM ****SYMBOLS****
11 REM
20 REM CLEAR      "
21 REM
22 REM HOME       "
23 REM
30 REM UP         "
31 REM
32 REM DOWN       "
33 REM
34 REM LEFT       "
35 REM
36 REM RIGHT      "
37 REM
40 REM BLK        "
41 REM
42 REM WHT        "
43 REM
44 REM RED        "
45 REM
46 REM CYN        "
47 REM
48 REM PUR        "
50 REM
60 REM GRN        "
61 REM
62 REM BLU        "
63 REM
64 REM YEL        "
65 REM
66 REM RVS ON     "
67 REM
68 REM RVS OFF    "
```

NOTES

NOTES

RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

ISBN 0-8359-3671-6

DATAMOST

9748 Cozycroft Ave., Chatsworth, CA 91311. (213) 709-1202