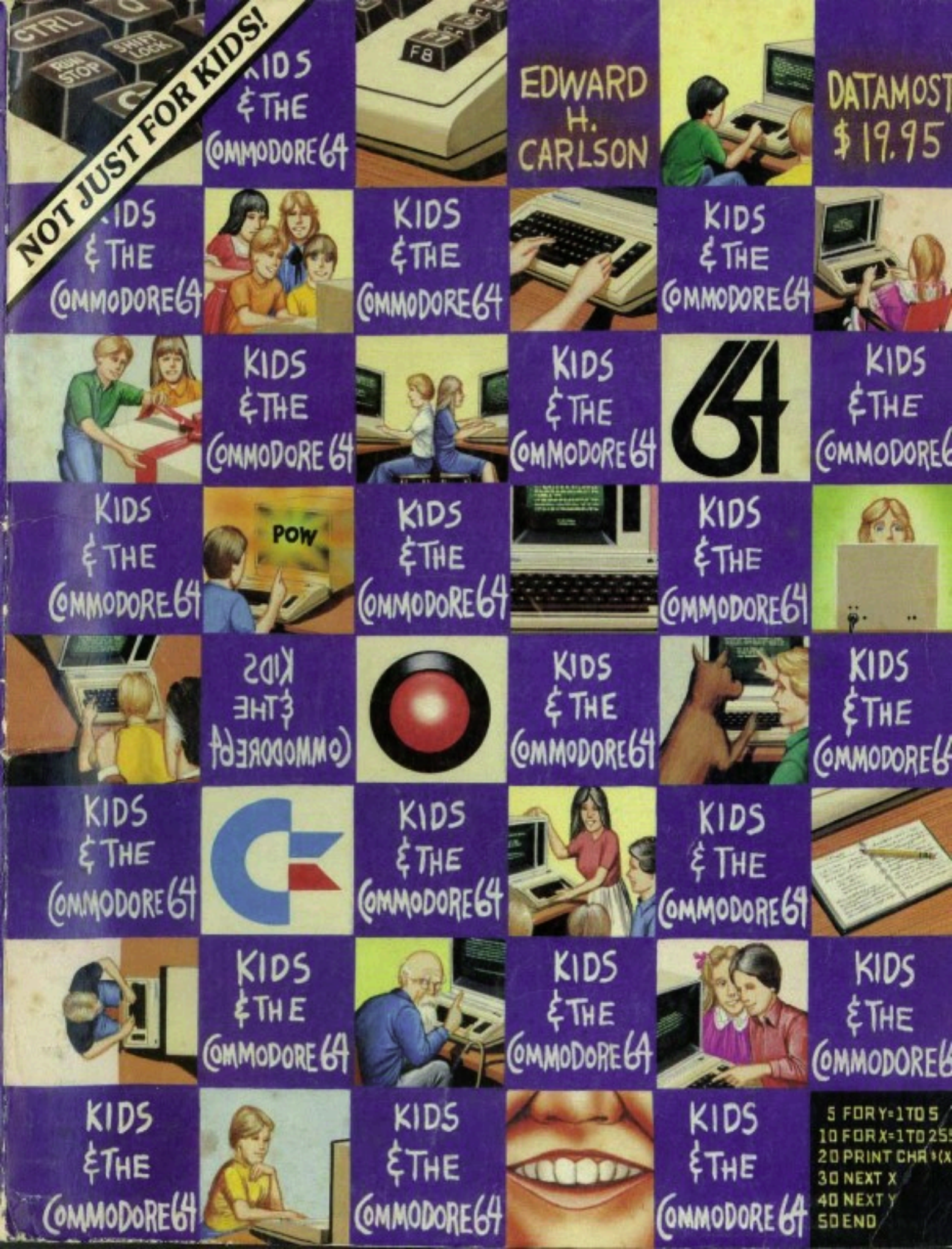

NOT JUST FOR KIDS!
KIDS & THE COMMODORE 64
EDWARD H. CARLSON
DATAMOST
$19.95
POW
64
5 FOR Y=1TO 5
10 FOR X=1TO 25
20 PRINT CHR
30 NEXT X
40 NEXT Y
50 END

KIDS AND THE COMMODORE 64

KIDS AND THE COMMODORE 64

by

Edward H. Carlson

Department of Physics and Astronomy
Michigan State University

Illustrated by

Paul D. Trap

ISBN 0-88190-172-5

DATAMOST™
8943 Fullbright Ave., Chatsworth, CA 91311-2750
(213) 709-1202

TABLE OF CONTENTS

INTRODUCTION

GRAPHICS, GAMES AND ALL THAT

ADVANCED PROGRAMMING

ACKNOWLEDGEMENTS

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

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

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

Mary Winter's vast knowledge of all things Commodore and skill in presenting computing topics to children has been especially helpful to me on many occasions.

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

It is a pleasure to acknowledge the contributions of Dave Gordon, a remarkable publisher and valued friend. I thank all of his staff at DATAMOST, especially Arlon Dorman, who guides the books to rapid completion and Marcia Carrozzo who guided each word and cartoon lovingly to its place on the pages of the last four books.

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

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

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

TO THE KIDS

This book teaches you how to write programs for the Commodore 64 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.

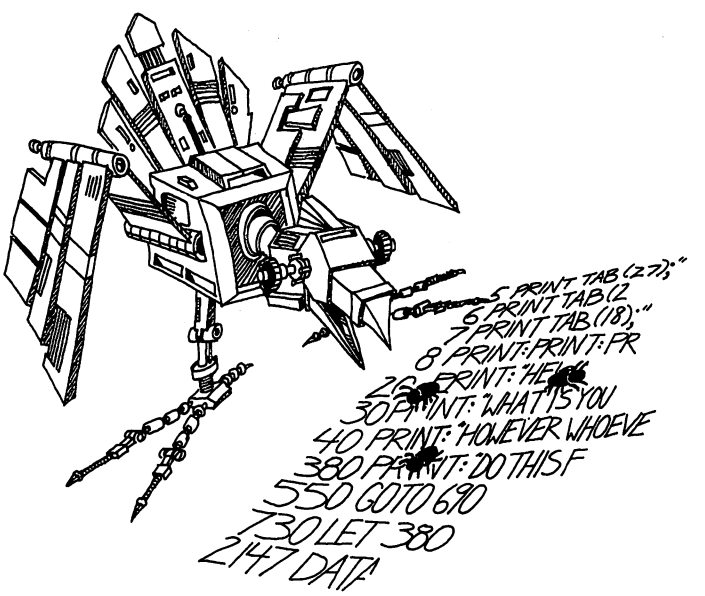
Perhaps your record collection or your paper route needs the organization your special programs can provide. If you are working on the school yearbook, maybe a program to handle the finances or records would be useful.

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

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

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



TO THE PARENTS

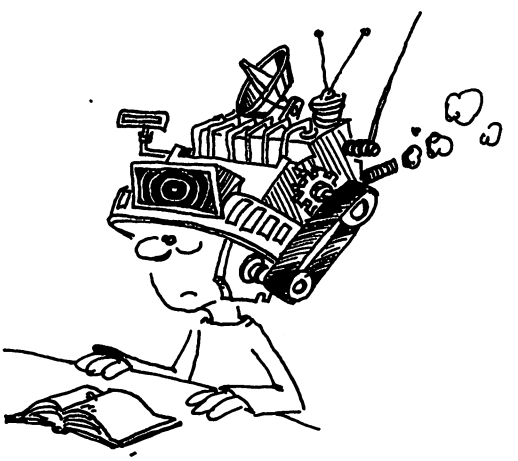
This book is designed to teach BASIC on the Commodore 64 to youngsters in the range from 10 to 14 years old. It gives guidance, explanations, exercises, reviews and "quizzes." Some exercises have room for the student to write in answers which you can check later. Answers are provided in the back of the book for program assignments.

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

Learning to program is not easy because it requires handling some sophisticated concepts. 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 which are possible once a repertoire of commands is built up.

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

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



TO THE TEACHER

This book is designed for students in about the 7th grade. It teaches BASIC and the features of the Commodore 64.

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

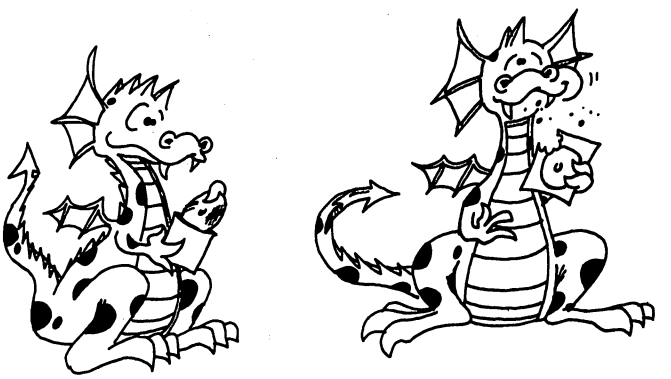
Like a block set which has many copies of a few types of blocks, BASIC uses a relatively small number of standard commands. Yet the blocks can be formed into a unique and imaginative castle and BASIC can be used to write an almost limitless variety of programs.

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

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

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

The 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

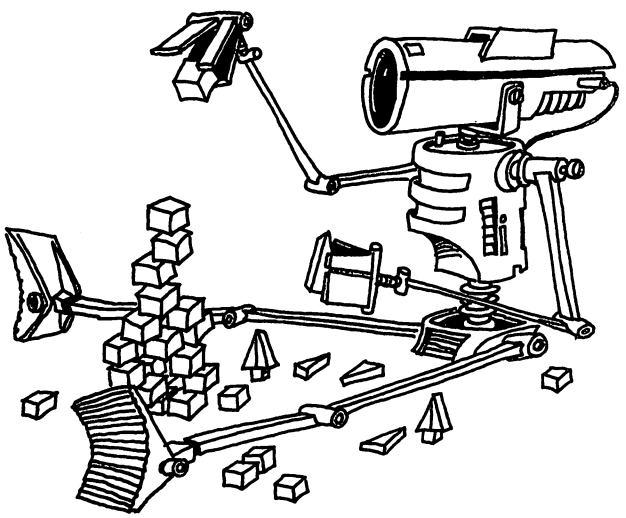
The book is arranged into 33 lessons, each with notes to the instructor and each containing assignments and review questions.

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 6, THE INPUT COMMAND, for an explanation. The central part of the book emphasizes more advanced and powerful commands. The final part of the book continues this, but also deals with broader aspects of the art of programming such as editing, debugging, and user friendly programming.

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

Lesson 14 SAVING TO TAPE, can be studied anytime after Lesson 3.



INSTRUCTOR NOTES 1 NEW, PRINT, REM AND RUN

This lesson is an introduction to the computer. There are many small questions your student may have at the start, so you should pull up a chair and help in the familiarization.

If something goes wrong and all else fails, tell your student to turn off the computer, then turn it on and start again.

The light blue writing on a dark blue background may be hard to read. Instructions are given for making white letters. If these are still hard to read, then the instructions to POKE 53281,0 to get a black background may be followed.

The contents of the lesson:

1. Turning on the computer.
2. Typing versus entering commands or lines. RETURN key.
3. The computer understands only a limited number of commands.
4. REM puts remarks in the program.
5. What is a program. Numbered lines.
6. Clearing the screen.
7. White letters on a black background.
8. Memory can be cleared with NEW.
9. 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.
10. RUN makes the computer go to memory, look at the commands in the lines (in order) and perform the commands.
11. One can skip numbers in choosing line numbers, and why one may want to do so.

QUESTIONS:

1. Write a program which 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 which prints HELLO.

6. Make it run.

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

8. How do you make the letters nice and white?

LESSON 1 NEW, PRINT, REM AND RUN

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 goes.

TYPING

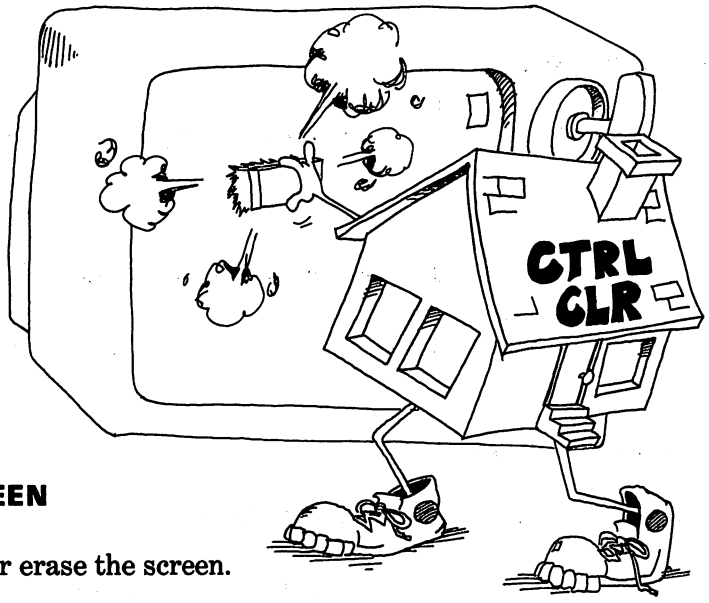
Type some things. What you type shows on the TV screen.

CURSOR, GO HOME!

The cursor's home is in the left hand corner of the screen at the top.

Find the CLR HOME key and press it. The cursor jumps to 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

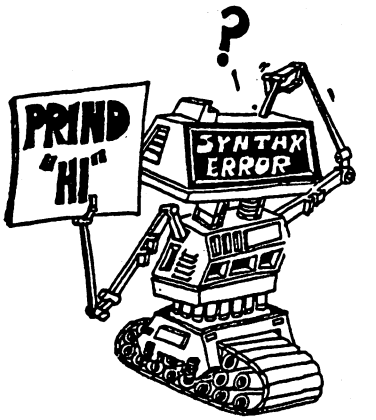
Try this. Type GIVE ME CANDY

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

Here are the first four words to learn:

NEW, PRINT, REM and RUN.

NICE WHITE LETTERS

If the letters on your TV screen are hard to read, try this:

Find the CTRL key.

Press it down and hold it down while you press the "2" key.

Now the letters you type will be pure white on a dark blue background.

NICE BLACK BACKGROUND

If it is still hard to read the letters on the TV, try this:

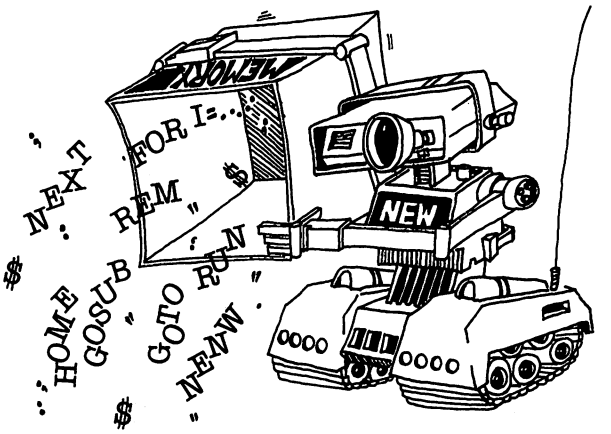
Type POKE 53281,Ø and press the RETURN key

The background on the TV screen turns black. Adjust the TV controls to make the writing on the screen easy to read.

THE NEW COMMAND

Type NEW and press RETURN.

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



HOW TO ENTER A LINE

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

1) Type a line
2) then press the RETURN key.

Clear the screen and enter this line:

10 PRINT "HI"

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

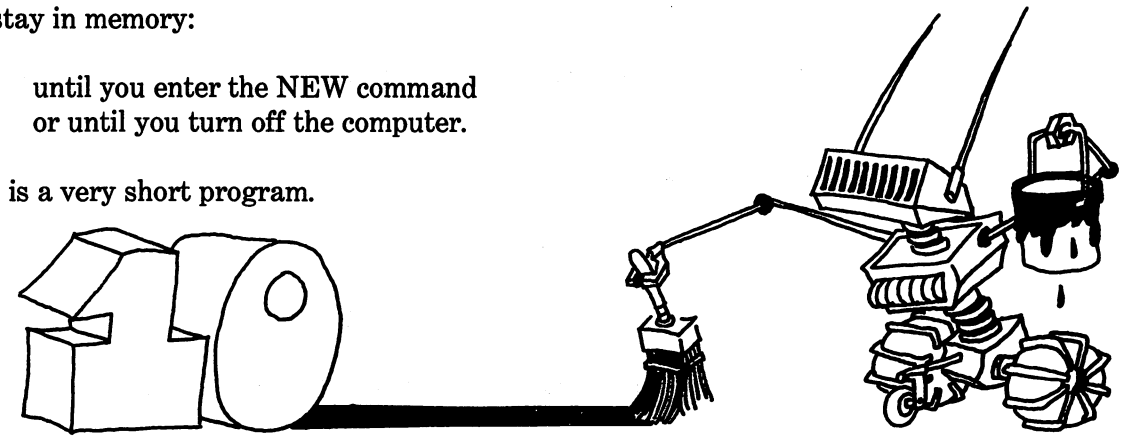
(Did you remember to press the RETURN key at the end of the line?)

Now line number 10 is in the computer's memory.

It will stay in memory:

until you enter the NEW command
or until you turn off the computer.

Line 10 is a very short program.



THE NUMBER ZERO AND THE LETTER "O"

The computer always writes the zero like this:

zero Ø

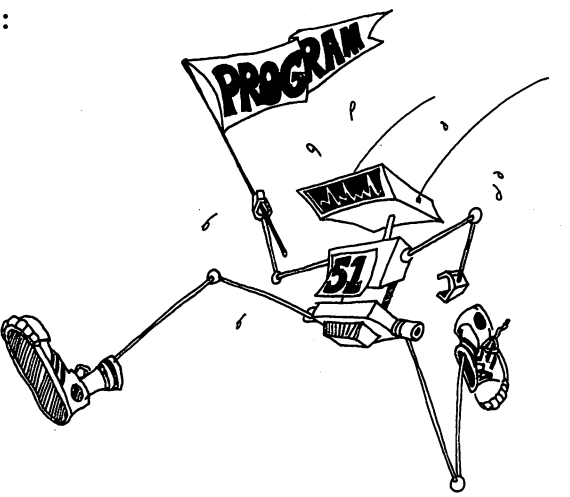
and the letter O like this:

letter O O

You have to be careful to do the same.

right 10 PRINT "HI"

wrong 1O PRINT "HI"



WHAT IS A PROGRAM?

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

HOW TO RUN A PROGRAM

A moment ago you put this program into memory:

10 PRINT "HI"

Now enter RUN

(Did you remember to press the RETURN key?)

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

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

HI

READY.

HOW TO NUMBER THE LINES IN A PROGRAM

Clear the screen.
Enter NEW and then this program:

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

This program has three lines. Each line starts with a command. You have already learned the PRINT command. We will tell you about the REM command in a minute.

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 HOME key to send the cursor home.
   Now use the same key (SHIFT CLR) to erase the screen.

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

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

INSTRUCTOR NOTES 2 COLOR AND THE KEYBOARD

The C-64 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 three functions, chosen by just pressing the key, or by pressing it while holding down the SHIFT, CTRL 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 C-64 some very powerful options, and several lessons in the book are devoted to exploring them.

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

If you choose white for the screen color and then white for the letter color, you will see nothing when you type! Try POKE 53281,15 for a grey screen which will show letters of any color obtained from the color keys.

Pressing the CTRL and RVS ON keys gives reversed characters. The reverse of a "space" is a colored block. A useful way to make color bars for adjusting the color TV is to do CTRL and RVS ON, then hold down the space bar to make a colored bar. Repeat for each of the colors you get from CTRL and a color key. Then adjust your TV for best color. Yellow and perhaps purple are the most sensitive to proper adjustment.

QUESTIONS:

1. How do you do each of these things:

   Make the computer type with red letters?
   Erase the screen?
   Empty the memory?
   Print your name?

2. How do you change the screen background color to white?

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

Turn your computer off, then on again. You are ready to start the lesson.

MAKING A WHITE BACKGROUND

Enter POKE 53281,1 The screen turns white.

CAREFUL! The number has to be exactly 53281. If you use another number by mistake, the computer may get "sick" and you will have to turn it off, then on, to make it well again.

MAKING RED LETTERS

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

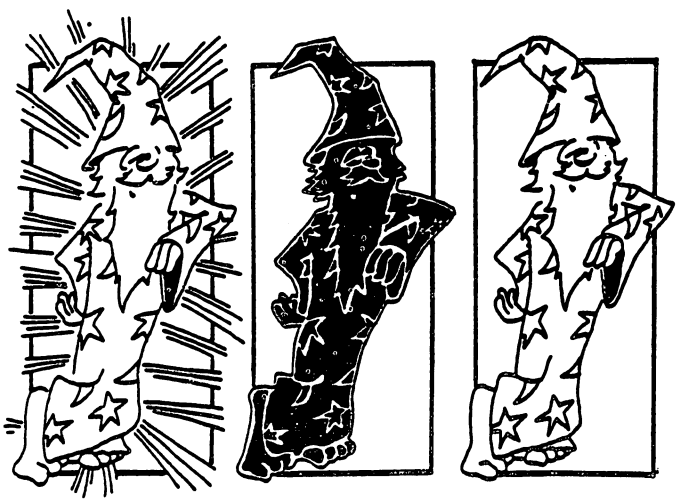
The character "3" in the middle.
The character "#" at the top.
The word "RED" on the front of the key.

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.

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

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



There are eight 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

Type letters in different colors. Hold down the CTRL key and press one of the color keys.

Now type. Try each color.

TUNE THE TV FOR GOOD COLORS

Tune the TV color controls so that the letters you typed are the right colors.

If you can make a good yellow, the rest of the colors are probably OK.

RAINBOW LETTERS

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

Enter

```
10 REM RAINBOW
20 PRINT "..."
30 PRINT "POT"
40 PRINT "..."
50 PRINT "OF GOLD"
```

but don't put dots in line 20 or line 40.

In line 20 press CTRL and the "BLK" key instead of dots.
In line 40 press CTRL and the "YEL" key instead of dots.

It will look like this on the screen:

```
20 PRINT " ■ "
40 PRINT " π "
```

Run the program. It should print "POT" in black letters and "OF GOLD" in yellow letters.

OH, OH! I'M IN TROUBLE, KIDS!

How can I tell you to press CTRL and a color key in a PRINT command?

I can't use just dots because you will not know what color I mean.

I know! I will use "little letters" for color keys.

Look: 20 PRINT "cyn"
means 20 PRINT " hold CTRL press CYN "
gives 20 PRINT " ◣" on the screen.

But 20 PRINT "CYN"

means type the three letters C, Y and N inside the quotation marks.

A SHORT CUT

Put the color key characters in the same PRINT statements as the words. The program looks like this:

in the book	on the screen
10 REM RAINBOW	10 REM RAINBOW
20 PRINT "grn POT"	20 PRINT " ▮POT"
30 PRINT "yel OF GOLD"	30 PRINT " π OF GOLD"

(Remember: "grn" means "hold down CTRL key and press GRN key.")

MORE RAINBOWS

Run:
```
10 REM LUCKY
20 PRINT"red HI"
30 PRINT"grn SALLY"  (Use your own name.)
```

Run:
```
10 REM LEPRECHAUN
30 PRINT"red P cyn I pur X grn I blu E yel S"
```

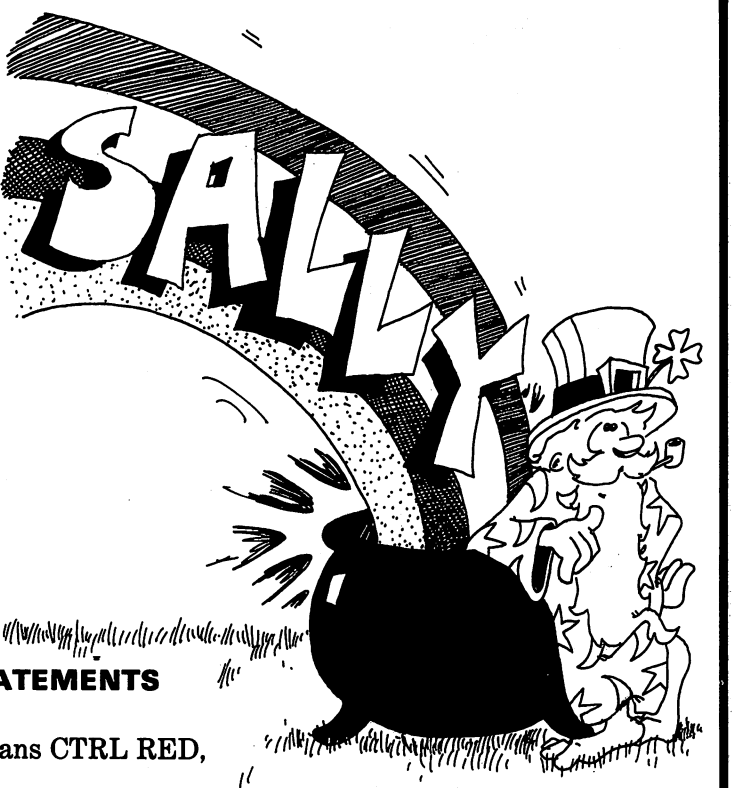
I put spaces in line 30 so you can read it easily. You should not put in the spaces. Without spaces it looks like this:

```
30 PRINT"redPcynIpurXgrnIbluEyelS"
```

Aren't you glad I put in the spaces?

How does line 30 look on the screen? _____

_____



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"
15 PRINT
20 PRINT "wht HI"
25 PRINT
30 PRINT "cyn HI AGAIN"
```

Lines 15 and 25 just print blank lines.

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

Assignment 2:

1. Write a program which 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
REM command
CLR key in a program
DEL key
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 which 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?

6. What happens if you hold down the DEL key?

7. How do you make a program clear the screen when it starts?

LESSON 3 LIST, BOXES IN MEMORY

Clear the screen and erase the memory.

(Start each lesson by clearing screen and memory.)

Now enter
```
10 REM HOUSE
20 PRINT"COME ON OVER"
```

Run this two line program. Then clear the screen.

The program is gone from the screen.

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

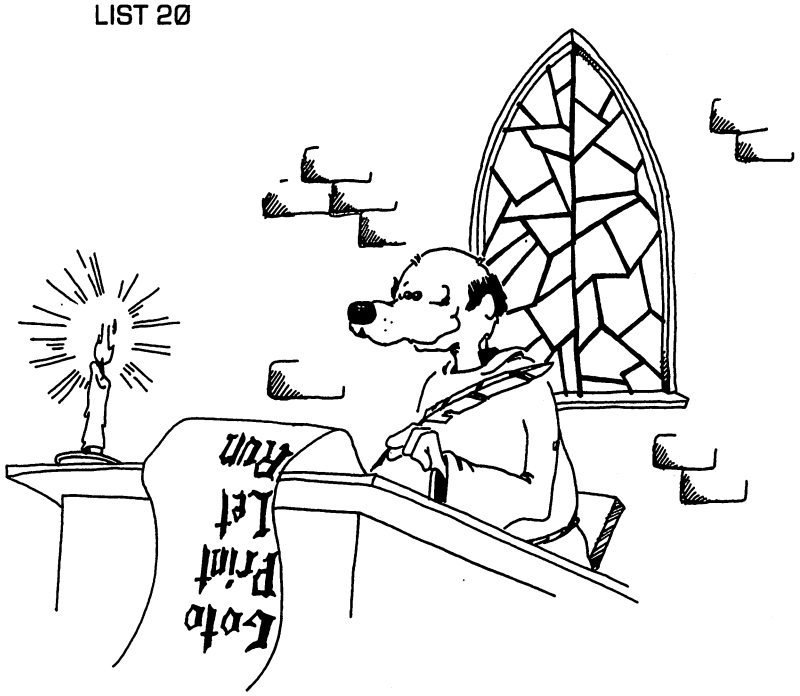
LISTING THE PROGRAM

Make the computer show you the whole program.

Enter LIST

To make the computer show you just one line of the program, enter LIST followed by the number of the line, like this:

```
LIST 20
```



THE MEMORY

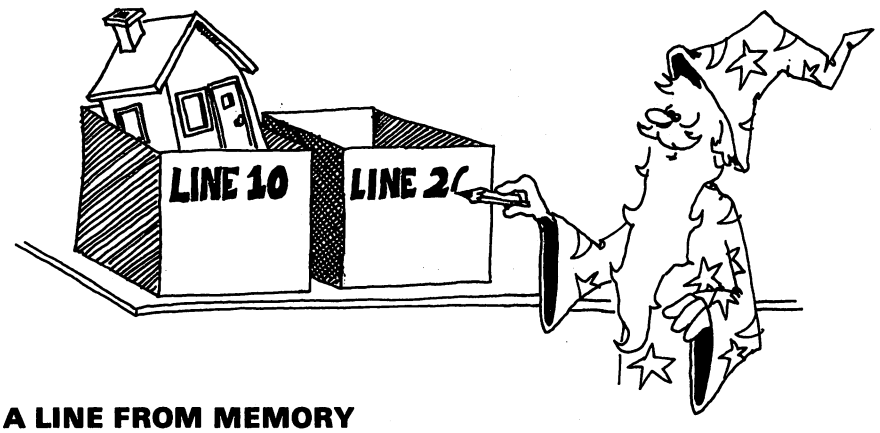
The computer's memory is like a shelf of boxes. There is a spot on the front of each box to write its name. At the start, all the boxes are empty and no box has a name.

When you entered 10 REM HOUSE

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

When you entered 20 PRINT"COME ON OVER"

the computer took the second box and wrote "Line 20" on its label. Then it put the statement PRINT "COME ON OVER" into the box and put that box into its place on the shelf.



ERASING A LINE FROM MEMORY

To erase one line of the program, enter the line number with nothing after it.

To erase line 20, enter 20

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

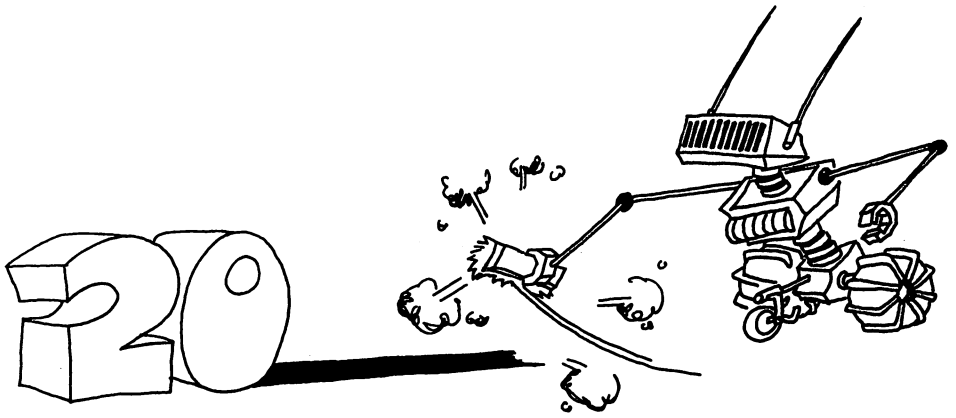
When you enter just a line number with nothing after it, the computer finds the box with that line number on it, empties the box and erases the name off the front of the box.

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 in your line. The computer puts it into the correct place.

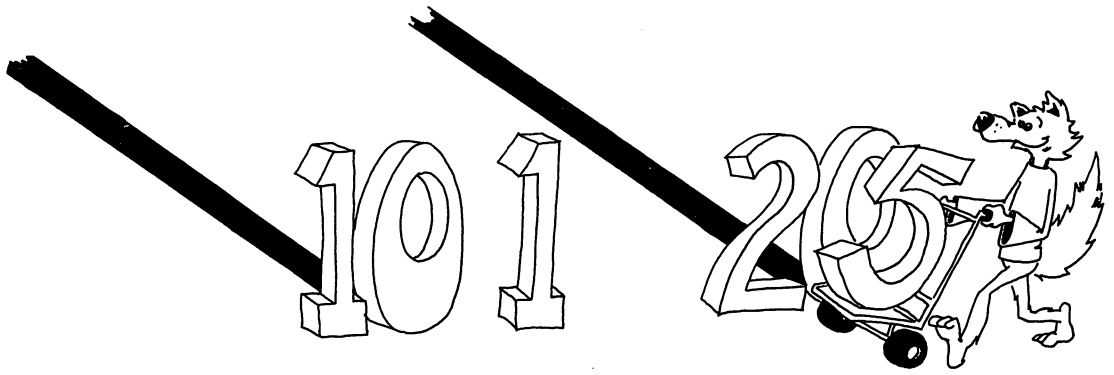
Enter NEW and this:

```
10 REM MORE AND MORE
20 PRINT "MORE LINES WANTED"
40 PRINT " NOW "
```

List it and run it. Now add this line:

```
15 PRINT " STILL"
```

List and run it again.

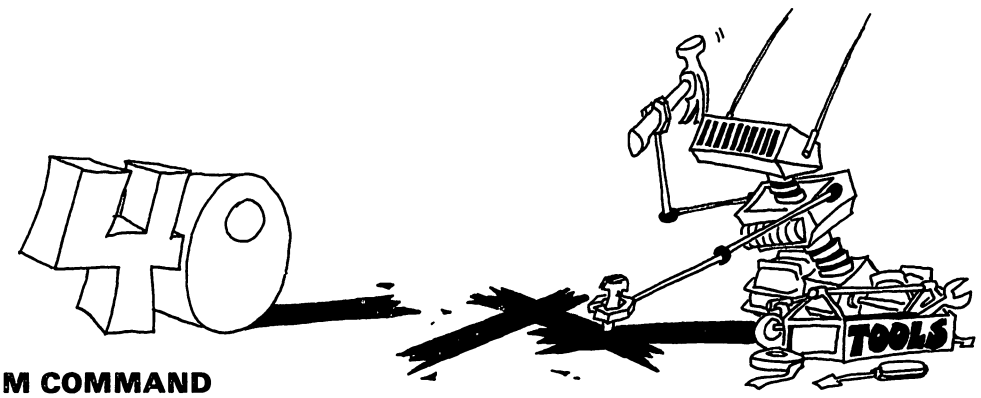


FIXING A LINE

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

```
40 PRINT "DOGIE"
```

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

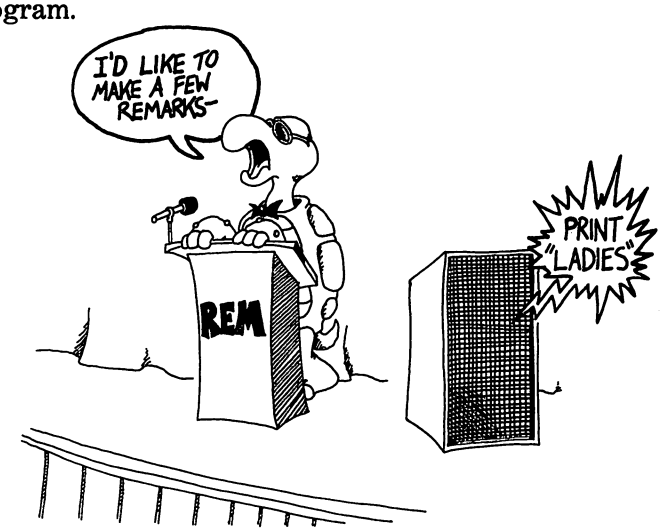


THE REM COMMAND

Enter NEW and this:

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

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



USING THE CLR KEY IN A PRINT COMMAND

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

Run:
```
10 REM CLASSY CAR
20 PRINT"clr cyn MERCEDES-BENZ"
```

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

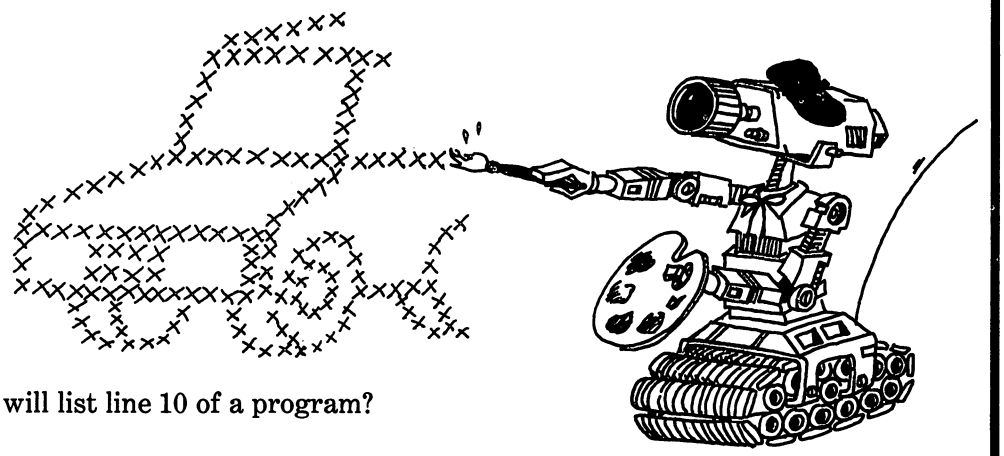
```
20 PRINT"■ MERCEDES-BENZ"
```

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 cyn MERCEDES-BENZ"
25 PRINT
30 PRINT " XXXXXX"
40 PRINT " XXXXXXXXXXXX"
50 PRINT "  O          O"
```

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



Assignment 3:

1. What command will list line 10 of 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 does the computer put into the "boxes" on its "shelf?"
5. Use REM and PRINT to draw three flying birds on the screen.

INSTRUCTOR NOTES 4 THE CURSOR KEYS AND DRAWING PICTURES

This lesson concerns the CRSR arrow keys, the graphics symbols on many keys and the COMMODORE 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.

For now we will not consider the insertion of characters, only their replacement by others or their deletion. When all is satisfactory, the line can be entered in the computer by pressing RETURN.

You can even change the line number. This allows you to move a line to another number. It also allows you to make copies of a line, either exact copies or slightly modified copies.

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.

The screen can be given any of 16 different colors by the POKE 53281,C command. C is a number from 0 to 15. The change screen color command can be given in a program line.

Be careful that you do not choose a screen color which is the same as the letter color! The writing would disappear until you changed one of the colors. In fact, this can be exploited in making "invisible writing." We will do this after the delay loop is introduced.

QUESTIONS:

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

2. Type the "solid ball" character on the screen. What keys do you need to use?

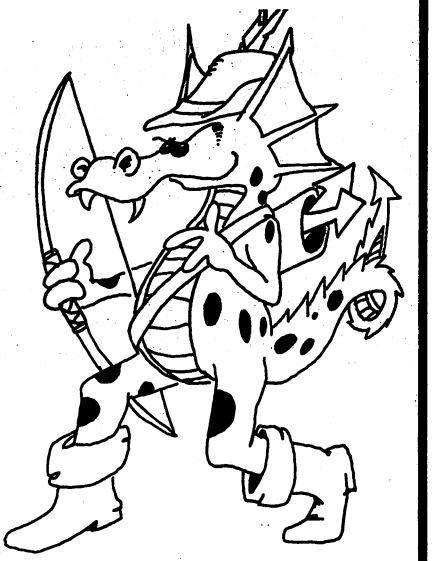
3. Type the triangle graphics character which appears on the "star" key. What keys do you need to use?

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. Now find the COMMODORE FLAG key. It is under the RUN STOP key and looks like a large "C" with a flag flying from its side.

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 FLAG key and press a graphics key.
> The character on the left will be put on the screen.

(You do not see the square which is on the key. You see just the character which is inside of the square.)

Do this: Use the CRSR keys to move the cursor to the screen center and draw a tiny red square. (Hint: use the "corner" characters which are on the O, P, @ and L keys.)

Now draw a large green square around the red square. (Hint: Use the "checker" character on the "plus" key.)

FIXING MESSED UP LINES

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

Enter 10 REM ZRAGON

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

Type a "D" instead.

Now the line is correct, reading

10 REM DRAGON

Press RETURN to store the correct line in memory.

THE DELETE KEY

The DEL key is your "eraser." (DEL is short for "delete.") Try this:

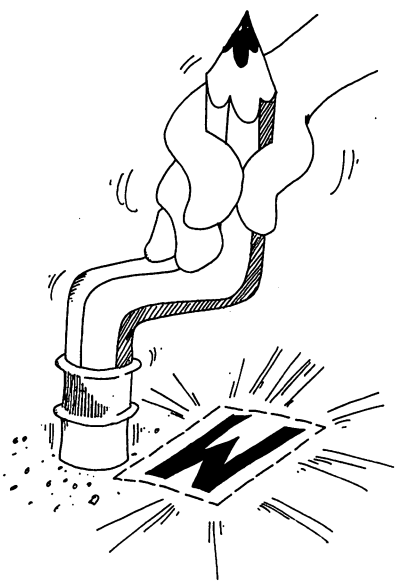
```
20 PRINT" HI THERW          (leave the cursor after the W)
```

Oops! The W should be an E.

You can erase the W by pushing the DEL key. Then type an "E".

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

RULE: The DEL key always erases the character to the left of the cursor.



SPEEDY ERASING

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

COPYCAT LINES

Enter

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

Run the program.

List the program. Then use the cursor arrow keys to move the cursor on the "2" of line 20. Type "30", then press the RETURN key.

Now RUN and LIST the program again. Line 20 has a twin line named "30".

COLORED SCREENS

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

Black: 10 POKE 53281,0

White: 10 POKE 53281,1

Choose another color by picking another number instead of 0 or 1. Any number from 0 to 15 can be used.

Enter
```
10 REM RED FLASH
20 PRINT "wht HI THERE"
30 POKE 53281,2
```

(Do you remember what to do when you see "wht" ?)

Assignment 4:

1. Try making different colored screens and different colors of letters on each screen. Which combinations look best?

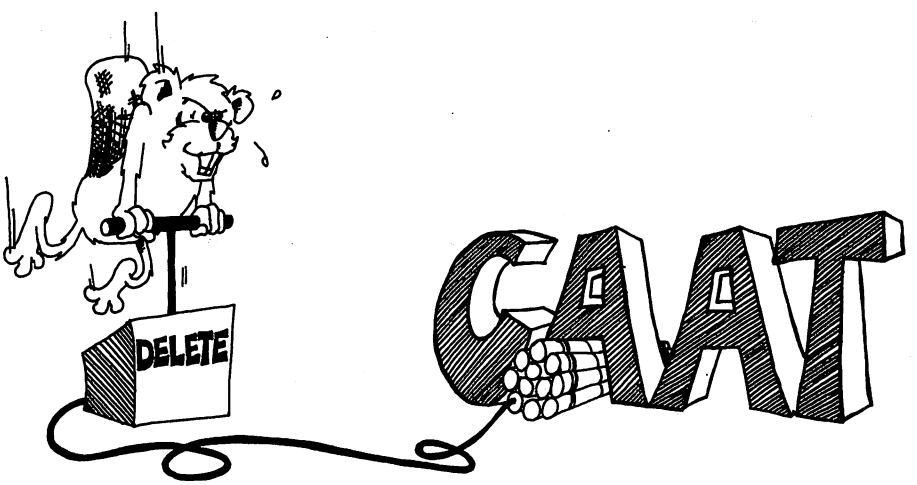
2. Practice using the DEL and the CRSR keys. Type and fix these lines:

   10 REM CAAAAAT (fix to "CAT")

   10 REM TTIIGGEEERRR (fix to "TIGER")

3. Draw a "smiley face" on a colored screen.

4. Draw a valentine. Use lots of different graphics symbols.



INSTRUCTOR NOTES 5 TRICKS WITH PRINT

In this lesson:

PRINT with a semicolon at the end
PRINT with semicolons between items
PRINT with commas between items
the "invisible" PRINT cursor
characters and string constants
review of keys

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 and also appears when executing the INPUT command.)

When a PRINT statement ends with a semicolon, the output cursor remains in place at the end of the last printed character. The next PRINT will start writing characters onto the end of the message printed by the current PRINT command.

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

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

If commas are used, the items will be printed in columns.

A series of printed strings will have their characters in contact. If spaces are desired, as in the "TOAST AND JAM" example, the spaces have to be put into 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 (two ways)?

LESSON 5 TRICKS WITH PRINT

ONE LINE OR MANY?

Enter this program:

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

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

Now enter

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

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

Run it.

What was different from the first time? _____

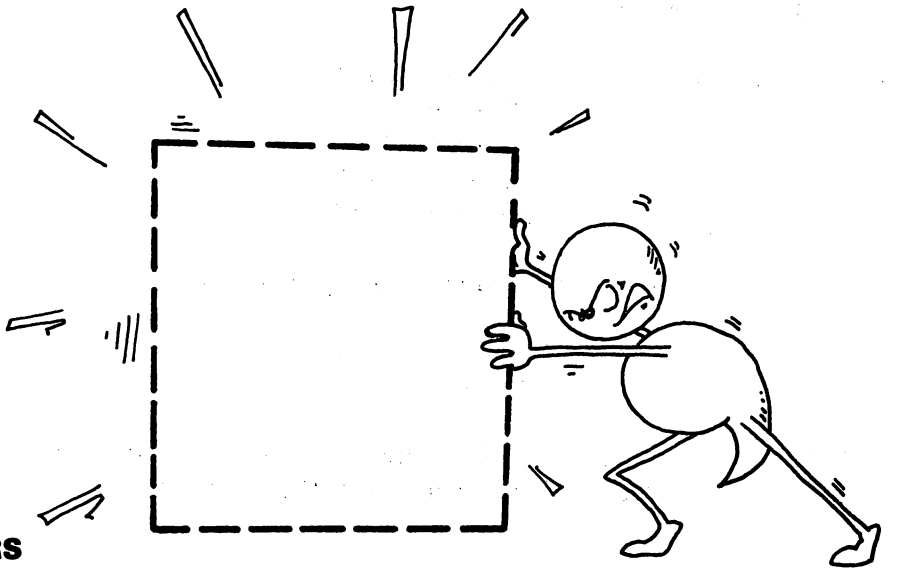
_____

THE HIDDEN CURSOR

Remember the flashing 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.) The INPUT command will be explained in the next lesson.

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

SQUASHED TOGETHER OR SPREAD OUT?

Enter NEW, then try this:

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

after you have run it, try also:

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

RULE: Putting a comma between things in PRINT puts spaces between them on the screen.

CHARACTERS

Look at these PRINT statements:

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

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

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

```
10 PRINT " "
```

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

STRING CONSTANTS

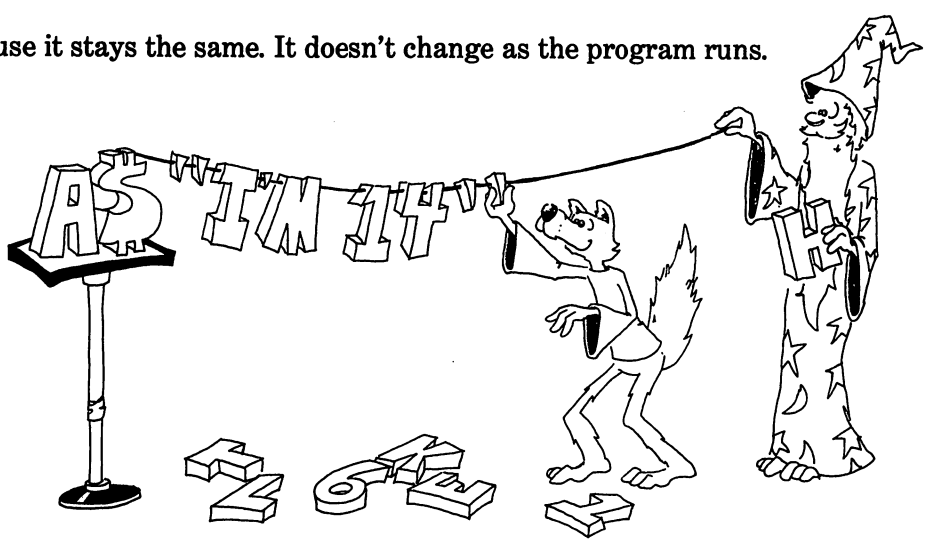
Characters in a row make a "string."

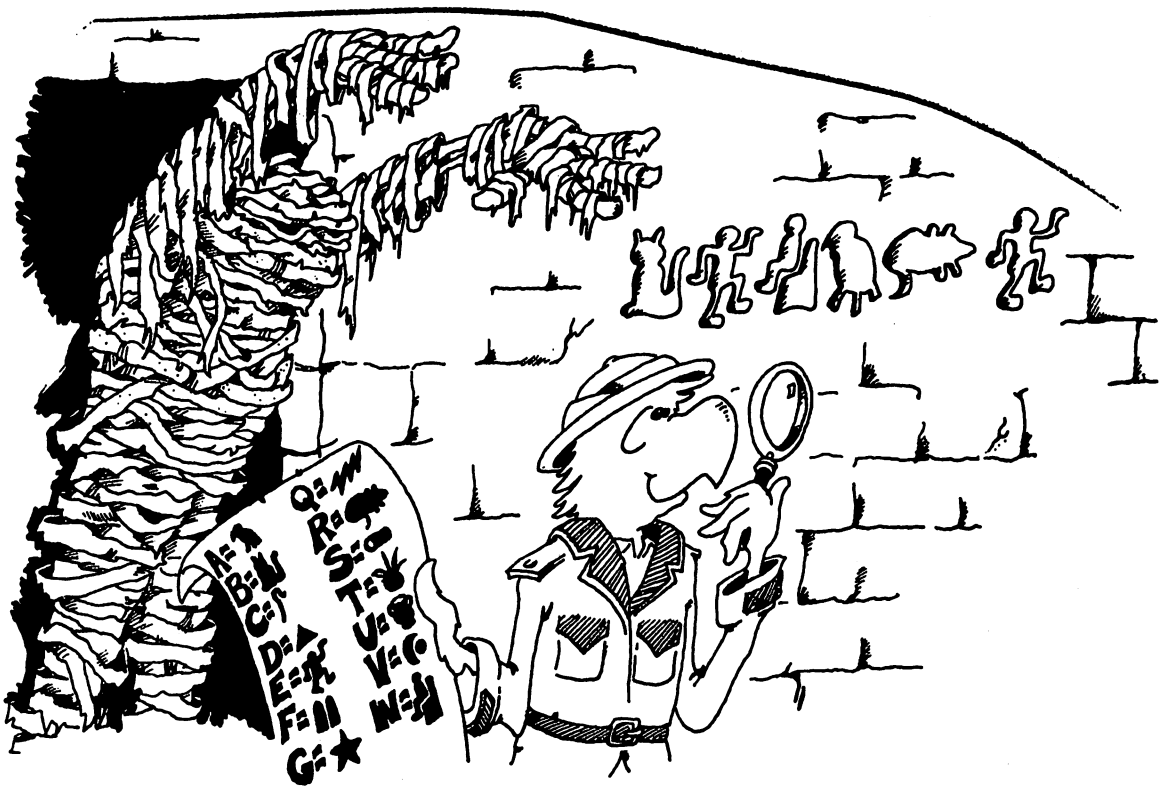
The letters are stretched out like beads on a string.

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.





Assignment 5:

1. Write a program which asks for the name of a musical group and one of their tunes. Then using just one PRINT command, print the group name and the tune name, with the word "plays" in between.

2. Do the same, but use three PRINT commands to print on one line.

3. Write a program which asks the user for three words. (Use three INPUT statements.) Then print them on one line with spaces between them. (Use PRINT with commas.)

INSTRUCTOR NOTES 6 THE INPUT COMMAND

This lesson concerns the INPUT command, the idea of a string variable and the difference between a PROGRAMMER and a USER.

In its simplest form, INPUT A$, there is no message in quotes in front. We want the student to concentate on the central feature of an INPUT.

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

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

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

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

The "two hats" of the student, programmer and user of the program, cause much confusion at assignment time. PRINT is the programmer speaking, while the user's comments are made only in response to an INPUT command and are stored in a string variable to be used or printed by the computer.

QUESTIONS:

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

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

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

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

5. Write a short program which uses REM, PRINT and INPUT.

6. Are you in trouble if the computer answers "EXTRA IGNORED" after an input? What made it do that?

LESSON 6 THE INPUT 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 enter "HI", the computer stores this word in a box named A$.

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

Run the program again and this time say something funny.

STRING VARIABLES

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

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

A$ is called a variable because you can put different strings into the box at different times in the program. The box can hold only one string at a time. Putting a new string into a box erases the old string which 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 into the string you type in answer to the computer.

If you accidentally 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.

YOU WEAR TWO HATS, USER AND PROGRAMMER

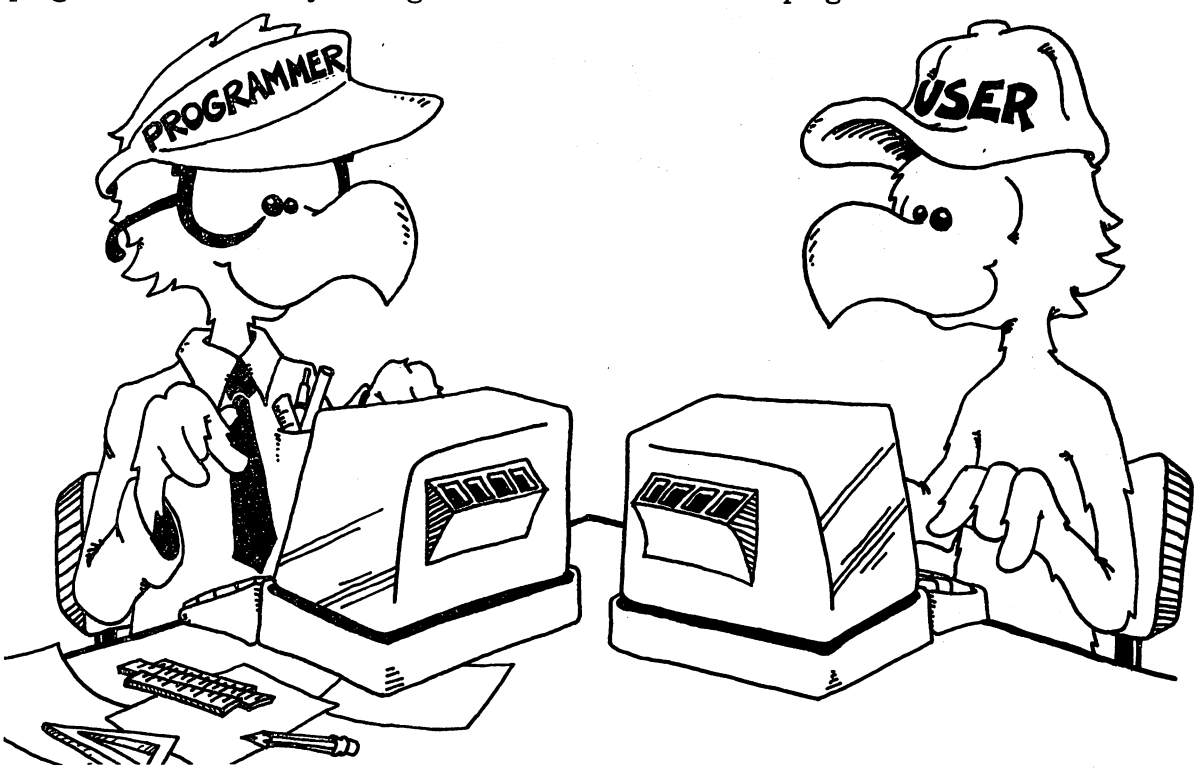
You are a PROGRAMMER when you write a program. The person who runs the program is a USER.

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

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

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

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



Assignment 6:

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

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

INSTRUCTOR NOTES 7 THE LET COMMAND, GLUING STRINGS

The LET command and concatenation are introduced.

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

Used so far:

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

Special keys discussed so far:

RETURN, CRSR arrows, SHIFT, CLR HOME, CTRL, DEL and the COMMODORE FLAG

QUESTIONS:

1. LET puts things into 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 LET 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$=H$+K$
```

LESSON 7 THE LET COMMAND, GLUING STRINGS

The LET command puts things into boxes. Enter and run

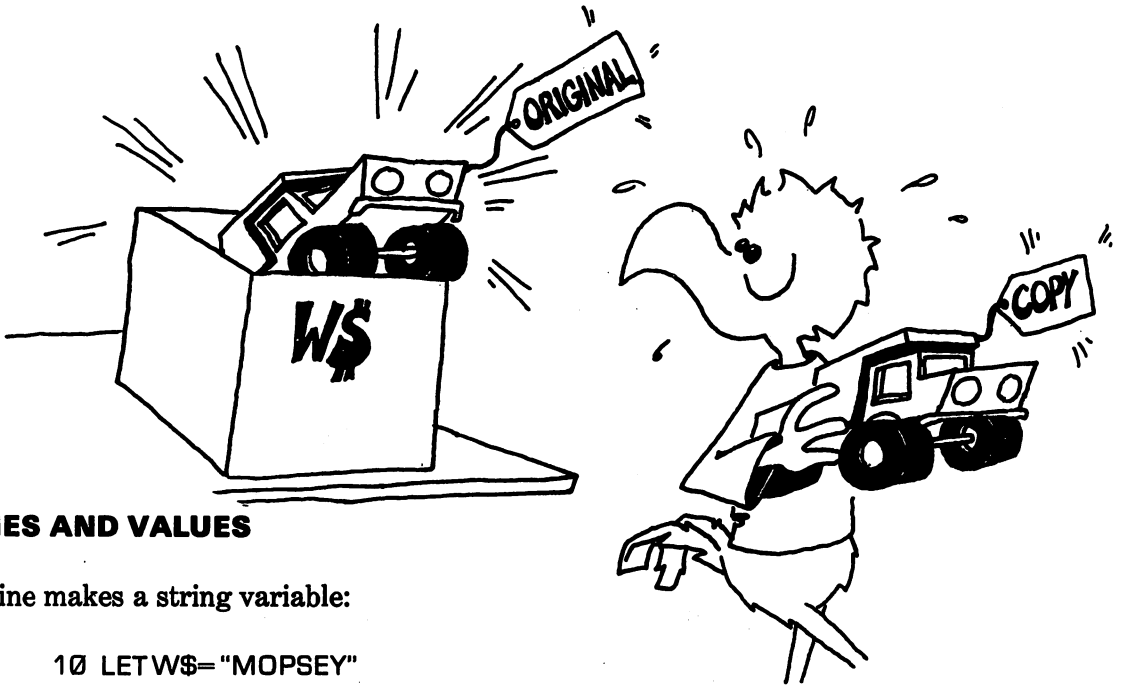
```
10 PRINT "clr"
20 LET W$="TRUCK"
40 PRINT W$
```

Here is what the computer does:

Line 10 The computer clears the screen.

Line 20 It sees that a box named "W$" is needed. It looks into its memory for it. It doesn't find one because "W$" 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 "TRUCK" into it.

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



NAMES AND VALUES

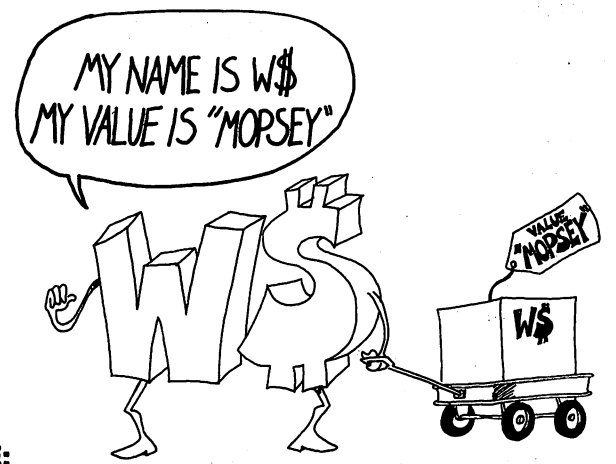
This line makes a string variable:

```
10 LET W$="MOPSEY"
```

The name of the variable is W$.

The value of the variable is put into the box.

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



ANOTHER EXAMPLE:

Enter and run

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

Explain what the computer does in each line.

10 _____

20 _____

30 _____

35 _____

40 _____

50 _____

GLUING THE STRINGS

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

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

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

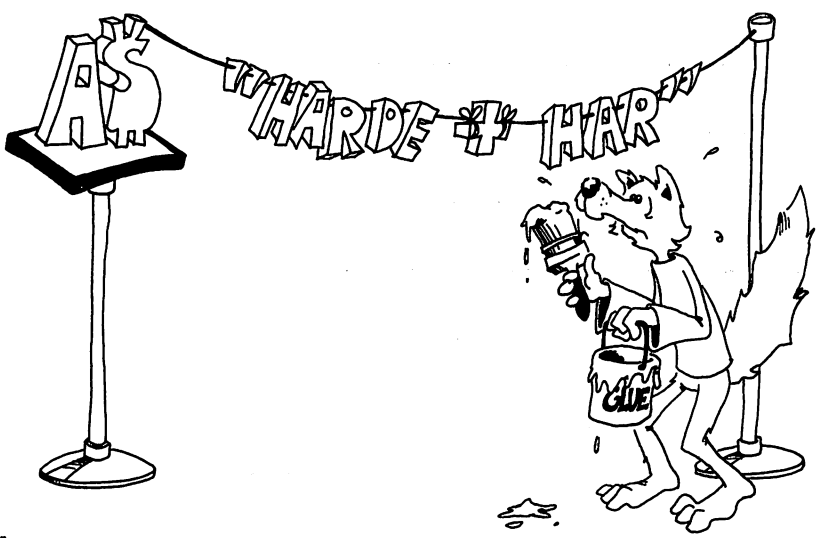
40 _____

70 _____

Now run the program to see if you were right.

Lines 30 and 60 glue strings together.

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



Assignment 7:

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

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

INSTRUCTOR NOTES 8 THE GOTO COMMAND AND THE STOP KEY

The GOTO command allows "dumb" loops which go 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. For cases where you have messed up the computer quite completely, pressing the STOP key may not in fact stop the program. For example, if the program reached an INPUT command (and shows the "?" and flashing cursor), pressing STOP does not stop the program. Try this. Hold STOP down and then press the RESTORE key once or twice.

GOTO allows the bad habit of "spaghetti" programming to grow. Examples of spaghetti are shown to the students. Although some fun is had with them, make the student conscious of the mess which undisciplined use of GOTO can make.

We now have three of the four major elements which 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 which 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 which 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:

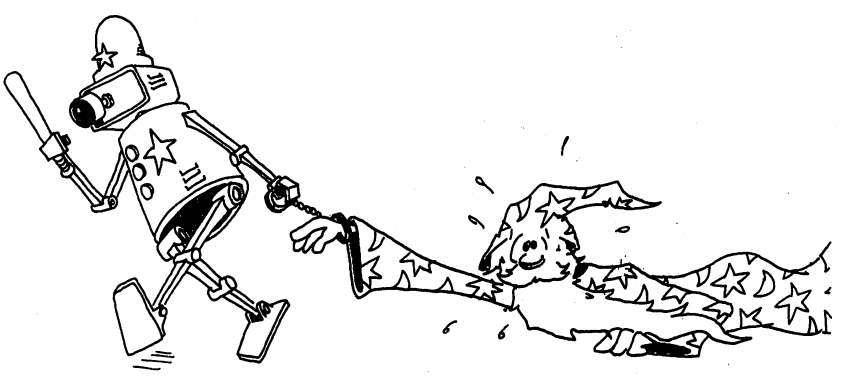
```
10 PRINT "clr"
20 PRINT "YOUR NAME?"
25 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 30
```

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.



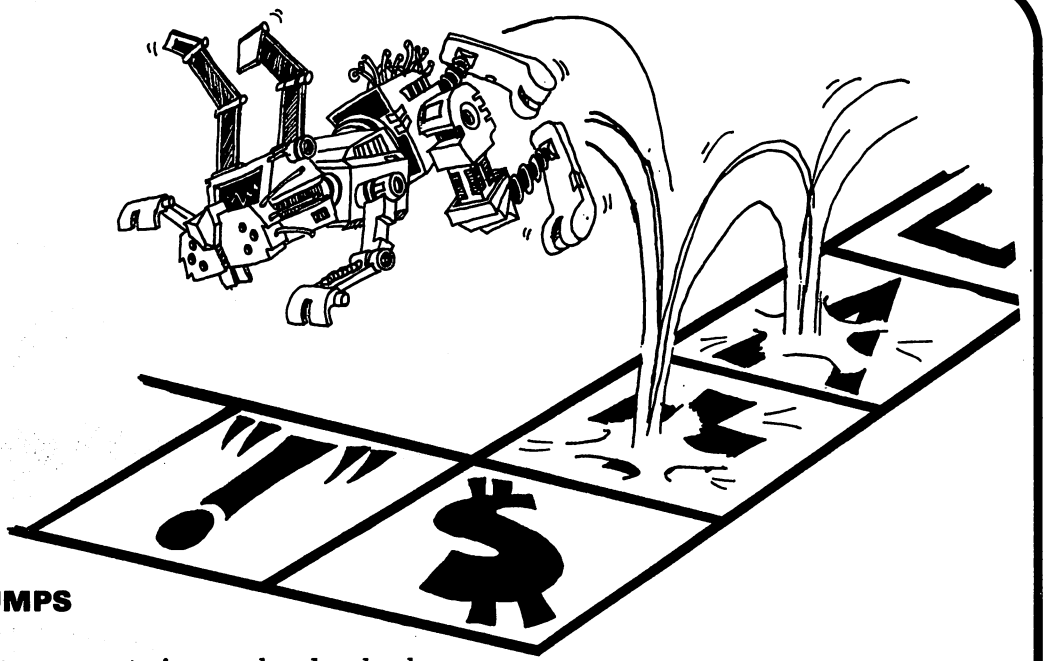
MORE JUMPING

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.



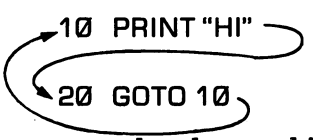
KINDS OF JUMPS

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:



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

Jumping ahead skips part of the program. Whatever for? We will see later in the IF command.

THE STOP KEY

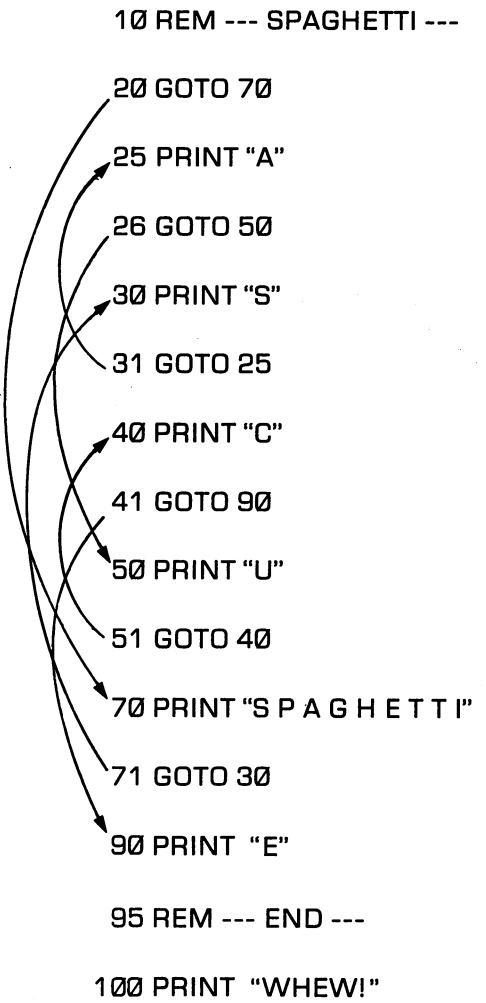
The STOP key is a "life saver." When you are in trouble, press STOP and the computer will stop running the program and wait for your next command. Your program is still safe in memory.

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.)

A CAN OF SPAGHETTI

Look at this:



This is not a good, clear program!

It is a "spaghetti" program.

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

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

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

20 GOTO 40

30 PRINT "N"

31 GOTO 60

40 PRINT "S"

41 GOTO 30

50 PRINT "E"

51 GOTO 99

60 PRINT "A"

90 PRINT "K"

91 GOTO 50

99 PRINT "B I T E"
```

2. Write a program which prints "TEEN POWER" over and over.

3. How do you stop your program?

4. Write another which 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.

5. Write a program which uses each of these commands: PRINT, INPUT, LET, GOTO. It also should glue two strings together and use two colors of letters.

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 which 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 to both 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.

"Phrase A" is used for the assertion being tested for truth. The phrase "command C" is used for the command to be done if the assertion is true.

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

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

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

Some care may be needed to separate and clarify these notions. When you see "A = B" it may not REALLY be true that A is equal to B because the assertion may actually be FALSE.

The larger set of relations:

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

will be treated in later lessons.

QUESTIONS:

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

```
15 PRINT "DOES YOUR TOE HURT?"
17 INPUT T$
20 IF T$="NAH" THEN GOTO 90
40 IF T$="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!"

3. What do we mean by "phrase A"?

4. What do we mean by "command C"?

5. Where is the "fork in the road" in an "IF" statement?

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 phrase A THEN command C
```

First the computer looks at "phrase A".

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

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

It looks like this:

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

or

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

Assignment 9A:

1. Clear memory and write a program which asks if the user is a "BOY" or "GIRL". If the answer is "BOY", the program prints "SNIPS AND SNAILS". If the answer is "GIRL", print "SUGAR AND SPICE".

THE "IF" IN ENGLISH AND IN BASIC

In English:

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

In BASIC:

```
40 IF      A$="DONE"      THEN PRINT "EAT SOME CAKE"
```

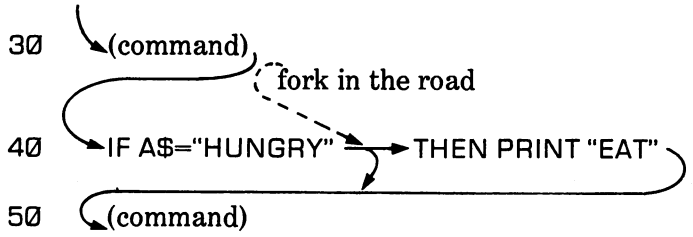
A FORK IN THE ROAD

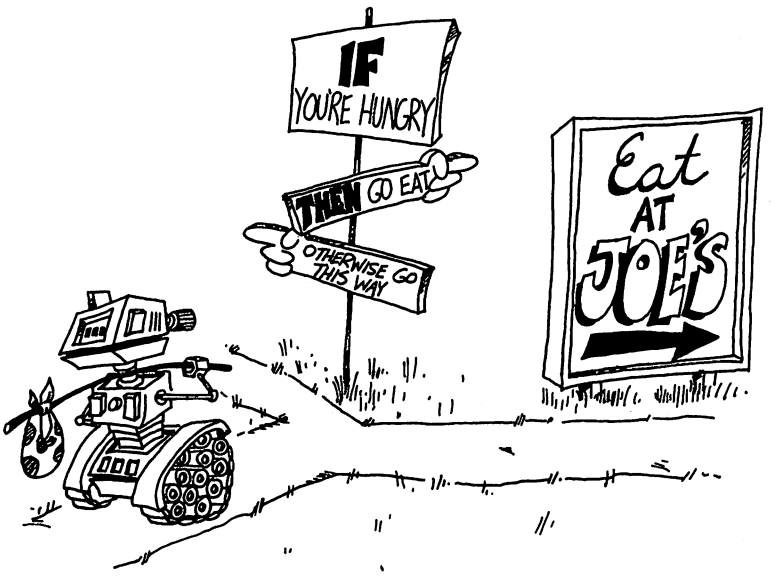
When it sees "IF", the computer must choose which road to take.

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

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

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





THE "NOT EQUAL" SIGN

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

To make the "< >" sign:

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

USING THE < > SIGN

```
40 IF phrase A THEN command C
```

"Phrase A" is a phrase which is TRUE or FALSE.

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

Put it into an IF command:

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

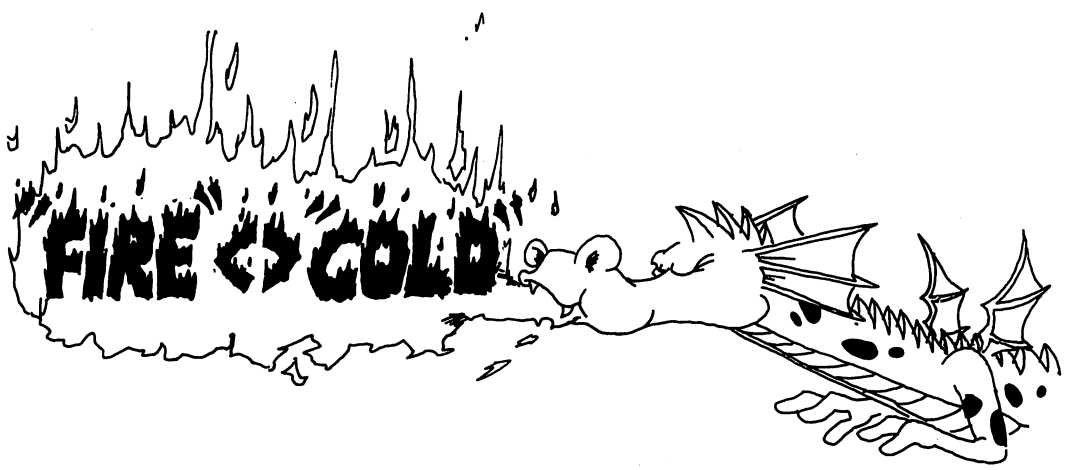
If the B$ box contains "COLD"
then B$ is not equal to "FIRE"
and the expression B$< >"FIRE" is TRUE.

The computer will print "FEED HIM SOME HOT CHILI".

Or if the B$ box contains "FIRE"
then the phrase B$<>"FIRE" is FALSE
and the computer will not print anything.

Here is how it looks in a program:

```
10 PRINT "WITH DOGS IT'S A COLD NOSE"
11 PRINT
20 PRINT "WITH DRAGONS, IT'S..."
21 PRINT
25 PRINT "HOW IS YOUR DRAGON'S BREATH?"
26 PRINT
28 PRINT "(ENTER 'FIRE' OR 'COLD')"
29 PRINT
30 INPUT B$
40 IF B$< >"FIRE" THEN PRINT "FEED HIM SOME HOT CHILI"
50 IF B$ = "FIRE" THEN PRINT "WATCH OUT!"
60 PRINT "NICE DRAGON!"
```



Assignment 9B:

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

2. Write a color guessing game. One player INPUTs a color in string C$ and the other keeps INPUTing guesses into string G$. Use two IF lines, one with a "phrase A"

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

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

INSTRUCTOR NOTES 10 INTRODUCING NUMBERS

Numeric variables and operations are introduced. The LET, INPUT and PRINT commands are revisited. The idea of memory as a shelf of "boxes" is extended to numbers. Again, variable names are limited to one letter for the time being.

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

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

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

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

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

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

Another idea from arithmetic which doesn't work in a LET is shown below.

Arithmetic: N = 3 means the same as 3 = N

BASIC: LET N = 3 is OK LET 3 = N is not

QUESTIONS:

1. Name the three kinds of "boxes" in memory. (That is, named by the kinds of things stored in the boxes.)

2. Explain why "N=N+1" for a computer is not like "5=5+1" in arithmetic.

3. Give another example of "bad arithmetic" in a LET command. Use the * or / symbols.

4. What does the computer mean by "TYPE MISMATCH ERROR"?

5. Give an example of a program line which would have a TYPE MISMATCH ERROR.

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

LESSON 10 INTRODUCING NUMBERS

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
```

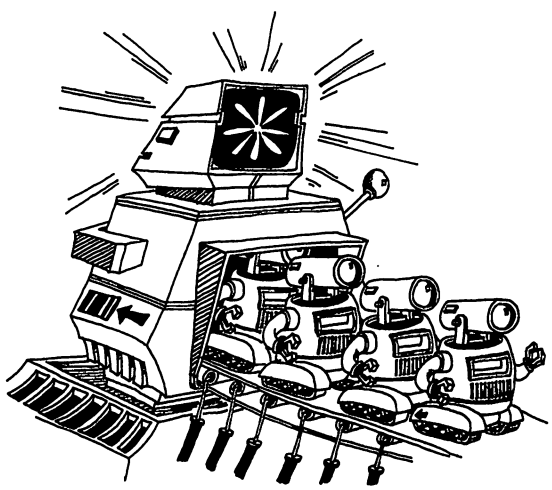
ARITHMETIC

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.

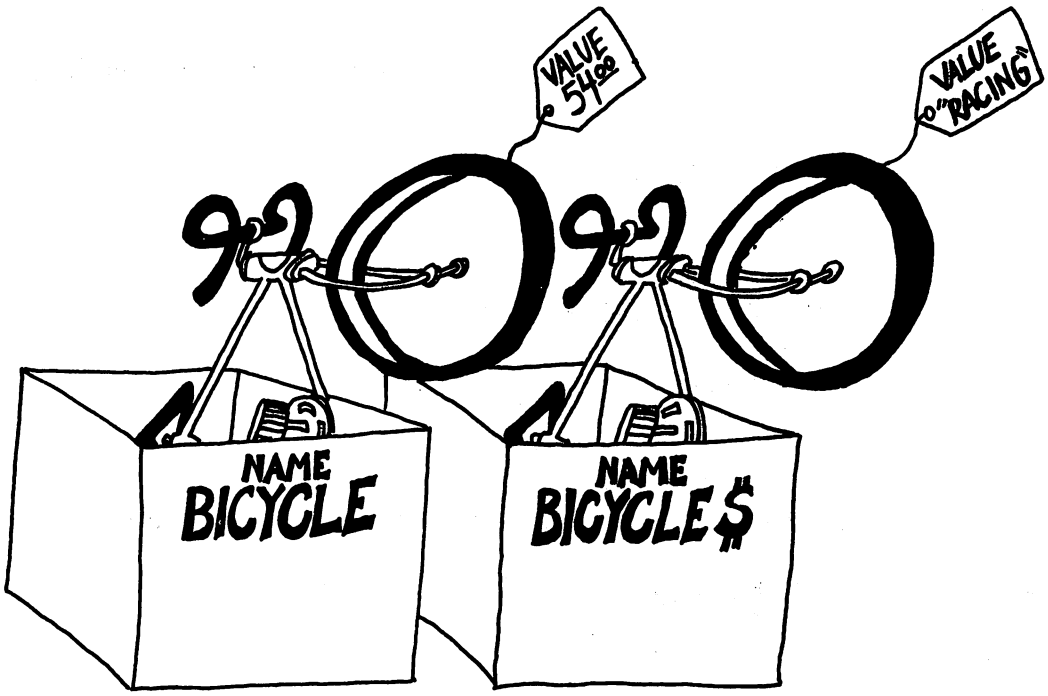


VARIABLES

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

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

The thing which is put into the box is called the "value" of the variable.



ARITHMETIC IN THE LET COMMAND

```
10 LET A=2001
20 LET B=1983
30 LET C=A-B
40 PRINT "HOW MUCH LONGER, HAL?"
50 PRINT C;" YEARS"
```

CAREFUL!

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

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

Some numeric constants: 5, 22, 3.14, –50

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

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

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

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

TYPE MISMATCH ERROR IN 10

There are two types of variables: number and string.

You cannot put a number into a string box or a string into a number box.

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

Lines 10 and 20 are OK, line 30 is wrong. What will the computer do when you run this little program? _____

Try to guess what each of these statements will print, then enter the line to see what happens:

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

MIXTURES IN PRINT

You can print numbers and strings in the same PRINT 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

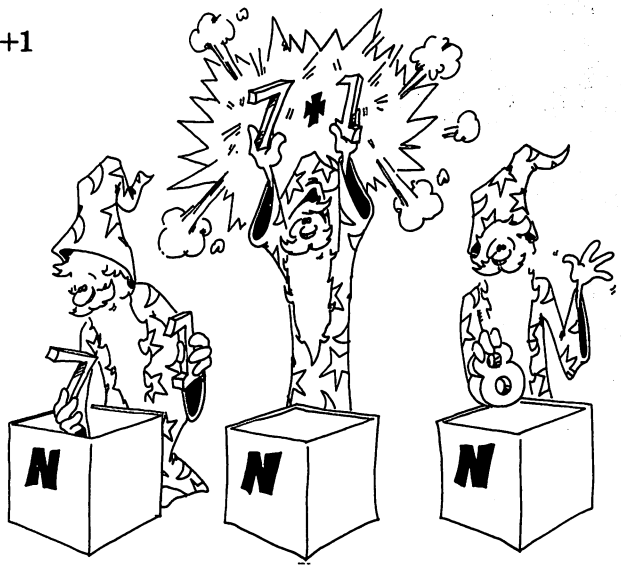
The "=" sign in computing does not mean "equals" exactly. Look at this program:

`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 into the box.

Another way to say the same thing is:

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

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

DON'T BE BACKWARD!

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

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

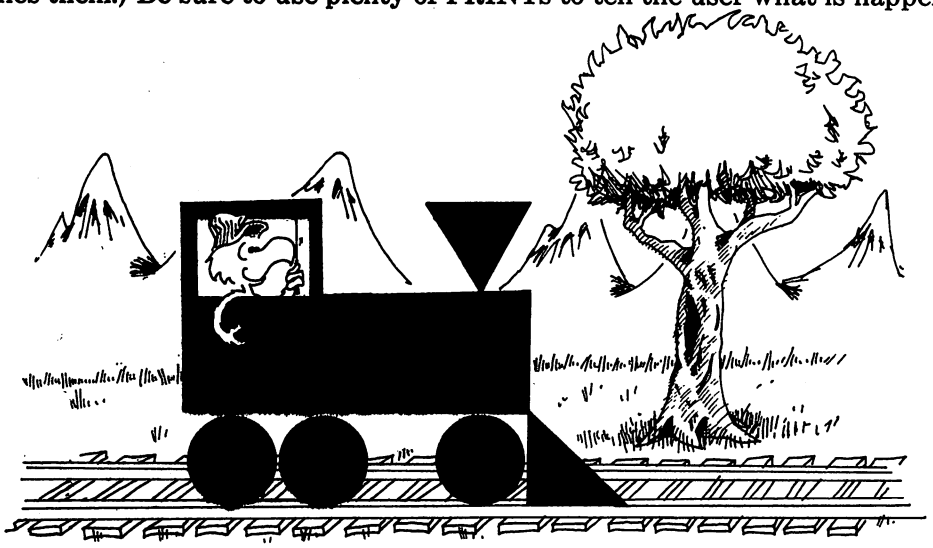
LET N = B means _____

LET B = N means _____

Assignment 10:

1. Write a program which 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 which asks for two numbers and then prints out their product. (Multiplies them.) Be sure to use plenty of PRINTs to tell the user what is happening.



INSTRUCTOR NOTES 11 TAB AND DELAY LOOPS

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 which 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 C-64 has a more general and powerful way of moving the cursor around. You simply put CRSR arrows in quotes into a PRINT command. This will be illustrated in later lessons.

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 one 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 which lasts for about two seconds.

2. Will this work for a delay loop?

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

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

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

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

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

LESSON 11 TAB AND DELAY LOOPS

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.

Try this:

```
10 PRINT "123456789ABCDEF"
30 PRINT TAB(3);"Y";TAB(8);"Z"
```

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

CAREFUL!

Run this:

```
10 TAB(5)
```

You see SYNTAX ERROR IN 10. TAB( ) has to be in a PRINT 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 move the printing only 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
```

HOW BIG A SPACE CAN TAB( ) MAKE?

There are 40 spaces across the screen. You can use any number 0 through 39 inside the TAB( ) parentheses. Larger numbers make the computer skip lines. Numbers larger than 255 will give an error message when the program runs:

ILLEGAL QUANTITY ERROR IN XX.

where XX is the line number.

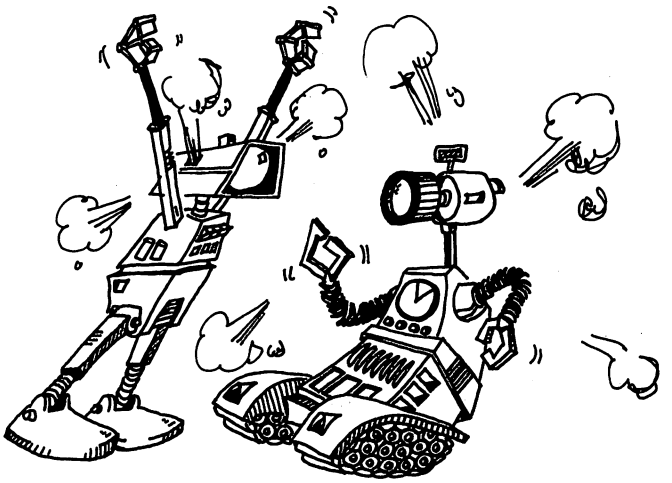
You can use TAB with strings too.

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 which 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:

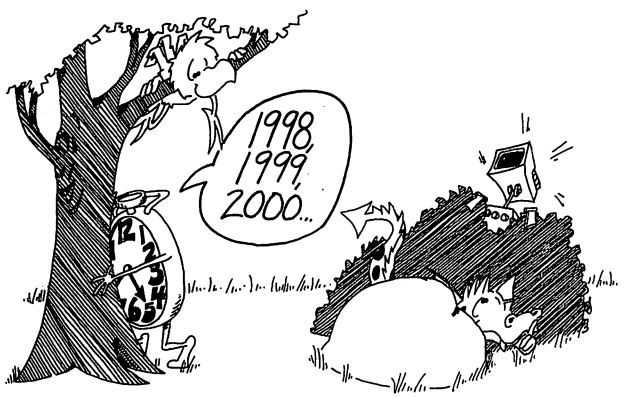
1. Write a program which asks for last names and nicknames. Then print the nickname starting at column 3 and the last name at column 15. Use a GOTO so the program is ready for another name-nickname pair.

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.

DELAY LOOPS

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

Run this program:
```
10 REM HIDE AND SEEK
20 PRINT "clr"
30 PRINT "HIDE!"
40 FOR I=1 TO 2000:NEXT I
50 PRINT "COMING READY OR NOT!"
```
Line 40 is the delay loop. The computer counts from 1 to 2000 before going on to the next line. It is like counting when you are "it" in a game of hide and seek.



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

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

```
10 REM -- TICK TOCK --
20 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 which prints out a four word message with several seconds between each word.

2. Write a digital clock program. It uses a timing loop to count seconds. Input the present time in hours, minutes and seconds. The clock then counts seconds and prints them out. When 60 seconds have gone by, add one to the minutes and put seconds back to zero. Same with hours. Run the clock a long time and adjust the timing loop so the clock keeps good time.

INSTRUCTOR NOTES 12 THE IF COMMAND WITH NUMBERS

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

= > < <>

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

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

A "home made" loop is demonstrated in the GUESSING GAME, but not discussed. The loop starts in line 50 and goes to 80. The exit test is made in line 57. The logic of this loop is that of a DO WHILE.

QUESTIONS:

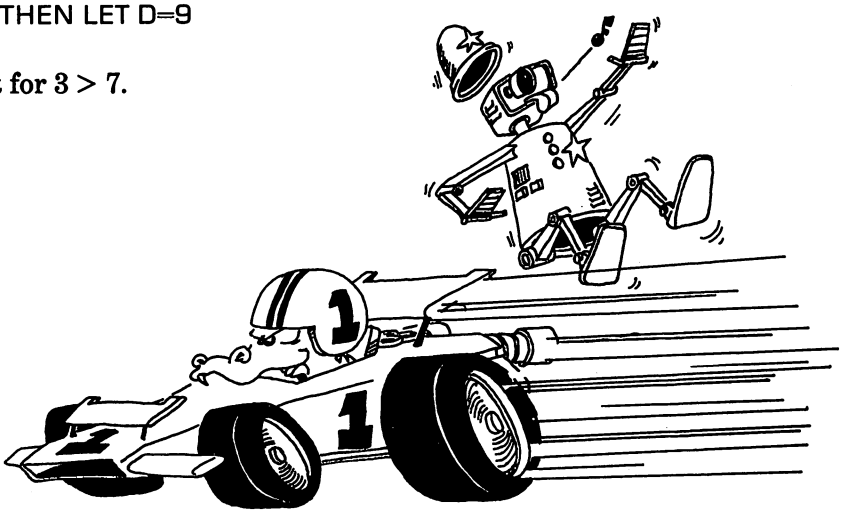
1. What part of the IF command can be TRUE or FALSE?

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 you used before with strings. Again we have

10 IF phrase A is true THEN do command C

"Phrase A" can have these arithmetic symbols:

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

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

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

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

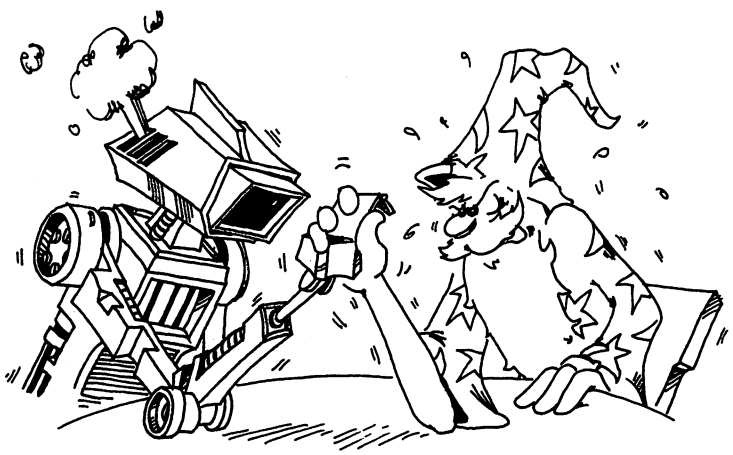
PRACTICE

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

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

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

A=C T F
B<>C T F



AN IF INSIDE AN IF

The "teenager" program above is missing something. Add

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

To understand this, break it into two parts:

60 IF A>12 THEN (command C) where

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

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

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

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

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

Assignment 12A:

1. Draw the "fork in the road" diagram for line 60 above. There will be two forks on the diagram. (See page 56.)

GUESSING GAME

```
10 REM GUESSING GAME
15 POKE 53281,0
20 PRINT "clr TWO PLAYER GAME"
30 PRINT "dn cyn FIRST PLAYER HIDE YOUR EYES!"
32 PRINT "dn grn SECOND PLAYER:"
34 PRINT " ENTER A NUMBER FROM 1 TO 100 dn"
40 INPUT N
45 PRINT "clr"
50 PRINT "dn yel MAKE A GUESS"
55 INPUT G
57 IF G=N THEN GOTO 90
60 IF G<N THEN PRINT "TOO SMALL"
65 IF G>N THEN PRINT "TOO BIG"
80 GOTO 50
90 REM GAME OVER
95 PRINT "clr red dn dn dn THAT'S IT! wht"
```

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 57, then you skip to line 90 and print "THAT'S IT!"

Assignment 12B:

1. What happens in each line if G is 31 and N is 88:

50 _____

55 _____

57 _____

60 _____

65 _____

80 _____

What happens if G is 88 and N is 88:

50 _____

55 _____

57 _____

60 _____

65 _____

80 _____

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

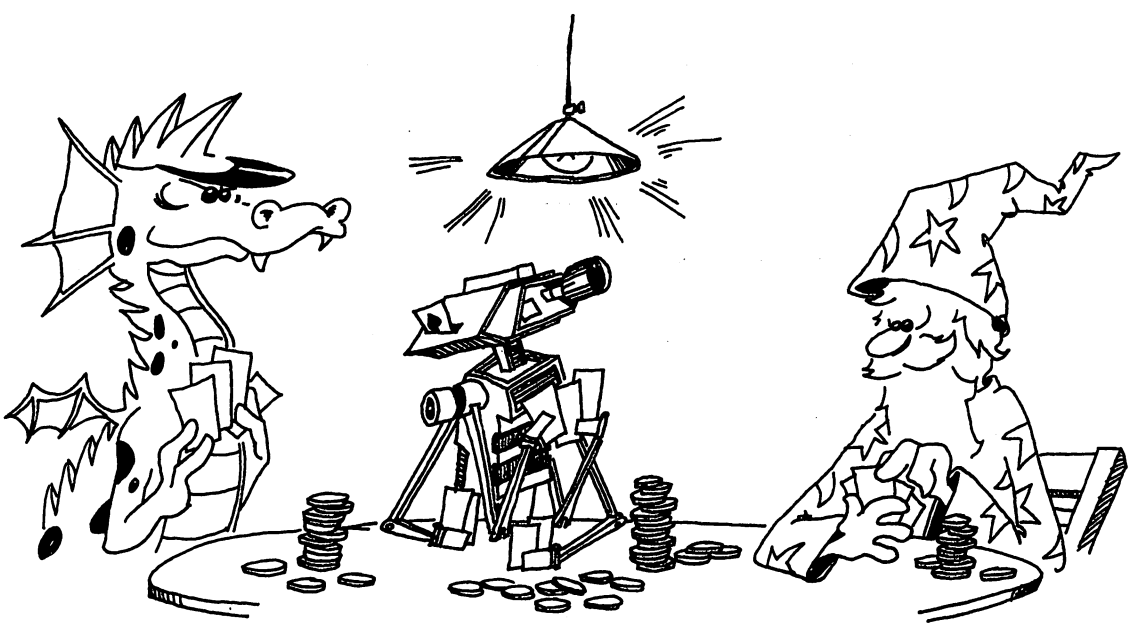
```
10 LET N=1
15 PRINT N;
20 IF N=13 THEN PRINT "cyn UNLUCKY! wht"
30 LET N=N+2
40 IF N>30 THEN GOTO 99
50 GOTO 15
99 PRINT "DONE"
```

What will it print if line 10 is changed to:

```
10 LET N=2
```

3. Write a program which 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 which player 1 has entered. Then player 2 must enter the suit (club, diamond, heart or spade) and the value (1 through 13) of the card. First she guesses the suit, then the program goes on to ask the value. Keep score.



INSTRUCTOR NOTES 13 RANDOM NUMBERS AND THE INT FUNCTION

This lesson introduces two functions: RND and INT. 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 and smaller than 1. 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 directly usable.

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

After extending the random number to a 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 "rounding off" may be familiar to your student. INT will round off a number if you first add 0.5 to it.

The concept of functions is 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(8) 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.

You need some way to "roll dice" and "deal cards" and do other unpredictable things with the computer.

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(8)
30 PRINT N
40 IF N<.95 THEN GOTO 25
```

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

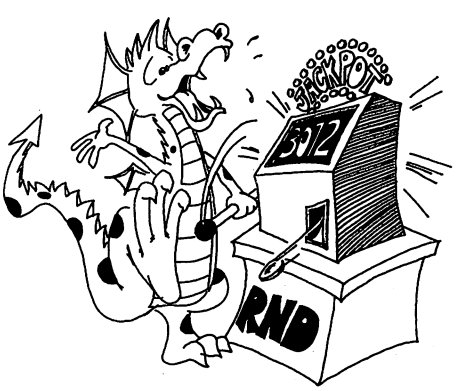
It doesn't matter what number you put into the parentheses just so long as it is positive. I chose "8" because it is near the "( )" signs on the keyboard, making it easy to type (8).

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

Change the program above to:

```
25 LET N=RND(8)*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 23 rather than decimal numbers like 7.03 and 23.62. 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. Change the program above and run again:

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

HOW IT WORKS

Use this one line program:

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

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

ROUNDING OFF NUMBERS

Perhaps you know about "rounding off" numbers. If the decimal part starts with .5 or above, you round up; if it starts with .4 or below, you round down.

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

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

Run

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

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

ROLLING THE BONES

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

```
10 REM ///ONE DIE///
20 PRINT "clr dn dn dn"
30 LET R=RND(8)
40 PRINT " RANDOM NUMBER";TAB(15);R
50 LET S=R*6
55 PRINT "dn TIMES 6";TAB(15);S
60 LET I=INT(S)
65 PRINT "dn INTEGER PART";TAB(15);I
70 LET D=I+1
75 PRINT "dn DIE SHOWS";TAB(15);D
80 FOR T=1 TO 2000: NEXT T
85 GOTO 20
```

WHAT GOES INSIDE THE ( ) ?

Numbers: `10 LET X=INT(34.7)`

Variables: `10 LET X=INT(J)`

Expressions: `10 LET X=INT(3*Y+2)`

Functions: `10 LET X=INT(RND(8))`

Here is how to save a lot of room.

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

Use just `70 LET D=1+INT(RND(8)*6)`

RANDOM NUMBERS IN THE MIDDLE

Suppose your game needs random numbers between 6 and 8. You have a funny die which shows only 6, 7 or 8 when you roll it.

Run this:

```
10 LET D=INT(RND(8)*3)+6
```

How it works:

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

Assignment 13:

1. Write a program which "rolls" two dice, called D1 and D2. Show the number on D1 and on D2 and the sum of the dice. You do not need the variables R, S and I in the program above. They were used to show how the final answer was found.

2. Write a program which shows the roll of a special die. It is a cube (six sides) and the sides have the numbers 10, 12, 14, 16, 18 and 20 on them.

3. Write a "paper, scissors and rock" game, 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. However, use your own judgement and insert this lesson at an earlier point in the flow of lessons if you wish.

The Commodore tape recorder is similar to ordinary recorders which play music, but the computer can automatically turn it on and off.

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 five 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.

On some versions of the Commodore 64, you have to press the CTRL key to continue LOADing and VERIFYing operations.

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

The Commodore 64 computer has a powerful tape operating system using file names. Our first example will not use file names.

SAVING YOUR FIRST PROGRAM TO CASSETTE TAPE

1. List your program to make sure it is still there.

2. Put a new tape into the recorder and rewind it.

3. Enter SAVE

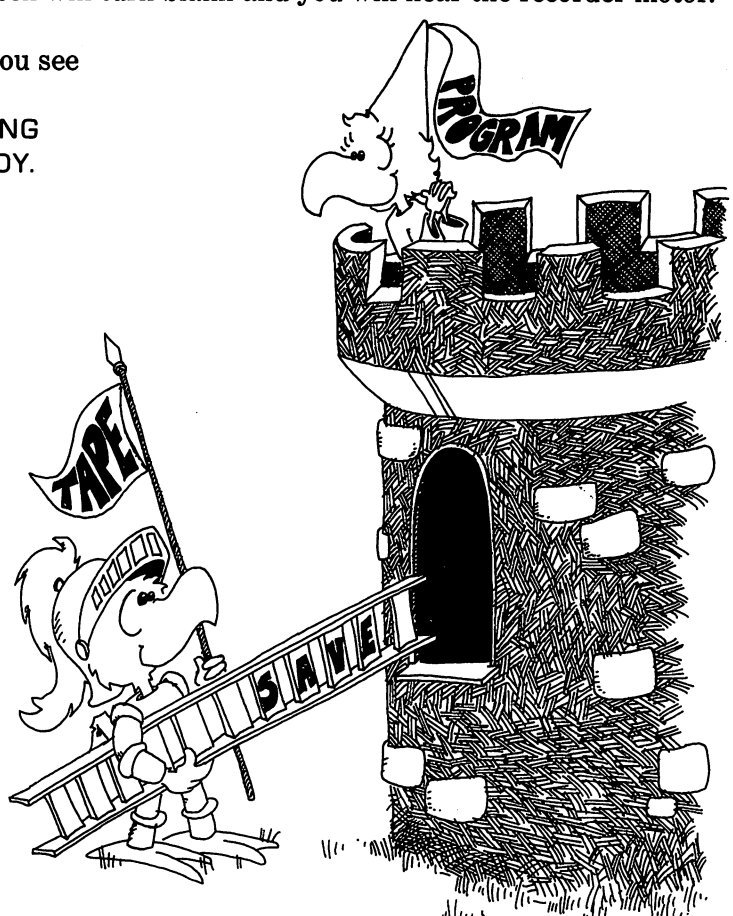
4. The computer will print

   PRESS RECORD & PLAY ON TAPE

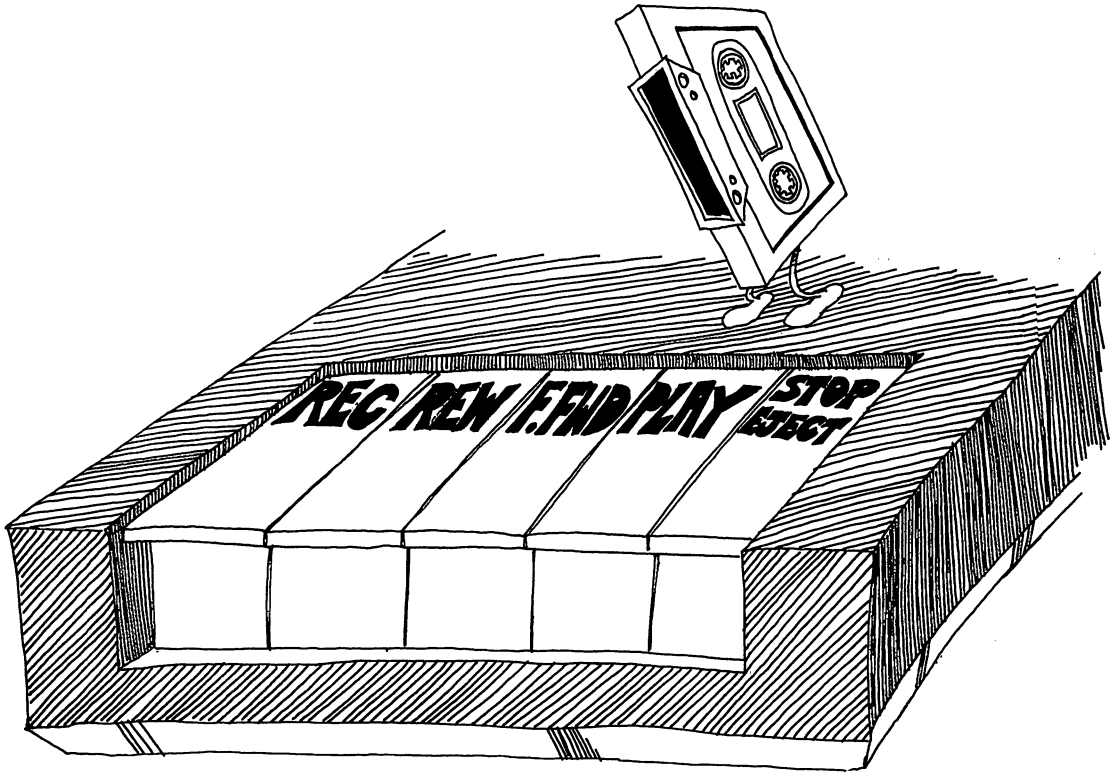
5. Hold down the REC button and press the PLAY button. Both will click and stay down. The screen will turn blank and you will hear the recorder motor.

6. In a moment you see

   SAVING
   READY.



The computer automatically turned 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

See if 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 on the recorder to put all its keys 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 screen will go blank. In a moment you see

```
SEARCHING
FOUND
```

6. The screen will go blank. In a moment you should see

```
VERIFYING
OK     or     ?VERIFY ERROR

READY.
```

(If the screen stays stuck showing FOUND, press the CTRL key to make it start verifying.)

If the computer says "?VERIFY ERROR", it means the program in memory is not EXACTLY the same as the one on tape. If you didn't change the one in memory, then the one on tape is no good.

LOADING A PROGRAM FROM TAPE INTO THE COMPUTER MEMORY

We will load a program without using a filename. The computer will load the first program it finds on the tape.

WARNING! 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. The screen will go blank.

5. In a moment, the computer will print

   SEARCHING
   FOUND

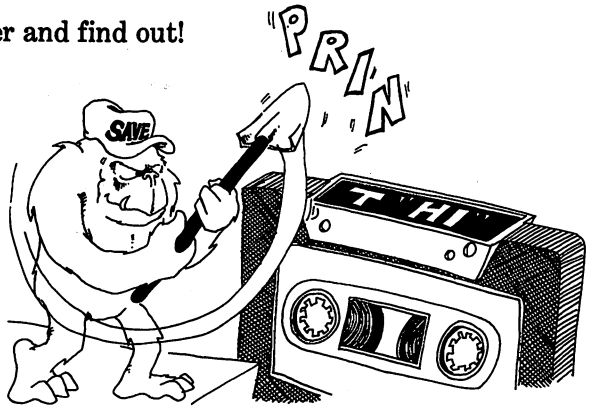
6. In a moment the screen will go blank again. (If it doesn't, press the CTRL key.) You hear the recorder start up. In a moment you see

   LOADING
   READY.

7. Now do a LIST to see if the program is in the computer.

WHAT DOES THE PROGRAM SOUND LIKE?

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
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 chose "HOT DOG" for the file name. It is a good idea to make 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.

   The file name can be up to 16 characters long.

2. The computer says

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

3. Do so. The computer says

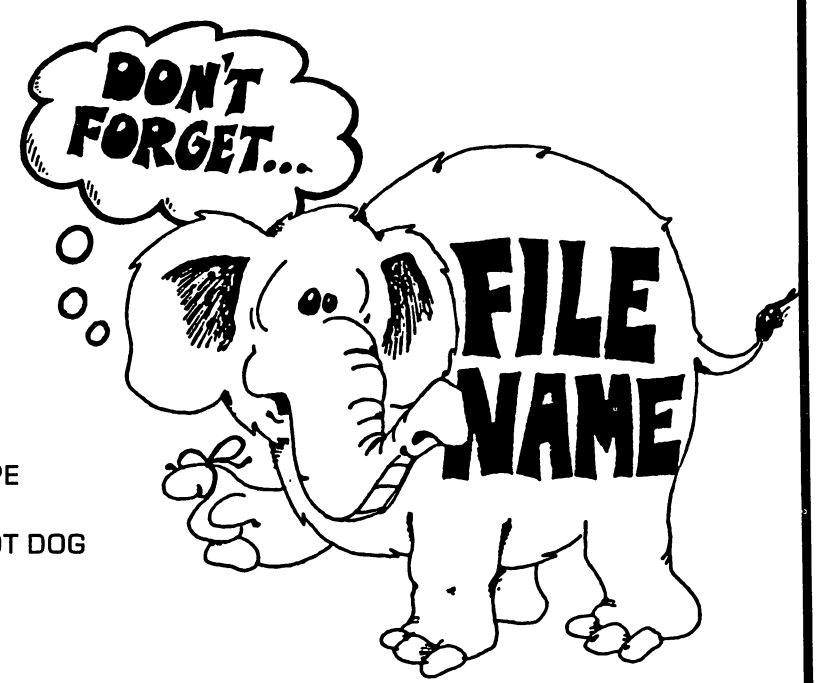
   ```
   OK

   SAVING HOT DOG
   READY.
   ```

4. Rewind the tape and enter

   ```
   VERIFY "HOT DOG"
   ```

   The computer says

   ```
   PRESS PLAY ON TAPE

   SEARCHING FOR HOT DOG
   FOUND HOT DOG

   VERIFYING
   OK

   READY.
   ```



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

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

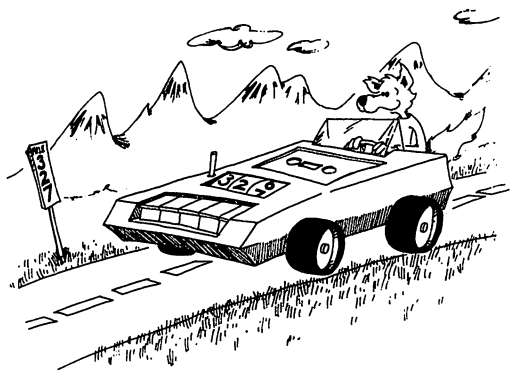
```
VERIFY "MICE"
```

You will see

```
PRESS PLAY ON TAPE  (do it.)
OK

SEARCHING FOR MICE
FOUND HOT DOG  (pause)

FOUND MICE  (pause)

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 doing a LOAD or a VERIFY without putting 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. 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)?

INSTRUCTOR NOTES 15 SOME SHORTCUTS

1.	?	used for PRINT
2.	LET	omission
3.	:	used between statements on a line
4.	INPUT	used with a message
5.	INPUT	error messages
6.	LIST X-Y	lists specify sections of program
7.	THEN 33	instead of THEN GOTO 33
8.	commands compared to statements	

Having reached RND and the saving of programs on tape, the sprint is over. 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 which have something in common.

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

The colon can mess up a program, too. Be careful about adding other statements onto a GOTO, a REM or an IF line.

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

QUESTIONS:

1. What shortcut does the "?" give?

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

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

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

5. What is wrong with each of these lines?

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

6. If the computer prints "??" after you answer an input, what does it mean?

LESSON 15 SOME SHORTCUTS

A PRINT SHORTCUT

Instead of typing PRINT, just type a question mark.

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

The computer substitutes the word PRINT for the question mark.

A LET SHORTCUT

These two lines do the same thing:

10 LET A=41 and 10 A=41

also these two: 20 LET B$="HI" and 20 B$="HI"

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

AN INPUT SHORTCUT

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

You can do
```
10 INPUT "ENTER YOUR NAME"; N$
```

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

ANOTHER INPUT SHORTCUT

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

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

You see `LOCATION?` on the screen.

You enter two numbers with a comma between them.

```
LOCATION? 5,6
```

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

After the "?" type
```
APRIL,29,1983
```

ERROR MESSAGE IN INPUT

If you do not enter enough answers, the computer asks "??". Then you should enter the rest.

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

? MAY, 1
?? 1984
```

If you enter too many answers, the computer replies

```
?EXTRA IGNORED
```

and goes on with the program.

Example:
```
55 INPUT"SLEEPY? <Y/N>";A$

? NO,FINE
?EXTRA IGNORED
```

ANOTHER ERROR MESSAGE

Run 10 INPUT N, A$

Try these pairs of answers:
1, B
B, 1
1, 1
B, B

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

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

A LIST SHORTCUT

There are 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 THEN SHORTCUT

Instead of 10 IF A=B THEN GOTO 33

Use 10 IF A=B THEN 33

A COLON SHORTCUT

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

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

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

When you LIST the line, you see

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

WHEN TO USE THE COLON SHORTCUT

Use the shortcut:

1. To make the program clearer.

   Put similar statements on the same line. Example:

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

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

2. To make the program shorter.

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

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

THE COLON AFTER AN IF COMMAND

You can make neater IF statements using colons.

Without:
```
50 IF A=0 THEN GO TO 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.

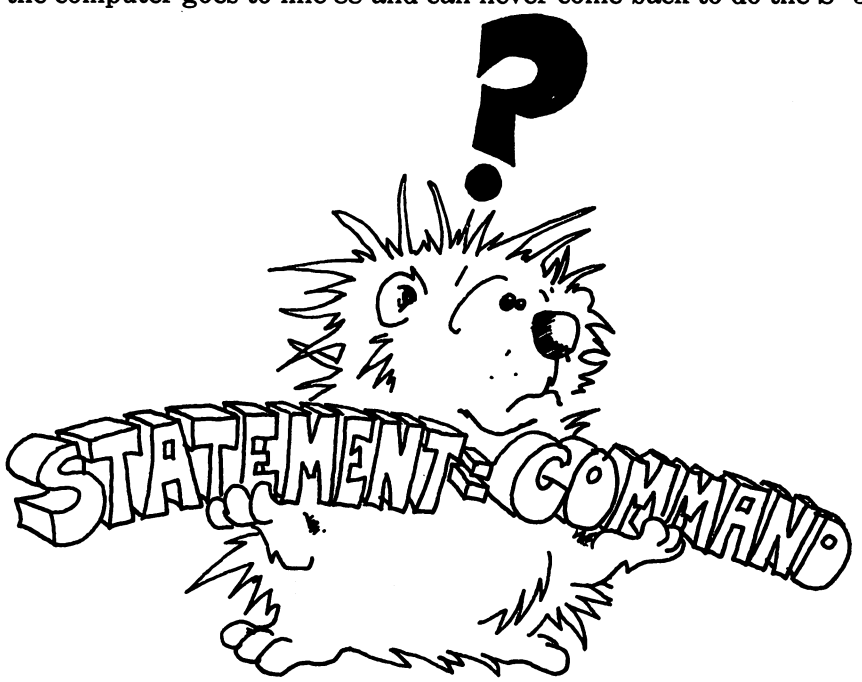
Correct: `35 P=3:REM P IS THE PRICE`

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 `LIST` We say we have "entered a command."

But if we write this line in a program:

```
20 LIST
```

we say that line 20 has one "statement," the LIST command.

Some lines have several statements, separated by colons.

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

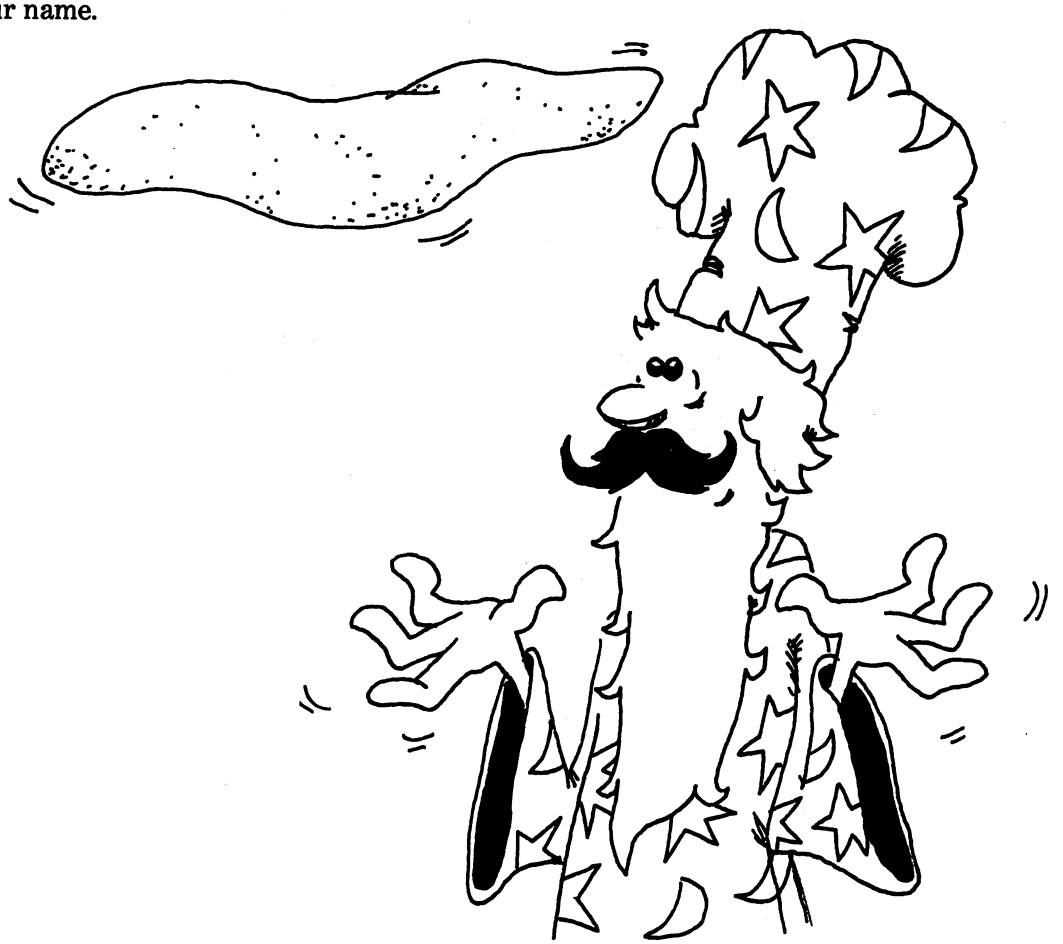
is a line with three statements.

Assignment 15:

1. Write a program which 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 which 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. A scroll will also occur if you print in the 40th column on line 25. This won't scroll:

```
10 FOR I=1 TO 24:PRINT I:NEXT I
20 PRINT"123456789012345678901234567890123456789";
30 GOTO 30
```

But add a "0" to the end of the string (making 40 characters) and the screen will scroll.

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

The remedy is slow, careful and complete analysis of the print statements, 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 which together will put a letter "A" on the screen at the point three lines down and seven spaces across.

3. In this line, the word "HAPPY" is printed. Where is the PRINT cursor after the word is printed?

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

   Can you see the PRINT cursor on the screen?

4. Now write a line 20 which will erase the letters "APP" from the word printed in question 3.

LESSON 16 MOVING PICTURES

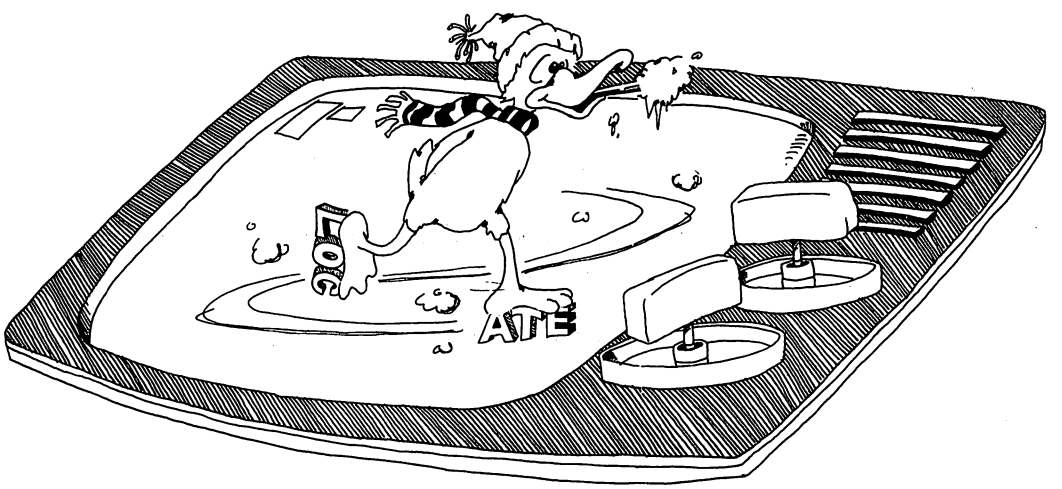
MOVING THE CURSOR UP AND DOWN

To move the cursor, put CRSR characters into 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 into the PRINT quotes. Do not put in any spaces! We write "dn dn" instead of "dndn" so you can read it easily. (If we want you to put in a space, we will write "sp".)

Remember that the "dn" character looks like a reverse "Q", "clr" like a reverse heart and "hm" like a reverse "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. (Or you can use spaces instead of "rt" characters.)

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";:REM WINGS DOWN
40 FOR T=1 TO 200:NEXT T
50 PRINT"  lf lf lf";:REM WINGS UP
60 FOR T=1 TO 200:NEXT T
70 GO TO 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?

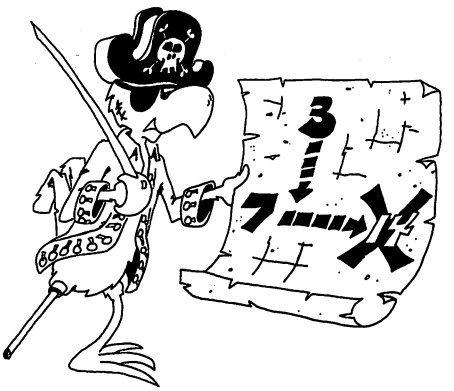
Read lines 30 and 50 character by character.

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, GO 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 which 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

A loop is made up of a FOR statement (which may contain a STEP command) and a NEXT statement. These statements may be separated by several lines and yet are strongly interdependent. The student builds on the notion of a delay loop and learns the utility of repeating a set of commands in the middle of 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 which 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 numeric variable. The STEP value, the ending value and the address of the first statement after the FOR are put on a stack.

From now on, all the looping action takes place at the NEXT 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. 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. What is the "loop variable" in this line?

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

2. How do you make the loop "count down" instead of "count up"?

3. What is meant by "nested loops"?

4. Write a loop which prints the numbers from 0 to 20 by two's.

5. Write a "ten little Indians" program which prints the numbers from 10 down to 0 Indians.

6. Write a pair of nested loops to print MINI in the outer loop and HA in the inner loop. Print three of the MINIs and for each print two of the HAs.

LESSON 17 FOR-NEXT LOOPS

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

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

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

Run this:

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

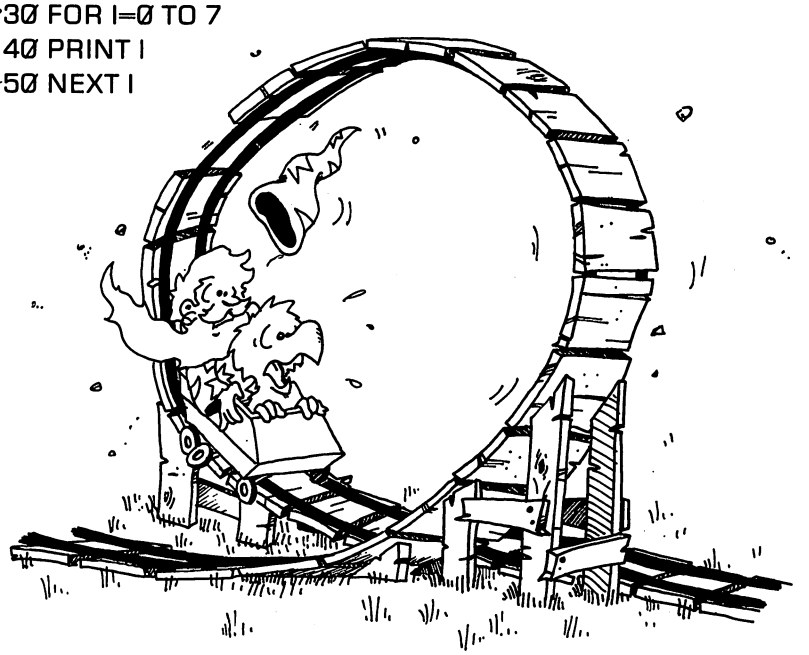
The loop can start on any number and end on any higher number. Try changing line 30 in these ways:

```
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 zero to 100.

COUNT DOWN LOOPS

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

Try this:

```
10 REM ** APOLLO 11 **
15 POKE 53281,15
20 PRINT"clr"
30PRINT" T MINUS 12 SECONDS AND COUNTING"
35 FOR I=1 TO 750:NEXT I
40 FOR T=11 TO 0 STEP -1
50 PRINT "clr dn";T
60 FOR I=1 TO 750:NEXT I:REM TIMING LOOP
70 NEXT T
80 PRINT "clr dn red ALL ENGINES RUNNING, LIFT OFF."
81 FOR I=1 TO 750:NEXT I
82 PRINT "clr dn grn WE HAVE A LIFT OFF."
83 FOR I=1 TO 750:NEXT I
84 PRINT "clr dn blu 32 MINUTES PAST THE HOUR."
85 FOR I=1 TO 750:NEXT I
86 PRINT "clr dn cyn LIFT OFF ON APOLLO 11."
90 POKE 53281,0
```

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 I=1 TO 2000:NEXT I
```

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

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

T 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:

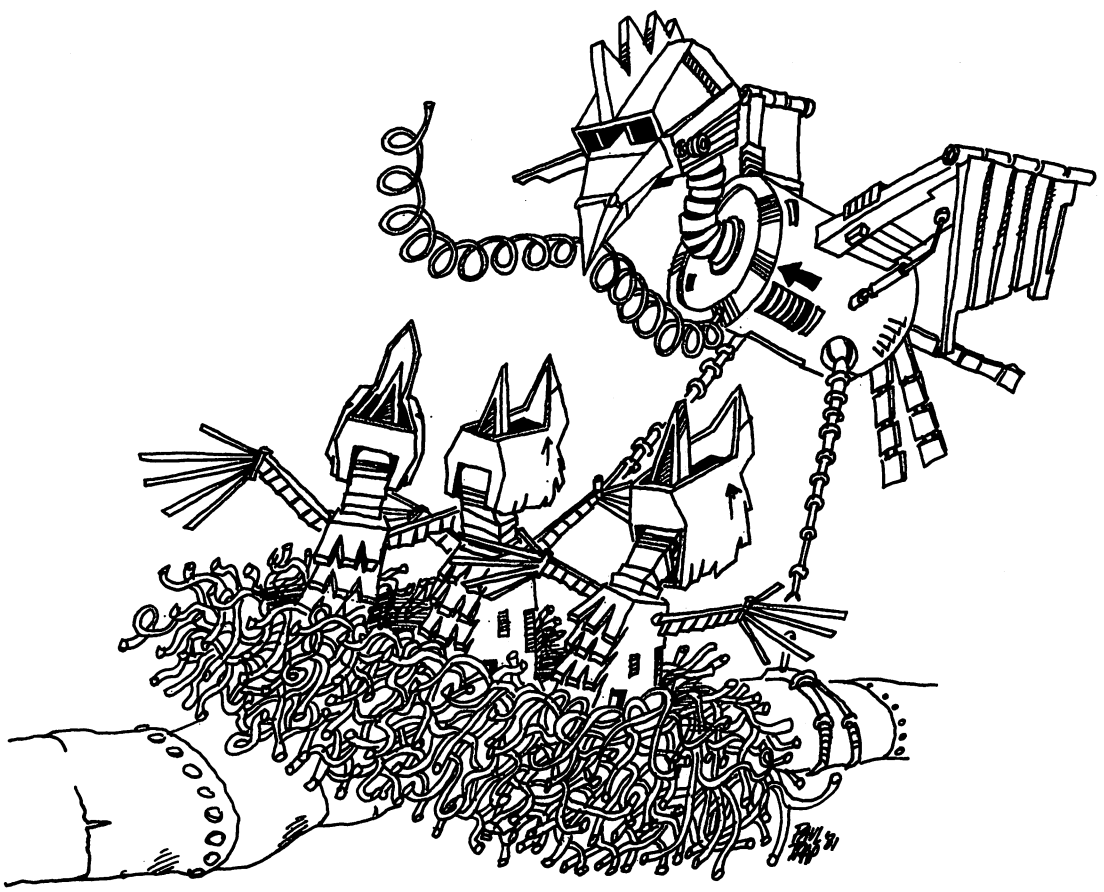
1. Write a program which prints your name 15 times.

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

3. Now make it write your name 23 times, starting at the bottom of the screen and going up. Use the "up" keys in a PRINT and put it all into a loop.

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

5. Write a program with loops nested three deep. Do the outer loop three times and have it print SING. Make it change the screen color each time through. The next loop prints TRA. Have it loop three times. The innermost loop prints LA three times.



INSTRUCTOR NOTES 18 DATA, READ, RESTORE

This lesson concerns 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 most common uses of DATA is to fill arrays with initial values. We also use it in the MUSIC lesson to store note pitches and durations.

QUESTIONS:

1. What happens if you try to READ more data items than are in the DATA statements?

2. What rule tells you where to put the DATA statements in the program? How about where to put the READ statements?

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

4. Can you put numeric data and string data into the same DATA statement?

5. What happens if you try to READ a string into a numeric variable?

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

LESSON 18 DATA, READ, RESTORE

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$;"wht"
```

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

2. The kind which is stored in the program at the time it is written.

```
10 REM THE SECOND KIND OF DATA
20 PRINT"clr dn dn dn"
30 X=57
40 Y$="FLAVORS"
50 PRINT X;Y$
```

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

STORING LOTS OF DATA

It is OK to store small amounts of data in LET statements. But it is awkward to store large amounts of data that way.

Use the DATA statement to store large amounts of data.

Use the READ statement to get the data from the DATA statement.

```
10 REM LOTS OF DATA
20 PRINT"clr dn dn dn"
30 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,
THURSDAY,FRIDAY,SATURDAY
40 READ D1$:READ D2$:READ D3$:READ D4$
60 PRINT D1$,D2$,D3$,D4$
```

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

STRANGE RULES

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

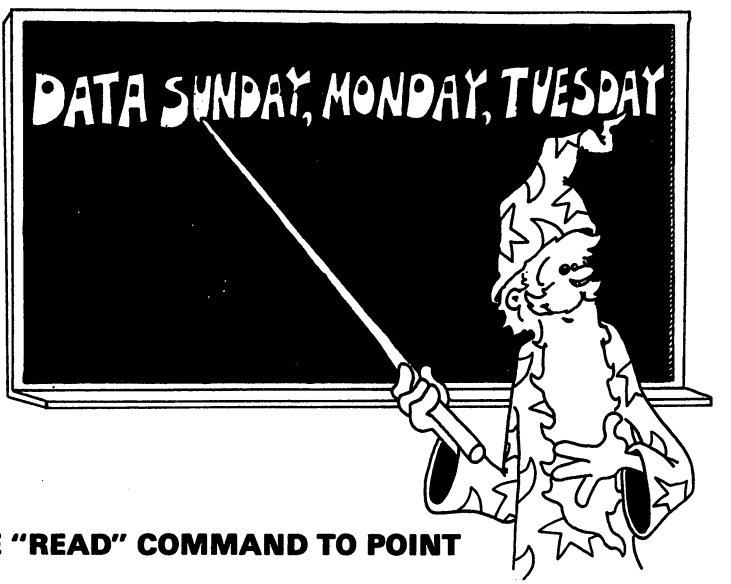
   Do this: Change line number 30 in the above program to line number 90. Run the program. It works just the same.

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

   Do this: Break the DATA statement into two:

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

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



IT IS POLITE 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
```

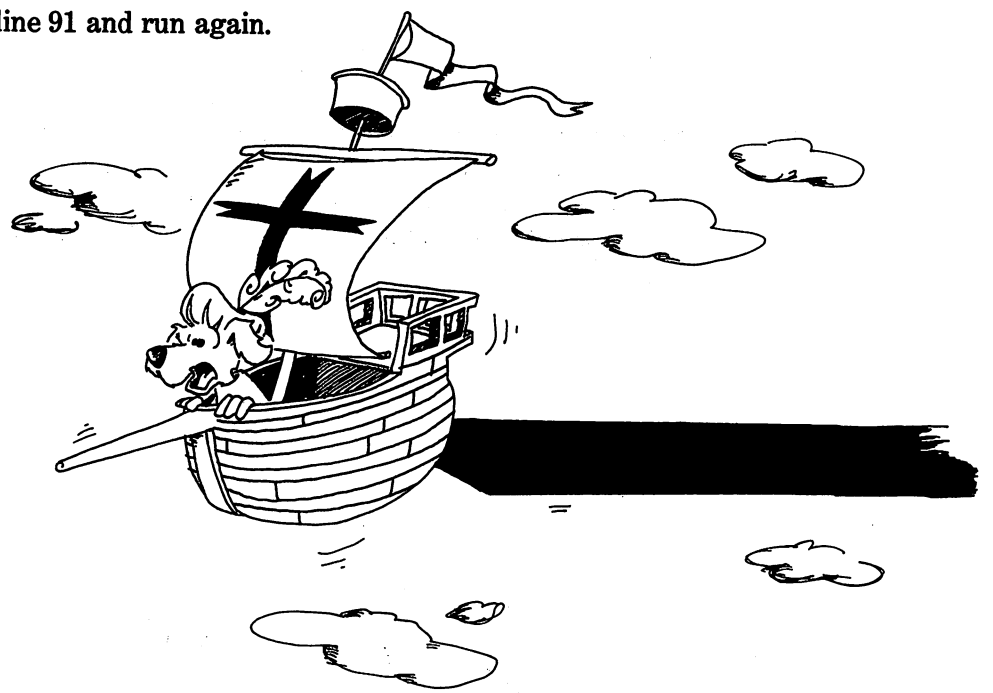
Run the program. It works just the same.

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 ERROR IN XX
```

Erase line 91 and run again.



BACK TO SQUARE ONE

At any point in the program, you have only two choices for the READ pointer.

1. You can do another READ: then the pointer moves ahead one item.
2. You can 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$
```

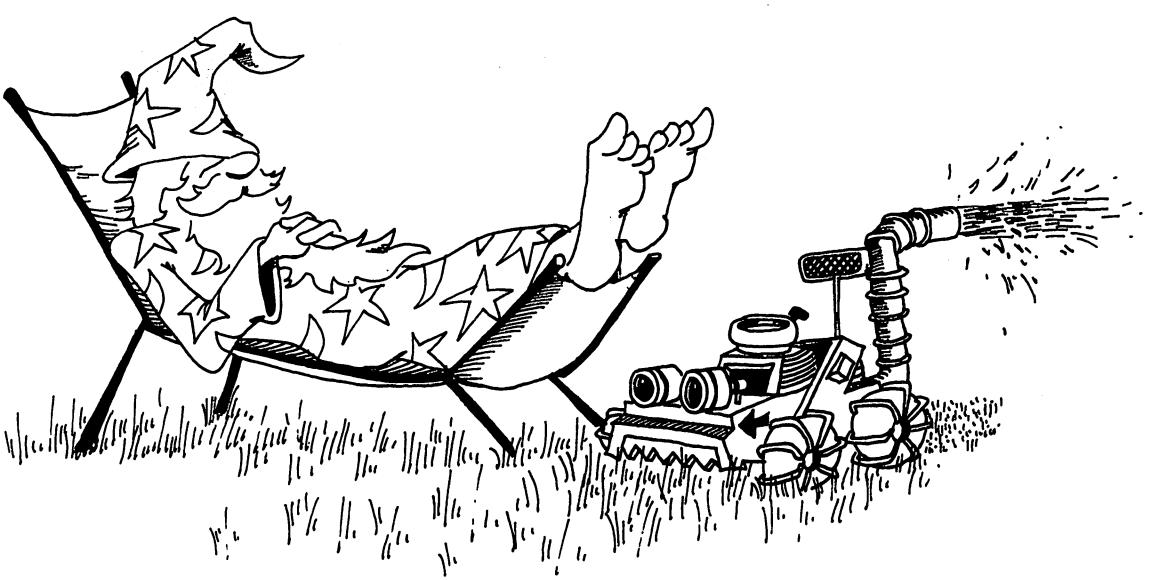
Wrong:

```
70 DATA 77,FUZZ
75 READ B$        OK, B$ box holds "77"
80 READ N         ?SYNTAX ERROR IN 70
```

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

Assignment 18:

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



INSTRUCTOR NOTES 19 SOUND EFFECTS

The 6581 Sound Interface Device (SID) chip in the C-64 produces sound and music. This lesson introduces some features of the chip which are useful for making sound effects. Musical applications are covered in a later lesson.

The SID chip is controlled by POKEing numbers into its registers. The chip is versatile but fussy to program.

Sound is sent to the TV speaker as part of the composite signal. If you use a color monitor instead, consult the C-64 manual to see the pin-out for sending the sound signal to an amplifier and speaker.

The chip can produce up to three independent voices or sounds at the same time. The pitch of each voice is set by POKEing two bytes. This allows very precise intonation, a feature lacking in many other home computers. For example, you can experiment with the various temperaments of the musical scale.

There are four voices for the wave form of each voice, producing different "timbres" for the voices: flute, banjo, etc. One of the choices is "white noise" which is very useful for producing sound effects (drums, gunfire, thunder, wind, waves, etc.).

The loudness of the total sound output is set by one POKE, and each voice can be switched into a programmable filter which modifies its harmonic content, further altering the timbre.

Each voice has a controllable pattern in loudness from start to end. Four numbers control the note shape: two for its start (the "attack" followed by a "release") and one each for its loudness as it is held (the "sustain" value) and the rapidity with which it decays after turning it off (its "decay"). These ADSR parameters are fully described in the music lesson.

The length of the note is controlled by a delay loop you write into the program, rather than by the SID chip. That is, you must start the note with a POKE, then go into a timing loop, and finally stop the note with one or two more POKEs.

QUESTIONS:

1. What things must you POKE to get a sound?

2. How do you turn off all the voices at once?

3. Why must you be careful of the box number you POKE?

4. What number gives the loudest sound?

LESSON 19 SOUND EFFECTS

Turn up the sound on the TV now.

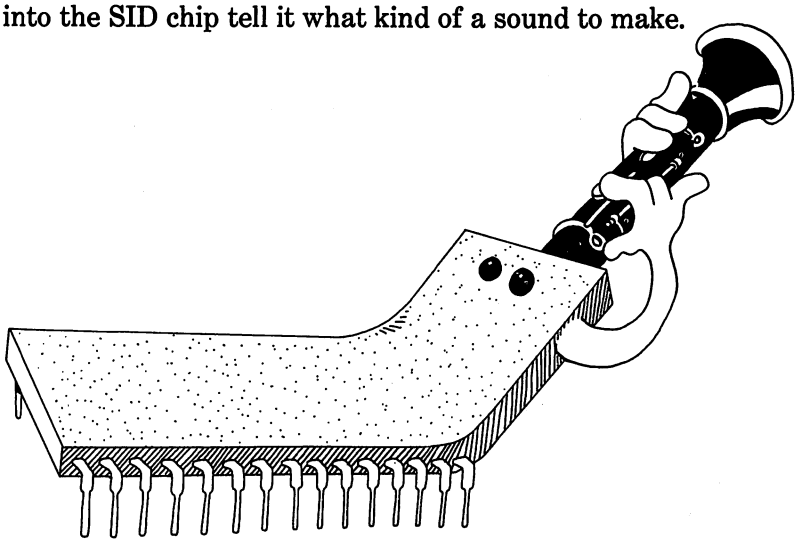
POKE SID A FEW TIMES

No, not Sidney. SID is the chip in the computer which makes the sounds and musical notes.

There are 14 boxes in memory which are right inside the SID chip. POKE tells the computer to put a number between 0 and 255 into a memory box.

POKE 54273,23 means put number 23 in box 54273

The numbers put into the SID chip tell it what kind of a sound to make.

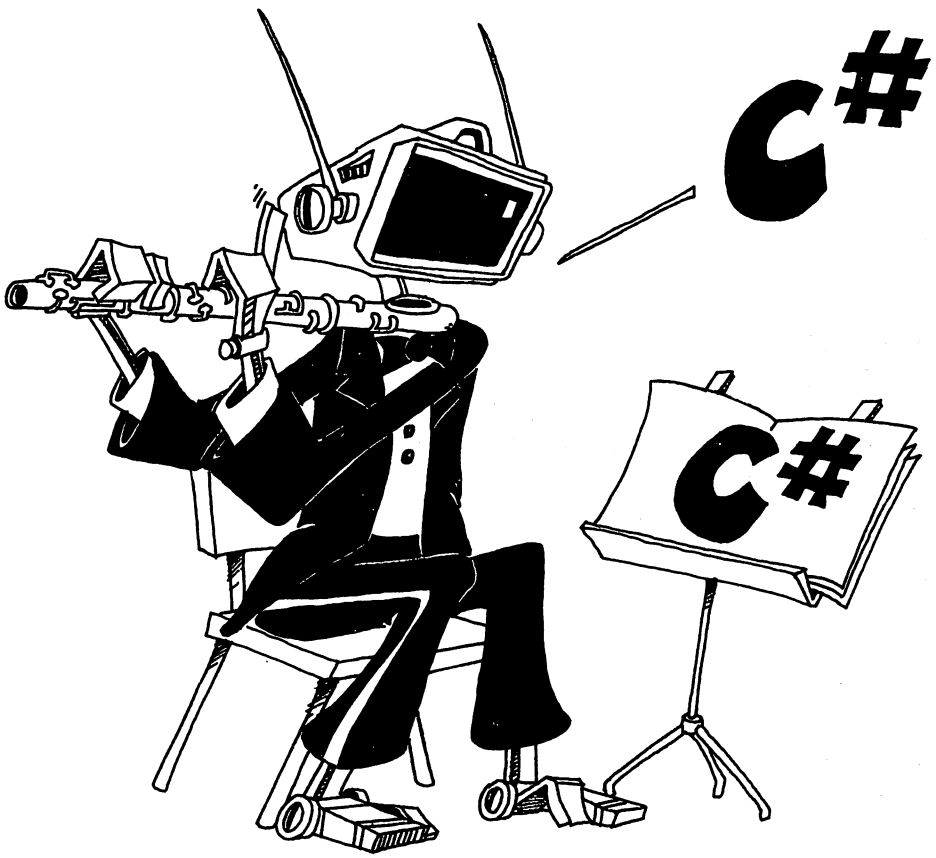


JOHNNY ONE NOTE

Enter this:

```
10 REM SID TUNES UP ON "CONCERT A"
20 POKE 54273,23 :POKE 54272,181
22 POKE 54296,15 :REM VOLUME 15
24 POKE 54277,9 :REM ATTACK-DECAY
26 POKE 54278,0 :REM SUSTAIN-RELEASE
30 POKE 54276,33 :REM TIMBRE, VOICE ON
85 FOR T=1 TO 1000:NEXT T
90 POKE 54276,32 :REM VOICE OFF
91 POKE 54273,0 :REM SOUND OFF
```

Save to tape or disk BEFORE running!



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 the program you have in memory.

Run it. You should hear the "orchestra tuning pitch" middle A. If you do not hear a sound, first check that the TV is turned up loud. Then check that EVERY number in the program is correct.

BUT HEAVENS, SO MUCH WORK FOR ONLY ONE NOTE?

Yes! It takes five POKES to set up the sound chip (lines 20, 22, 24 and 26). It takes one more POKE to turn on the sound (line 30) and two to turn it off again (lines 90 and 91).

But notes are cheaper by the dozen. You can make more notes, even a whole song, with only two more POKEs per note and, in fact, if you do it in a loop and use DATA, it is almost easy.

Let's explore the SID chip some more.

TONES BY THE BUSHEL

Enter:

```
10 REM ******* TONES *******
11 :
15 REM ---------- CHIP BASE ADDRESS
16 C=54272
17 REM ---------- VOICE 1
18 :
19 REM ---------- REGISTER ADDRESSES
20 LO=C :REM --- HI FREQUENCY NO.
21 HI=C+1 :REM --- LO FREQUENCY NO.
24 TM=C+4 :REM --- TIMBRE OF NOTE
25 AD=C+5 :REM --- ATTACK-DECAY
26 SR=C+6 :REM --- SUSTAIN-RELEASE
29 VL=C+24:REM --- LOUDNESS
30 REM ---------- CLEAR THE CHIP
31 FOR L=C TO C+24:POKE L,0:NEXT
40 REM ---------- ADSR SOUND, LOUDNESS
42 POKE AD,9
43 POKE SR,2*16+8
44 POKE VL,15+0
50 REM ---------- PICK NOTE
52 PRINT "clr dn ENTER NOTE:<2 TO 253>"
53 PRINT "dn dn 0 TO END PROGRAM dn"
54 INPUT N
56 IF N=0 THEN END
60 POKE HI,N:REM - POKE NOTE
61 POKE LO,0
65 POKE TM,33:REM ------ START NOTE
66 REM 33 IS A SAWTOOTH WAVE FORM
70 FOR T=1 TO 1000:NEXT
72 POKE TM,32:REM ------ END NOTE
73 POKE HI,0 :REM ------ TURN OFF SOUND
80 GOTO 40
```

Save to tape or disk before running!

Try these numbers: 2, 22, 222.

Then try these: 20, 40, 80, 160. They play octaves of the lowest note. Try other numbers.

SID has a block of 24 boxes starting at address 54272. Line 16 in the program stores this address.

Lines 20 to 29 make and store the addresses of more of SID's boxes.

Line 31 puts a zero into each of SID's boxes, which TURNS HIM OFF completely!

It takes two boxes, called HI and LO, to tell SID what tone to play. HI makes a big difference in the pitch, and LO is for fine tuning. In this program we change only the number in HI. LO remains at zero.

Line 65 turns the note on and also says that the note should be a "sawtooth" wave form. We will come back to this later.

Line 70 says the note should be about one second long.

Line 72 turns the note off. That is, it turns the sawtooth off.

But you would still hear a faint lasting sound unless you also turn the sound itself off. This is done by line 73 putting a zero in the HI box.

PICCOLO, PIANO OR WHAT?

SID is a one man band. You can get him to play different instruments by changing the numbers POKEd into the register called TM at address 54276 in lines 65 and 72 of TONES.

"TM" stands for "timbre." Timbre is a musician's word which means "how the sound of one instrument is different from the sound of another instrument."

line 65	line 72	waveform
start	stop	
17	16	triangle
33	32	sawtooth
65	64	pulse
129	128	white noise

If you use "pulse," you also need to POKE a number between 1 and 15 into register 54275. The sound is thin and reedy for low numbers and more mellow for numbers near 15.

Describe the sound of each waveform here:

sawtooth _____

triangle _____

noise _____

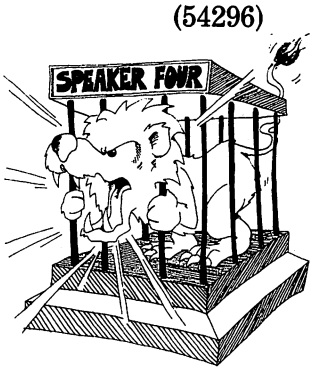
pulse, 50 _____

pulse, 9 _____

SOUND EFFECTS

You can do many things to get different sound effects:

different frequencies	(54272,54273)
different waveforms	(54274-54276)
different ADSR numbers	(54277,54278)
different loudness	(54296)



You can use a loop to change the numbers in these boxes while the sound is going on!

Try this:

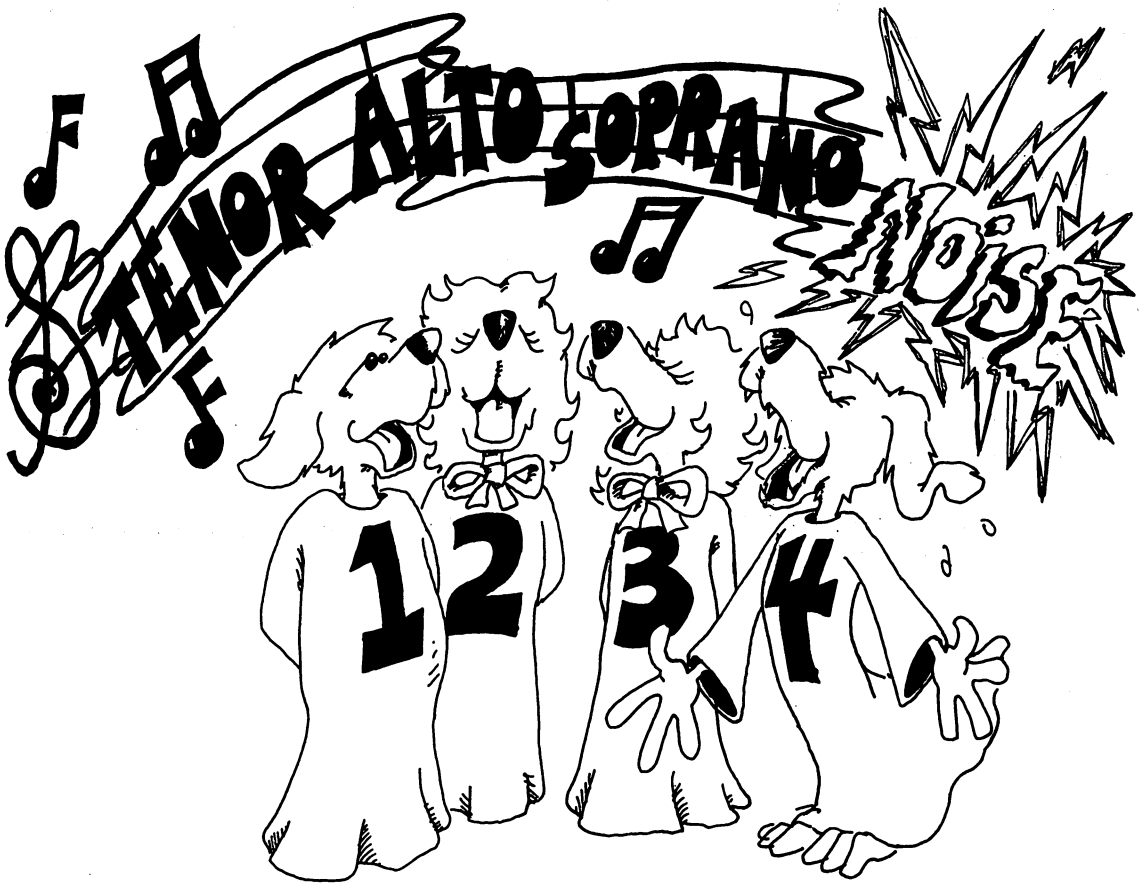
```
10 REM GOING UP
20C=54272:LO=C:HI=C+1
22 AD=C+5 :SR=C+6
24 VL=C+24:TM=C+4
30 FOR L=C TO C+24:POKE L,0:NEXT L
32 POKE VL,15
34 POKE AD,9 :POKE SR,2*16+8
35 POKE TM,33
40 FOR I=2 TO 222
42 POKE HI,I :REM PITCH CHANGES
49 NEXT I
50 POKE TM,32:POKE HI,0
```

Save to tape or disk before running.

Assignment 19:

1. Try to make these sound effects: laser gun, wind (use "noise" waveform), siren, explosion. Try POKEing into different boxes in the SID chip.

2. Write a program to show a flying bird and make the bird chirp as it goes.



INSTRUCTOR NOTES 20 NAMES, CLOCKS AND MODES

This lesson treats the rules for naming variables, the "jiffy" clock and the edit and RUN modes.

Descriptive variable names help clarify programs but they take more typing and there are two hidden "gottcha's." One is that BASIC records only the first two letters of a name. This means that those beautiful descriptive names may not be unique. Even experienced programmers occasionally get caught here, and the bug may be quite hard to detect. One "cure" is to use only one and two character names.

The other gottcha 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 and are listed in an appendix to this book. If a name contains a reserved word the computer prints

```
?SYNTAX ERROR IN XX
```

where XX is the line number containing the illegal name.

The jiffy clock is a register in the computer which contains a six digit decimal number. The number is incremented every 1/60th of a second. The number is in a BASIC variable box named TI. A real clock (hours, minutes and seconds) is described in a later lesson.

The student has been entering lines in the "edit mode" from Lesson 1 on, and has been RUNning programs. Now we explain a little about the difference, and point out how the edit mode treats lines entered into the line buffer. The lesson on debugging makes extensive use of the edit mode options.

QUESTIONS:

1. Names longer than two characters may give you trouble. Why?

2. Names which 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 not have "reserved words" at the front, inside, or at the back. Which words are "reserved"? Where is there a list of reserved words?

5. How do you ask the jiffy clock "what time is it"?

6. How does the jiffy clock differ from ordinary clocks?

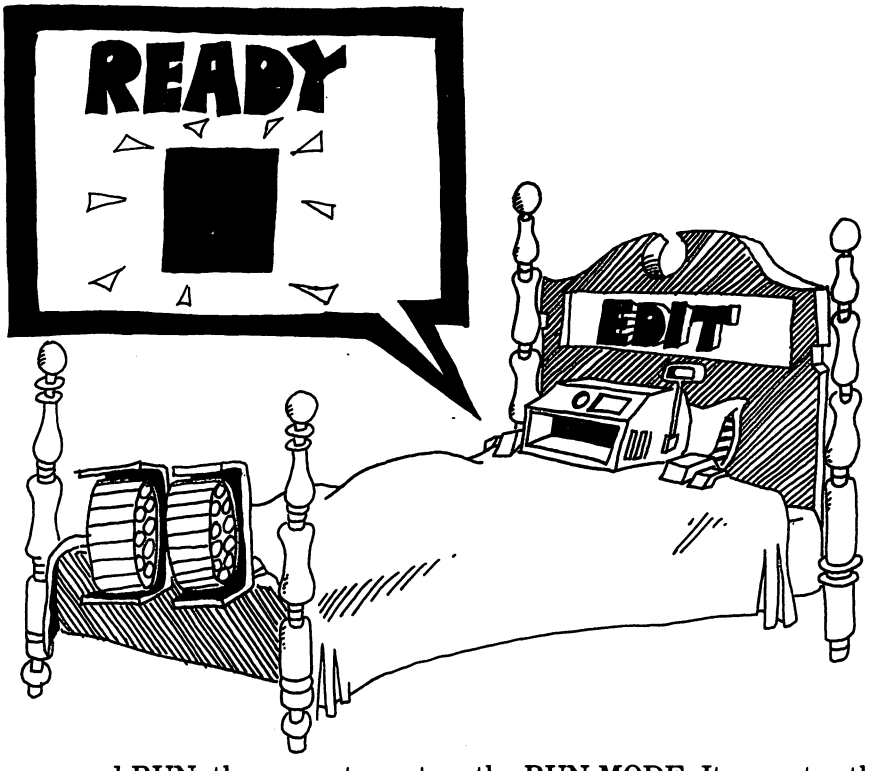
THE EDIT MODE AND THE RUN MODE

The computer is in the EDIT MODE when you first turn it on. You know this because you see

```
READY.
```

and a flashing cursor.

In the EDIT MODE, the computer is almost "asleep," waiting for you to type something and press the RETURN key. The flashing cursor is like the computer "snoring."



When you command RUN, the computer enters the RUN MODE. It executes the program which is in its memory. When done, it goes back to the EDIT MODE.

THE LINE BUFFER

When you are in the EDIT MODE and enter a line, one of two things can happen:

Enter this: `99 G=G+1:PRINT G`

The computer adds this line to the program in memory.

Enter this: `G=17*16:PRINT G`

The computer prints the answer right away.

Anytime you type, the characters are stored in a special box in memory called a "line buffer." When you press the RETURN key, the computer looks into the line buffer box.

RULE: If it sees a line number at the start of the line, the computer moves the line into the program in memory.

RULE: If it doesn't see a line number, it executes the line like a program. The line may have several statements separated by colons ":". After execution, it forgets the line!

THE COMPUTER'S JIFFY CLOCK

The computer has a clock which starts when the computer is turned on and runs all the time. (Well, most of the time. It is turned off while the computer loads from tape or saves to tape.)

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

Run

```
10 PRINT TI
20 GOTO 10
```

The clock can tell you how long something takes to do by subtracting before and after readings of the TI variable. Divide the jiffy clock reading by 60 to get the time in seconds.

Run

```
10 T=TI
20 FOR I=1 TO 1000:NEXT I
30 S=TI-T
40 PRINT "THE LOOP TOOK";S;"JIFFIES"
50 PRINT "THAT IS";S/60;"SECONDS"
```

Assignment 20:

1. Read the naming rules numbered 1 through 8. For each rule make up two names, one which is correct, and one which 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.

2. Is SUPERCALIFRAGELISTICEXPIALIDOCIOUS a legal name? Why or why not? If not, fix it.

3. How can you tell if the computer is in the edit mode?

4. What does the computer do in the RUN mode?

5. What mode does the computer enter when the program has finished running?

6. Write a "reaction time tester." Flash a question mark on the screen at a random time. The user presses the RETURN key as soon as she sees the question mark. Measure the time delay by counting how much the "jiffy" clock has changed. Print out the time for the user to see.

INSTRUCTOR NOTES 21 COLOR GRAPHICS

This lesson demonstrates most of the color features of the C-64.

The next lesson finishes with the POKE commands which 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 POKEs to the memory addresses 53280 and 53281.

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 into 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 you are writing a program which has POKEs, it is wise to save it to tape or disk FIRST, before running it. Then if you have a computer crash, you can cold start, 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 box do you POKE to change the screen background color?
6. What box do you POKE to change the screen border color?
7. What happens if you press the COMMODORE FLAG key and a color key?
8. What danger is there in using POKE?

LESSON 21 COLOR GRAPHICS

ADJUST YOUR COLOR TV OR MONITOR

Put all the colors on the screen like this:

First press CTRL and RVS ON. (Now "space" will be a colored spot.)

Press CTRL and the RED key. Then hold down the SPACE BAR to make a red strip on the screen.

Repeat with the rest of the color keys. Press CTRL and RVS OFF.

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

COLOR SCREEN

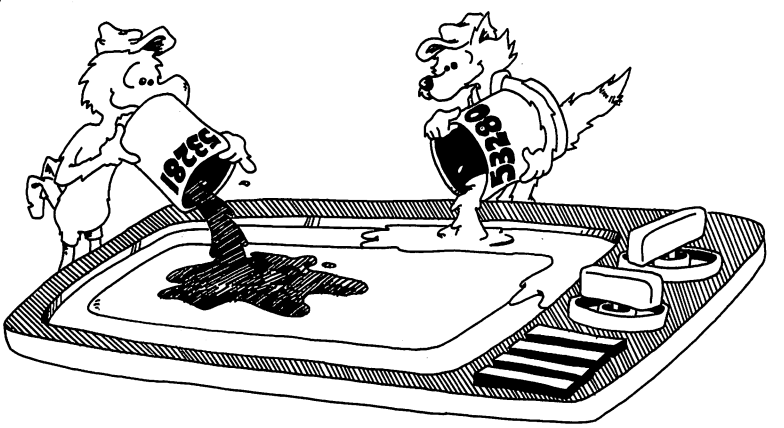
Remember that the box numbered 53281 holds a number from 0 to 15 which picks a number for the screen background color.

There is a another memory box which tells the computer what colors to show on the border of the screen.

Its number is 53280.

Run:

```
10 REM COMBINATIONS
20 SBOX = 53281:REM     SCREEN BOX
21 BBOX = 53280:REM     BORDER BOX
25 PRINT "clr dn dn dn"
30 INPUT "WHAT COLOR SCREEN <0 TO 15>";SC
37 POKE SBOX,SC
38 PRINT "clr dn dn dn"
40 INPUT "WHAT COLOR BORDER <0 TO 15> ";BC
50 POKE BBOX,BC
90 GO TO 25
```



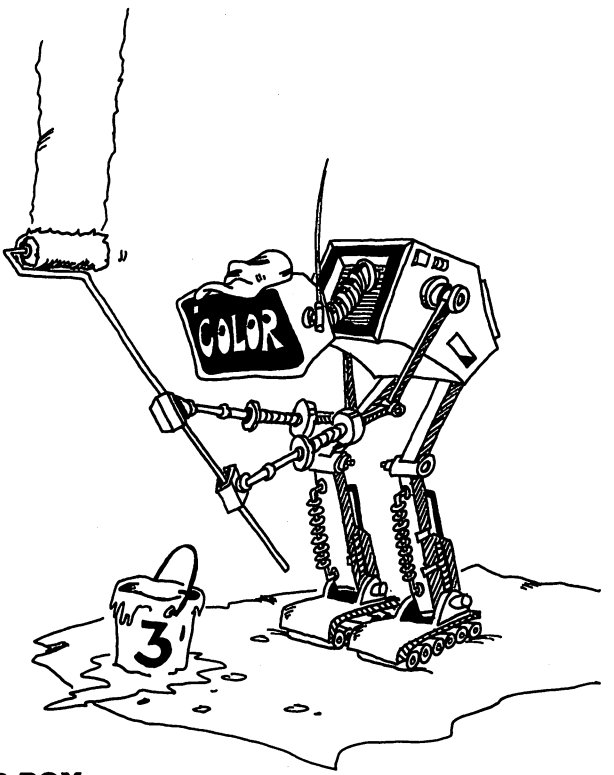
COLOR NUMBERS

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

The screen can have any of these 16 colors.

(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!



THE CHARACTER COLOR BOX

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

You can change the color of the letters which are printed while the program is running by poking a number from 0 through 15 in that box.

Add these lines to the above program:

```
60 INPUT"WHAT COLOR LETTERS <0 TO 15>";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 press CTRL and a color key anyway, and the printing will appear again in some color or another!

JUMPING NAME

```
10 REM JUMPING NAME
15 PRINT "clr dn dn dn"
16 POKE 53281,0      :REM BLACK SCREEN
20 CB=646      :REM CHAR. COLOR BOX
30 INPUT "NAME";N$
32 INPUT "HOW MANY LETTERS IN IT";N
39 PRINT "clr";
40 FOR I=1 TO RND(8)*24      :REM MOVE DOWN
45 PRINT "dn";:NEXT I
50 FOR I=1 TO RND(8)*39-N      :REM MOVE ACROSS
55 PRINT "rt";:NEXT I
57 NC=INT(RND(8)*8)      :REM RANDOM COLOR
58 IF NC=0 THEN 57      :REM NOT BLACK
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 "black."
We do not want black letters on a black 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 which 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 box number 646.

4. Write a program which draws your country's flag.

INSTRUCTOR NOTES 22 POKEing 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 1024 continuously to 2023, and the color cells from 55296 to 56295. A row, column method of addressing is clearer, and a program using addressing in that form is included in the lesson.

The advantages 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 1024. Where on the screen is the character at address 1027 shown?

2. What address is the color box for this character?

3. What color numbers may be put into this box?

LESSON 22 POKEing GRAPHICS

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

THE SCREEN IS A MAP OF MEMORY BOXES

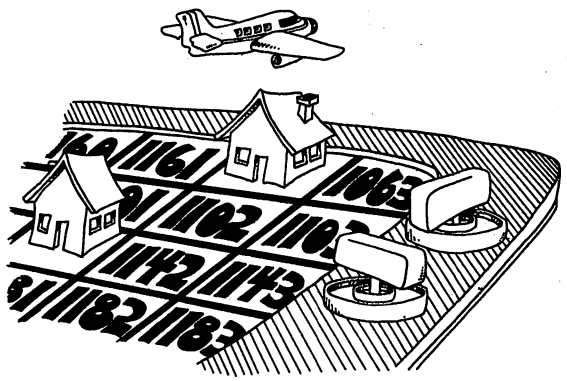
Imagine a little town with 25 streets. Each street has 40 houses on it.

Each house is a box in memory and can hold one character.

The streets are the 25 lines, counting down the screen. There are 40 boxes counting across a line.

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

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



MOVING A CHARACTER INTO A HOUSE

Enter this: POKE 1Ø24,83:POKE 55296,2

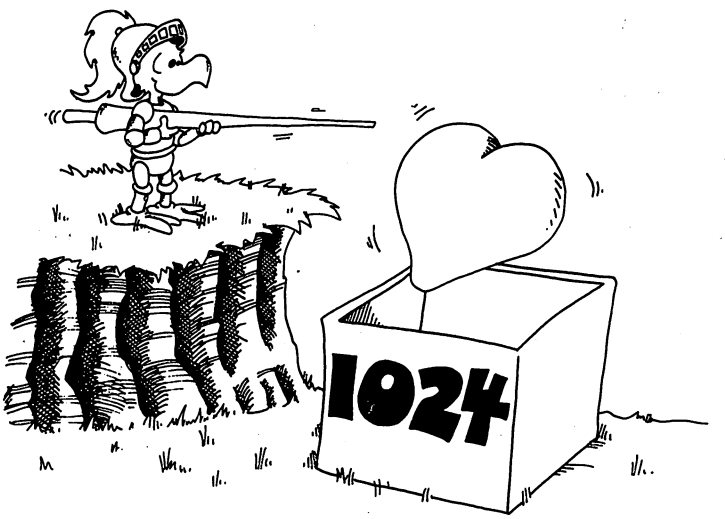
You will see a red heart in the "home" position on the screen (upper left corner). WHAT HAPPENED? We POKEd the heart (character number 83) into the memory box number 1024 which shows in the home position on the screen.

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

Then we POKEd a color, red, into another memory box (55296) which gives the color of the character in the home position.

Try this: Change the number 83 to any other number from 64 through 127 and run it again.

Change to another color number from 0 to 15.



ADDRESSES ON THE SCREEN

Each house has an address.

The box we call the "home" of the cursor has the lowest address. It is 1024. The label on the front of the box is 1024.

The next box to the right is 1025, and the numbers increase across the screen and down.

The highest number belongs in the lower right corner, address 2023.

THE WHOLE TOWN

Enter

```
10 REM EVERY THIRD HOUSE
12 POKE 53281,0      :REM BLACK BACKGROUND
15 PRINT"clr wht"    :REM CLEAR SCREEN
20 D$="                " (40 spaces)
22 FOR I=1 TO 25:PRINT D$:NEXT I
25 FOR I=1024 TO 2023 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 before running this program.

This time all the characters were white because the WHT character was printed in line 15 and spaces in line 22.

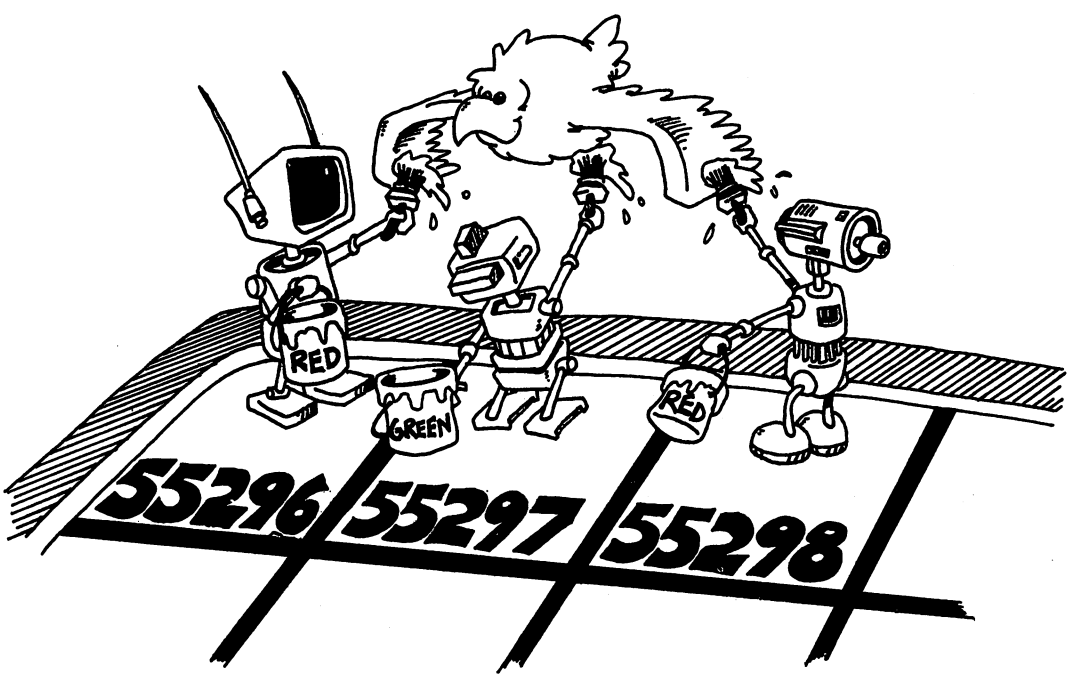
If you want another color, you have to POKE a number into the correct color map box.

Change the above program.

Enter
```
20
22
65 POKE I-1024+55296,RND(8)*7+1:REM COLOR BOXES
```

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

```
30 CH=INT(RND(8)*64)
```



COORDINATE NUMBERS

Numbers like 1024 and 55296 are hard to remember.

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

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

Street 24 at the bottom of the screen

We number the houses on each street:

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

House 39 at the right end of the street

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

screen address = 1024 + 40 * street number + house number

color address = 55296 + 40 * street number + house number

(See line 50 below.)

Enter

```
10 REM TOWN MAP
15 SC= 1024      :REM SCREEN CORNER
16 CC=55296      :REM COLOR CORNER
17 B$="hm dn sp sp ... sp"          (16 spaces)
20 PRINT"clr";
30 INPUT"hm dn WHICH STREET";S
35 PRINT B$;     :REM ERASE WRITING
40 INPUT"hm dn WHICH HOUSE";H
43 PRINT B$;
45 C=5      :REM COLOR NUMBER 5
50 AD = 40*S + H      :REM ADDRESS
55 POKE AD+SC,102      :REM POKE SCREEN CELL
56 POKE AD+CC,C      :REM POKE COLOR CELL
57 FOR T=1 TO 2000:NEXT T
80 GOTO 30
```

Save to tape or disk, then run.

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

Assignment 22:

1. Write a program which 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 one to six spots on it like dice.

3. SINBAD'S MAGIC CARPET program is shown below. You improve it. Changing the formula in line 215 gives different carpets. Add a "super" outside loop which draws one carpet after another, each changed a little in color design. Do this by putting variables into line 215 which are changed in the super loop.

```
10 REM SINBAD'S CARPET
20 PRINT "{CLR}"
100 CC=55296
105 C=CC+500
110 SC=1024
115 CH=96+128
150 FOR I=SC TO SC+999
155 POKE I,CH
160 NEXT I
210 FOR I=1 TO 11:FOR J=0 TO 18
212 R=(I+J)*7/22
215 K=K+(R+3)/(R+5)
216 IF K>15 THEN K=0
220 POKE C+I*40+J,K
222 POKE C-I*40+J,K
224 POKE C+I*40-J,K
226 POKE C-I*40-J,K
250 NEXT J,I
```

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 GETs 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 into the line buffer for a character. If it finds one, it returns it to the variable. (If the variable is numeric and the keystroke was not, a ?SYNTAX ERROR is printed.)

If the line buffer is empty, it returns zero to a numeric 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 into a loop, and keep looking at the buffer until a non-empty string is present.

The line buffer has a "queue" up to 10 characters long. So for 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 into 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

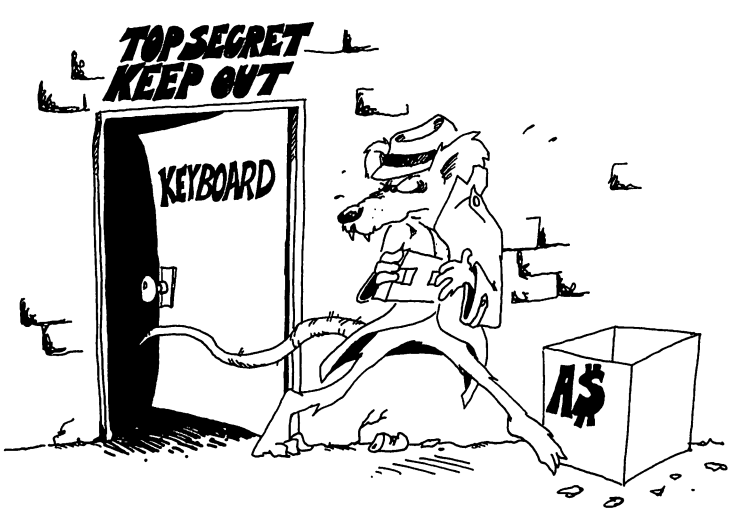
The GET command is different from INPUT. It gets a single character from the keyboard. Example:

```
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"dn wht 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 ##
15 POKE 53281,0     :REM BLACK SCREEN
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$=""THEN 40
50 IF L$="." THEN 80:REM TEST FOR END
60 W$=L$+W$         :REM ADD LETTER TO FRONT OF WORD
65 GOTO 40          :REM TO GET ANOTHER LETTER
80 REM WORD IS FINISHED
82 PRINT "clr dn dn dn dn grn HERE IT IS BACKWARDS"
85 PRINT "dn pur sp";W$;"wht"
```

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:

Enter and run
```
10 GET N:IF N=0 THEN 10
15 PRINT N
20 GOTO 10
```

Press number keys. Now press a letter key. You see

```
?SYNTAX ERROR
```

when you try to GET a letter character in a numeric variable.

Assignment 23:

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

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

   The program changes the border color to the user's choice.

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

INSTRUCTOR NOTES 24 PRETTY PROGRAMS, GOSUB, RETURN, END

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

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

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

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

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

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 into 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:

```
100 REM TAKE A TRIP
101 :
105 POKE 53281,0
110 PRINT "clr HOP TO THE SUBROUTINE dn"
120 GOSUB 200
130 PRINT "grn BACK FROM THE SUBROUTINE dn"
133 FOR T=1 TO 1000:NEXT T
135 PRINT "blu HOP AGAIN dn"
140 GOSUB 200
150 PRINT "cyn HOME FOR GOOD dn wht"
190 END
199 :
200 REM SUBROUTINE
201 :
210 PRINT "red GOT HERE OK dn"
215 FOR T=1 TO 1000:NEXT T
220 PRINT "yel PACK YOUR BAGS, BACK WE GO dn"
230 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.

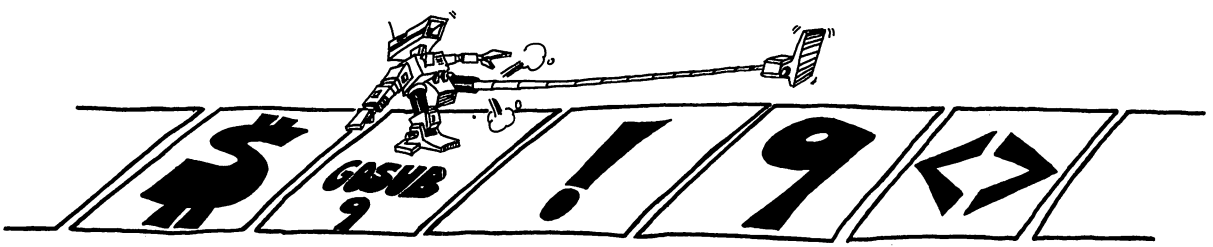
Where there are PRINT commands, you may put in many more program lines.

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 to the commands in the subroutine, does them and then comes back.

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

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



Assignment 24A:

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

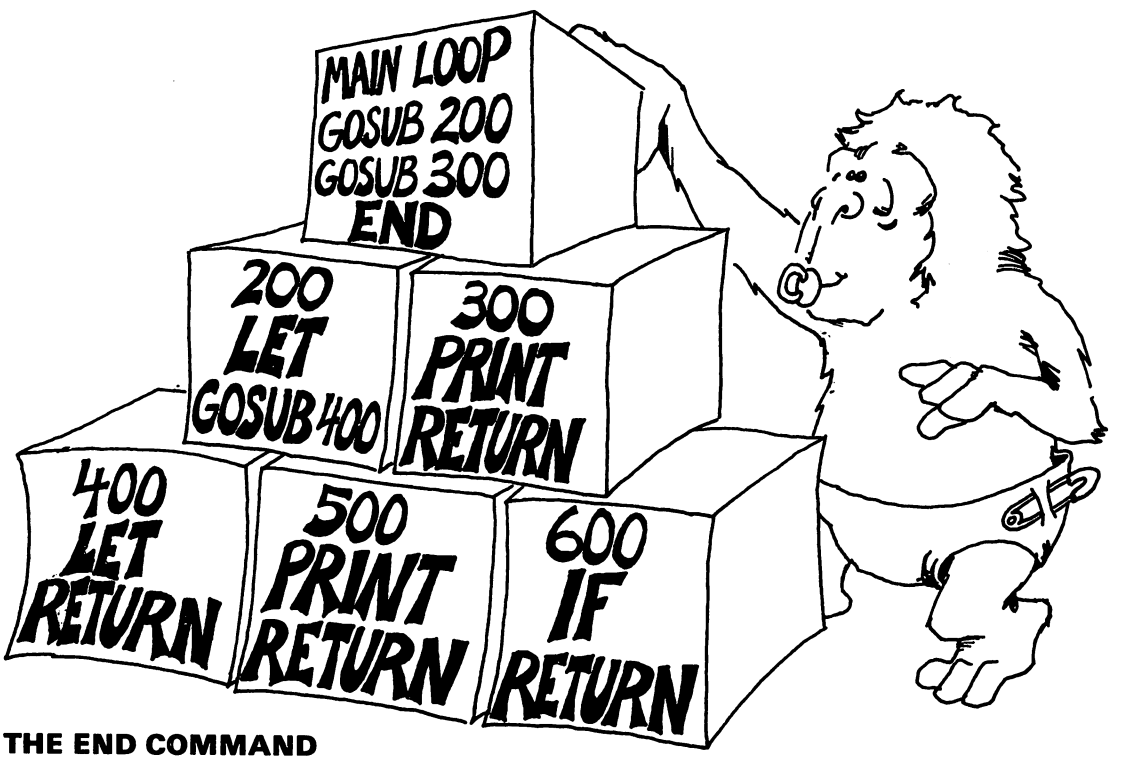
WHAT GOOD IS A SUBROUTINE?

In a short program, not much 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 POKE 53281,0:PRINT"red clr"
20 X=10:Y=10:D=1
25 FOR J=1 TO 6: FOR I=1 TO 6
29 A$="rev sp off":REM      INVERSE SPACE
30 GOSUB 100:REM      DRAW
34 A$=" ":REM      SPACE
35 GOSUB 100:REM      ERASE
40 Y = Y - D:REM      MOVE
50 NEXT I
55 D = -D:REM      REVERSE DIRECTION
60 NEXT J
80 PRINT "wht"
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$
111 NEXT K
120 PRINT TAB(X);A$;" ";A$
130 PRINT TAB(X);" ";A$;A$
190 RETURN
```

Save to tape or disk

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

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

Assignment 24B:

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

   Call one of them twice from the main program.

   Call one of them from another of the subroutines.

   Call one of them from an IF statement.

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

   Change the subroutine so it prints your own initial.

   Change the color of your initial to blue.

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

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

   Change the distance the slide goes to eight steps instead of six.

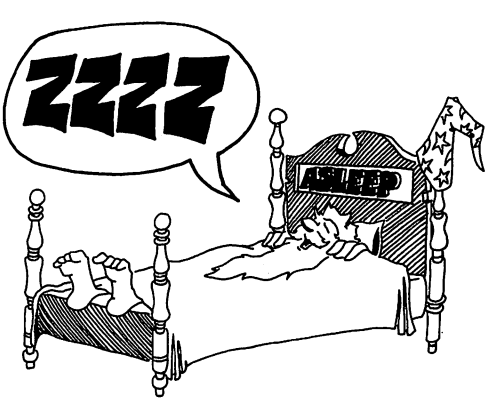
   Change the size of each step from one to two.

   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.

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



INSTRUCTOR NOTES 25 LOGIC: AND, OR, NOT

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

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

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

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

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

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

QUESTIONS:

1. For each IF statement, tell if anything will be printed:

```
10 IF 3=3                    THEN PRINT "HI"
10 IF NOT(3=3)               THEN PRINT "HI"
10 IF 3=3 OR 0=2             THEN PRINT "HI"
10 IF 3=3 AND 0=2            THEN PRINT "HI"
10 IF "A"="B"                THEN PRINT "HI"
10 IF NOT("A"="B")           THEN PRINT "HI"
```

2. What number will each of these lines print?

```
10 A=-1                      PRINT A, NOT A
10 A= 0                      PRINT A, NOT A
10 A=-1:B=-1                 PRINT A AND B
10 A= 0:B=-1                 PRINT A AND B
10 A= 0:B= 0                 PRINT A AND B
10 A= 0:B=-1                 PRINT A OR B
10 A= 0:B 0                  PRINT A OR B
10 PRINT NOT 0
```

LESSON 25 LOGIC: AND, OR, NOT

ANOTHER TEENAGER PROGRAM

```
10 REM < AND, OR, NOT >
20 POKE 53281,0:PRINT "clr dn"
30 INPUT " NAME";N$
35 PRINT
40 INPUT " AGE";A
45 PRINT
50 IF (A>12) AND (A<20) THEN PRINT "sp"; N$;"IS A TEENAGER.dn"
55 NFLAG = (A<13) OR (A>19)
60 IF NFLAG THEN PRINT "sp";N$;"IS NOT
A TEENAGER! dn"
70 IF (NOT NFLAG) AND (A=16) THEN PRINT "AND";N$;:PRINT"dn IS cyn
SWEET SIXTEEN ! wht 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. It is, so the computer prints "-1" meaning TRUE.

PUTTING TRUE AND FALSE INTO BOXES

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

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

The number 0 is stored in box N because 3=22 is FALSE.

And this:

```
10 N= "B"="B"
20 PRINT N
```

The number -1 is stored in 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 (phrase A) THEN (command C)
```

Try these in the edit mode:

```
IF 0 THEN PRINT "TRUE"
IF -1 THEN PRINT "TRUE"
```

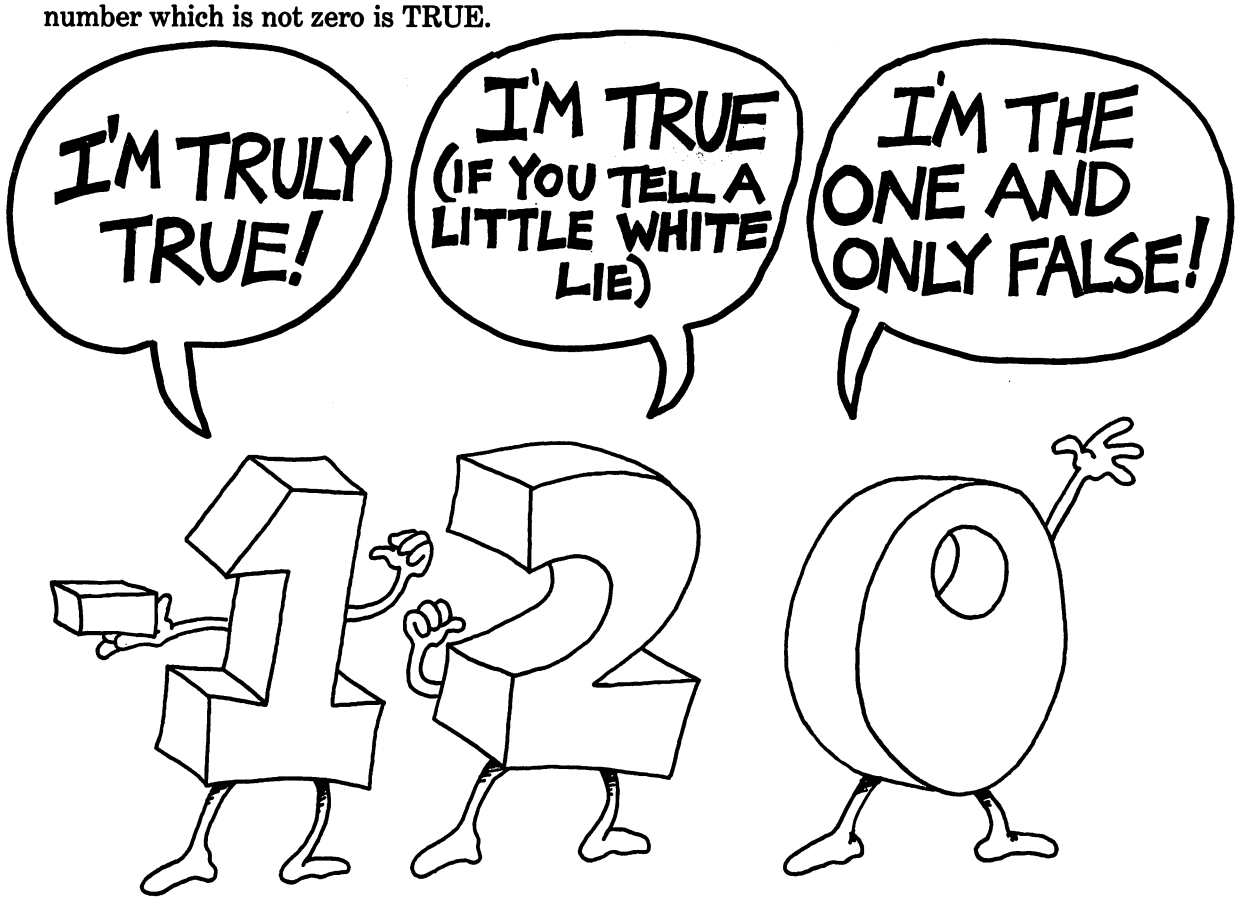
Now try this: `IF 22 THEN PRINT "TRUE"`

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

If it is zero, the computer says "phrase A is FALSE," and skips what is after THEN.

If it is not zero, the computer says "phrase 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 which is not FALSE," that is, any number which is not zero is TRUE.

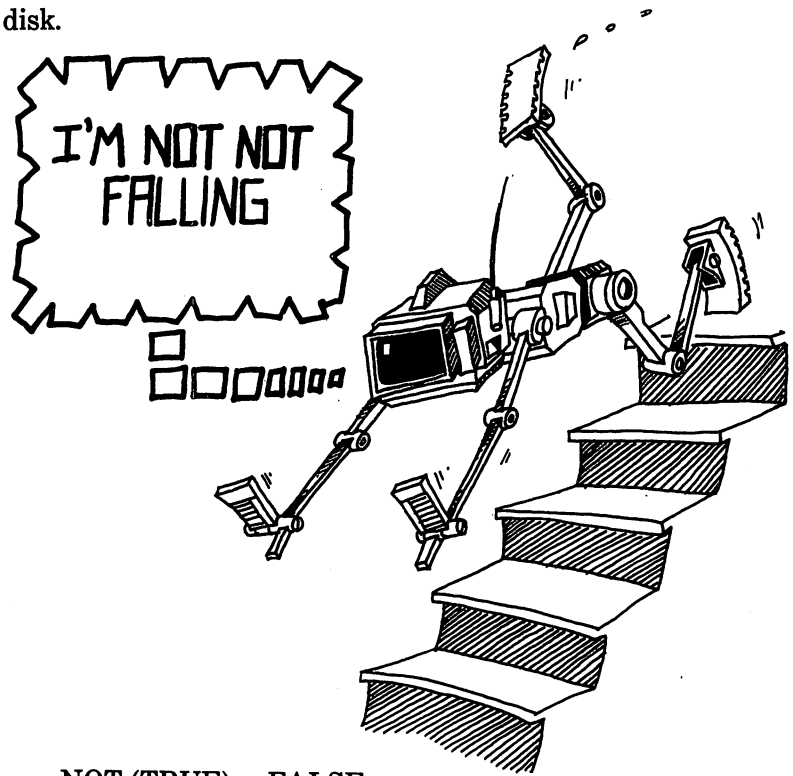


WHAT DOES "NOT" MEAN?

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 or disk.



CAREFUL! NOT (TRUE) = FALSE
and NOT (FALSE) = TRUE

works only 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 (NOT 3) doesn't give 0.

THE LOGIC SIGNS

You can use these six symbols in a "phrase A":

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

You have to press two keys to make the < > sign, the <= sign and the >= sign.
The last two are new so look at this example to see the difference between < and <=:

2<=3 is TRUE	2<3 is TRUE
3<=3 is TRUE	3<3 is FALSE
4<=3 is FALSE	4<3 is FALSE

These two "phrase A" phrases mean the same:

2<=Q	(2<Q) OR (2=Q)

Assignment 25:

1. Tell what will be found in 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 26 SNIPPING STRINGS: LEFT$, MID$, RIGHT$, LEN

In this lesson the functions:

LEFT$ MID$ RIGHT$ LEN

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

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

As in earlier explanations, the main characteristics of the functions are shown, but not all the special cases, especially those which 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 which 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

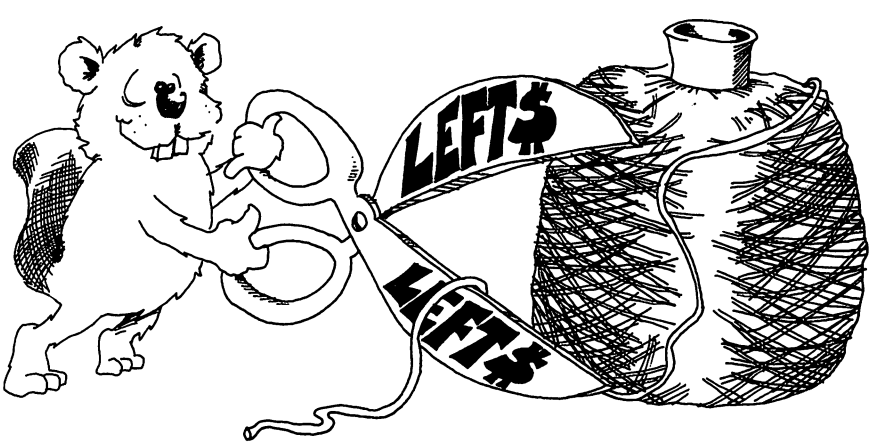
Let's cut a piece off a string. Enter and run:

```
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 into 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 from 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 AND GLUE :::
20 PRINT"clr"
30 N$="THE CAT'S MEOW"
35 FOR I=1 TO 9
40 L$=LEFT$(N$,I) : R$=RIGHT$(N$,I)
50 PRINT I;L$+R$
60 NEXT I
```

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

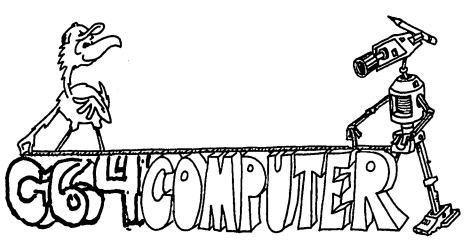
```
55 IF I=4 THEN PRINT: PRINT R$ + L$ :PRINT
```

HOW LONG IS THE STRING?

Run:

```
10 REM : HOW LONG? :
20 PRINT "clr"
30 INPUT " GIVE ME A STRING: ";N$
35 PRINT "clr"
40 L=LEN(N$)
50 PRINT " THE STRING: dn"
52 PRINT "";N$;"'dn"
55 PRINT " IS";L;"CHARACTERS LONG"
```

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



CUTTING A PIECE OUT OF THE MIDDLE

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

Run:
```
10 REM ### MIDDLE ###
20 PRINT"clr"
30 N$="WQXEMIDUICXZ"
40 P$= MID$(N$,5,3)
45 PRINT P$
```

In line 40, P$= MID$(N$,5,3) means

Get the string from box N$.

Count over five letters and start saving letters into box P$.

Save three letters.

LOOK MA, NO SPACES

Enter
```
10 REM ------------------------- <NO SPACES>
20 PRINT "clr dn dn dn"
21 PRINT " GIVE ME A LONG SENTENCE dn"
35 INPUT S$
40 L=LEN(S$)
45 T$=""
50 FOR I=1 TO L                 :REM RUN THROUGH THE WORDS
60 L$=MID$(S$,I,1)              :REM LOOK AT EACH LETTER
70 IF L$<>" " THEN T$=T$+TL$:    REM SAVE ONLY LETTERS
80 NEXT I
90 PRINT"clr dn dn";T$
```

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

A REAL CLOCK

The C-64 has a clock which is always running. The time is kept in a string named TI$. There are six digits in the string. You have to split them apart to make the clock easy to read. You can reset it with a LET command.

TI$="123456" means 12 hours, 34 minutes and 56 seconds

```
Run     10000 REM ::: CLOCK :::
        10020 TI$="120000":REM        NOON
        10025 PRINT "clr"
        10029 PRINT "hm dn dn rt rt";
        10030 PRINT LEFT$(TI$,2);":";MID$(TI$,3,2);":";RIGHT$(TI$,2)
        10035 GOTO 10029
```

You set the clock with an INPUT TI$. We used large line numbers so you can put these lines into programs which need a time of day clock. The clock runs off the "jiffy clock" and will stop during the time you do tape LOAD and SAVE.

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 forth, 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 takes all the letters up to the first vowel and puts them on the back of the word, followed by AY. If the word starts with a vowel, it only adds LAY. Examples:

   TURKEY becomes URKEYTAY
   ADAM becomes ADAMLAY

4. Write a program which speaks "double dutch." It asks for a sentence, then removes all the vowels and prints it out.

5. Write a program which 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.

6. Add to the clock program so the user can set it. You need an

   INPUT TI$

   statement. Don't forget to help the user with PRINTed instructions.

INSTRUCTOR NOTES 27 SWITCHING NUMBERS WITH STRINGS

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

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

VAL does just the opposite, taking a string and making a numeric value from it. It accepts decimal and "scientific" notation (e.g., "1.2 E+31"). If the first character is not a decimal digit, or + or –, it returns the value "0". Otherwise, it scans the number, terminating at the first non-numeric character (other than the E of the scientific notation).

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

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

QUESTIONS:

1. If your number marches too quickly in problem 2 of the assignment, how do you slow it down?

2. If your program has the string "IN 1732 GEORGE WASHINGTON WAS BORN", write a few lines to answer the question "How long ago was Washington born?" (You need to get the 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. How do you convert the string "1999" to a number you can use in arithmetic?

5. What is an argument of a function? How many arguments does the MID$( ) function have? How many for the CHR$( ) function?

6. Where in the line do commands always go? Can you put a function at the start of a line?

7. What will you see if your program has the line:

```
65 PRINT TI$
```

LESSON 27 SWITCHING NUMBERS WITH STRINGS

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

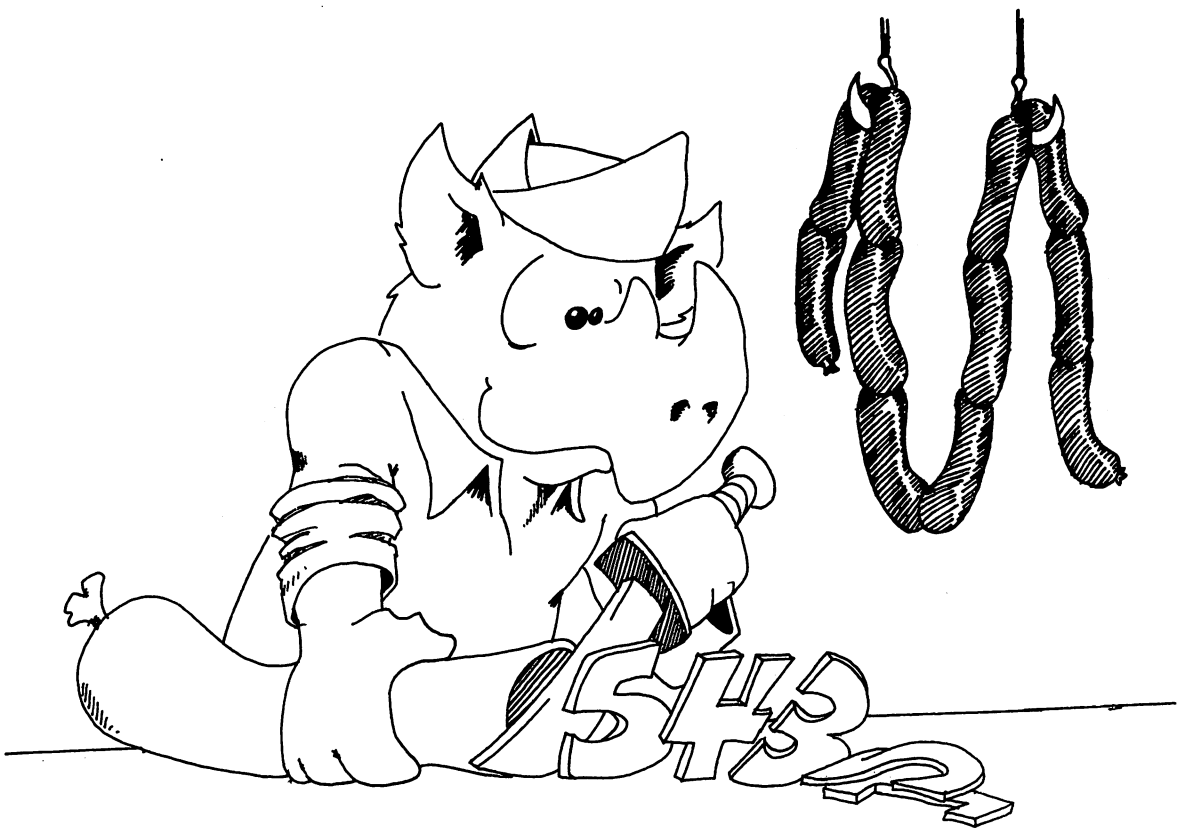
MAKING STRINGS INTO NUMBERS

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

Run

```
10 REM STRINGS INTO NUMBERS
20 PRINT"clr"
30 L$="123" :M$="789"
50 L=VAL(L$):M=VAL(M$)
70 PRINT L
72 PRINT M
74 PRINT " ---"
76 PRINT L+M
```

VAL stands for value. It changes a string to a number, if it can.

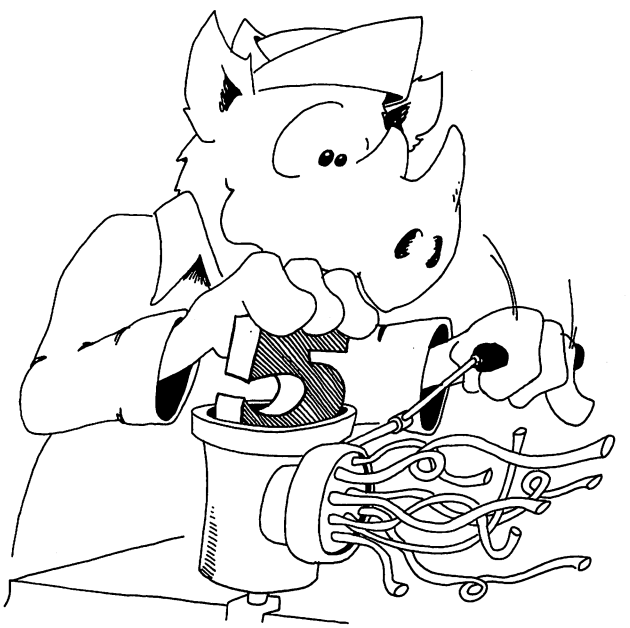


MAKING NUMBERS INTO STRINGS

Run

```
10 REM NUMBERS INTO STRINGS
20 PRINT "clr"
21 POKE 53281,0
25 INPUT " GIVE ME A NUMBER ";NB
30 N$=STR$(NB)
35 L=LEN(N$)
40 FOR I=L TO 1 STEP -1
45 B$=B$+MID$(N$,I,1)
50 NEXT I
60 PRINT "cyn dn HERE IT IS BACKWARDS dn"
65 PRINT "dn ";B$
```

STR$ stands for string. It changes a number to 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 into 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 into string variable boxes, numeric values go into numeric boxes.)

PRACTICE WITH FUNCTIONS

For each function in the list below:

Give the names of the arguments, and tell whether the argument value is a number or a string. Tell if the argument is a constant, variable or function.

RND(8)	fn _____
	arg _____
RIGHT$(U$,Y)	fn _____
	arg _____
	arg _____
VAL("231")	fn _____
	arg _____
STR$(INT(RND(8)))	fn _____
	arg _____

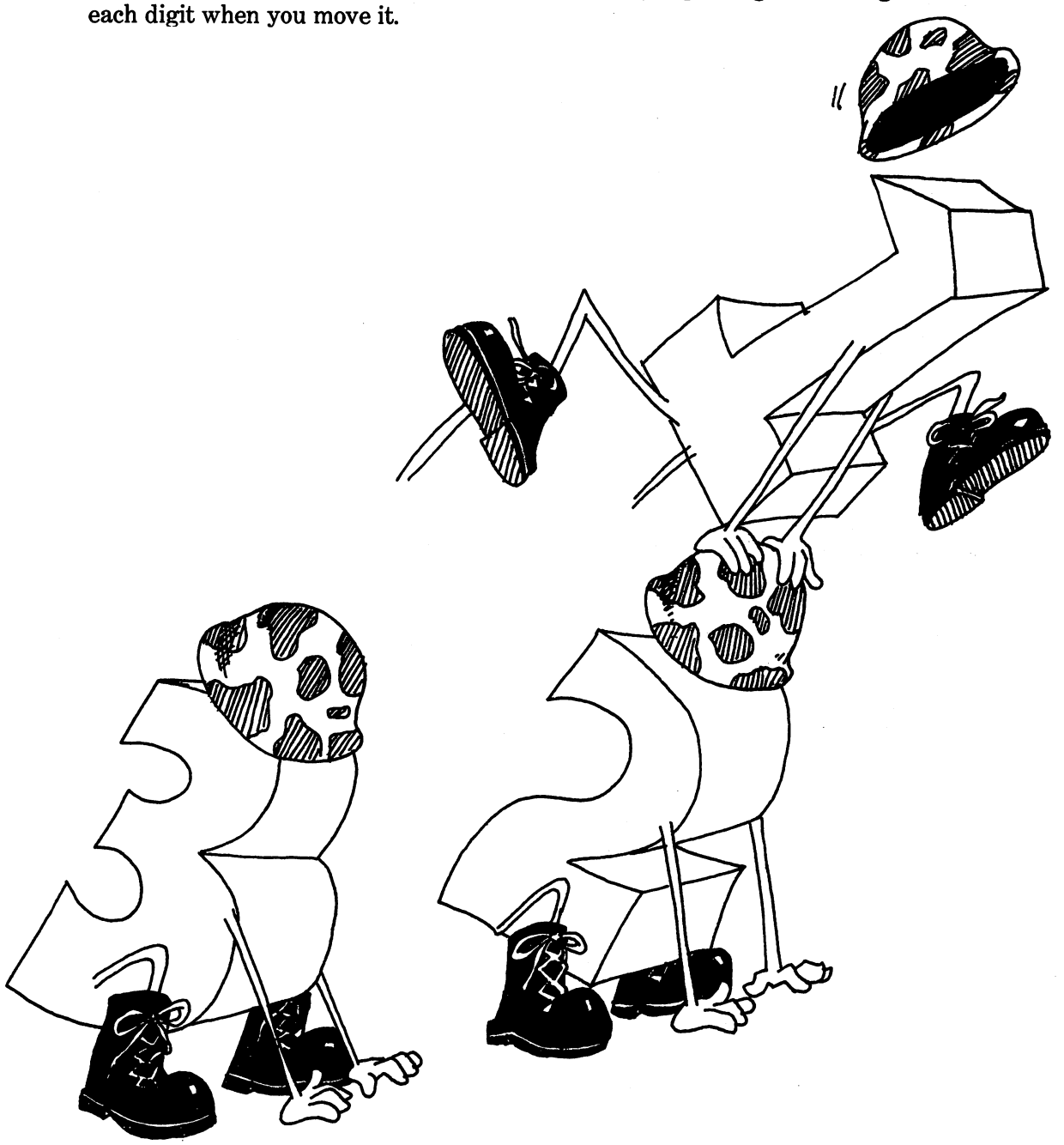
Each line below has errors. Explain what is wrong.

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

Assignment 27:

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

2. Make a number play leapfrog slowly across the screen. That is, write it on the screen, then take its first digit and move it to the end. 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 a somewhat abstract 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 which 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 numeric constants to floating point as the program runs. That is why lines 55, 60 and 65 compare K to variables F1, F3 and Z0 rather than to 4, 5 and 64. This practice is probably the single most important factor in obtaining fast programs.

The fastest and most versatile way to make graphics is to use the "sprites" in the C-64. This topic is covered later in this book.

QUESTIONS:

1. What ASCII numbers do the function keys have?

2. How can you detect if a function key has been pressed?

3. How can you detect if 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?

6. Explain how the computer knows to move the arrow and the diamond at the same time.

LESSON 28 ACTION GAMES AND THE FUNCTION KEYS

THE FUNCTION KEYS

The four large keys on the right side of the C-64 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
15 D$="clr dn dn dn dn dn sp"
16 PRINT D$
20 F1$=CHR$(133)
30 GET K$:IF K$="" THEN 30
40 IF K$=F1$ THEN PRINT D$;"YOU PRESSED F1":GOTO 30
50 PRINT D$;K$;
60 GOTO 30
```

(Remember: "sp" means push the space bar.)

Run it. Press any key and see that it is printed to the screen. But when you press the f1 key, the special message gets printed.

Notice that trying to PRINT character number 133 gives nothing. The eight characters which are function keys do not print.

USING THE FUNCTION KEYS IN PROGRAMS

First you GET a one-character string from the keyboard. If it has an 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:

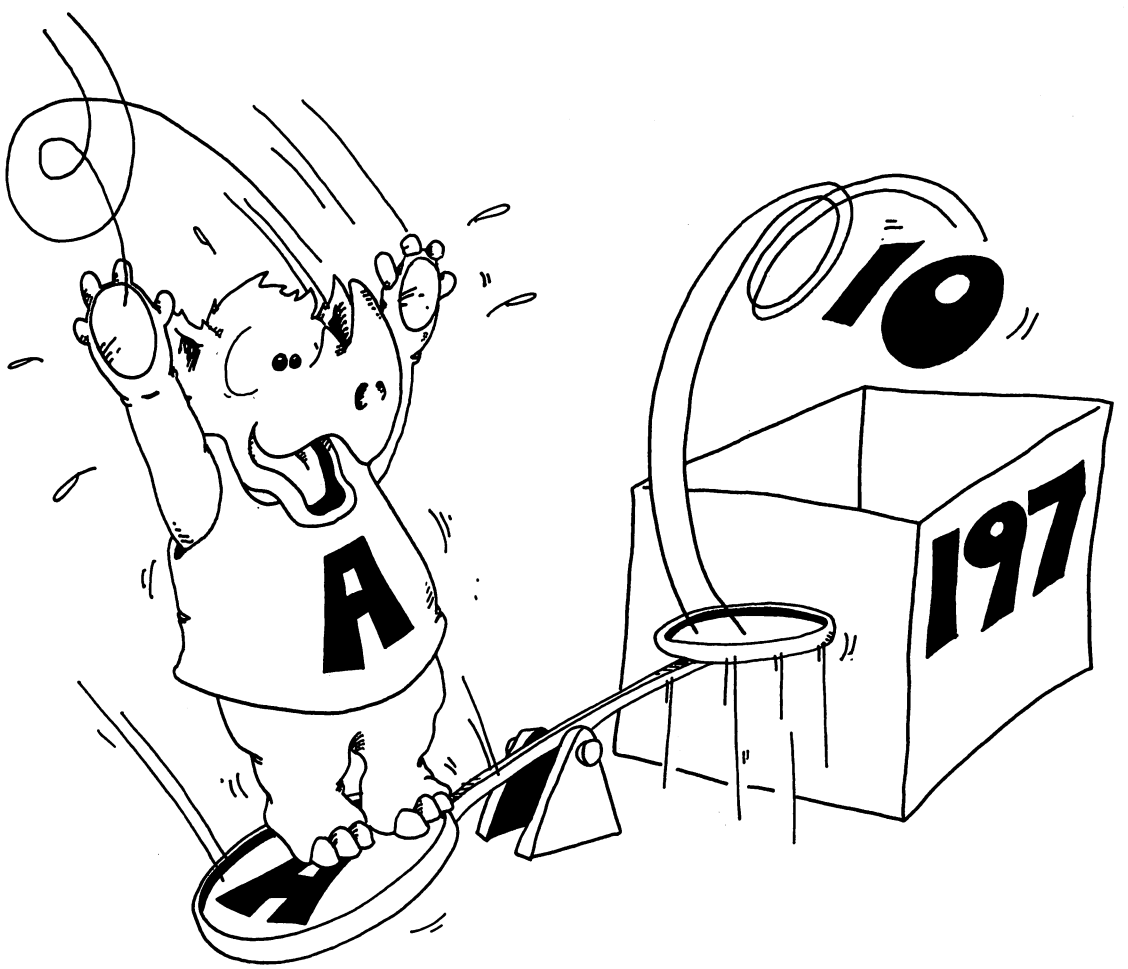
```
12 C=2
40 IF K$=F1$ THEN GOSUB 200
200 REM SCREEN COLOR CHANGE
210 POKE 53281,C
220 C=C+1:IF C=15 THEN C=2
299 RETURN
```

Now the f1 key sends you to a subroutine which changes the 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 which 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
```

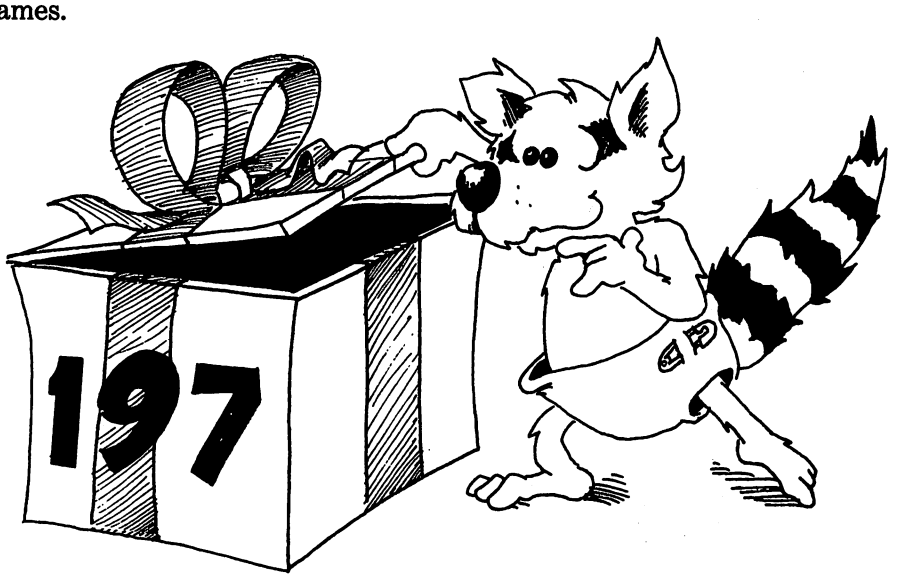
Notice that the number keeps printing as long as you hold the key down.

Try holding two keys down at once. The bigger number is the one which is in the box.

The only keys which do not put numbers into the box are CTRL, RUN STOP, RESTORE, 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 15, 52, 58, 60, 61 and 63.) This table will be useful when you design games.



MOVING A DIAMOND

Make the f1 and f3 keys control a moving diamond.

It moves up while you press the f1 key.

It moves down while you press the f3 key.
It stops if you don't press either key.

```
10 REM CUPID
11 PRINT "clr"
12 POKE 53281,2                    :REM RED SCREEN
14 D$="sp sp ... sp"                 (40 spaces)
15 FOR I=1 TO 24:PRINT D$;:NEXT I
20 F1=4:F3=5:Z0=64                 :REM KEY NUMBERS
22 SB=1982                         :REM SCREEN BOTTOM
24 SC=1102                         :REM SCREEN CORNER
26 Y =SC                           :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=-40              :REM MOVE UP?
65 IF K=F3 THEN D= 40              :REM MOVE DOWN?
90 POKE Y,32                       :REM ERASE OLD DIAMOND
120 Y=Y+D                          :REM SPOT FOR NEW ONE
130 IF Y>SB THEN Y=SB              :REM OFF BOTTOM?
140 IF Y<SC THEN Y=SC              :REM OFF TOP?
170 POKE Y,90                      :REM POKE NEW DIAMOND
199 GOTO 50                        :REM DO AGAIN
```

Save 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 into 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 onto the screen.

Line 60 tests to see if the f1 key is being pressed. If it is, D is set to -40. This means the next diamond will be put one line higher than the present diamond.

Line 65 sets D to 40 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) onto its screen spot.

Lines 130 and 140 do another bit of housekeeping. They check the value of Y (the address of the diamond on the screen) to make sure that the diamond will not try to move off the top or bottom of the screen.

Line 170 POKEs the new diamond onto the screen, and the cycle starts again.

SHOOTING ARROWS

We want the diamond to shoot arrows when we press the "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=Ø:X=Y
7Ø IF K=62 AND S=Ø THEN S=1
75 IF S< >Ø THEN GOSUB 3ØØ
3ØØ REM SHOOT
31Ø IF S=1 THEN X=Y-1:POKE X,31:S=2:RETURN
315 POKE X,32
316 IF S=38 THEN S=Ø:RETURN
32Ø X=X-1:S=S+1
33Ø 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:

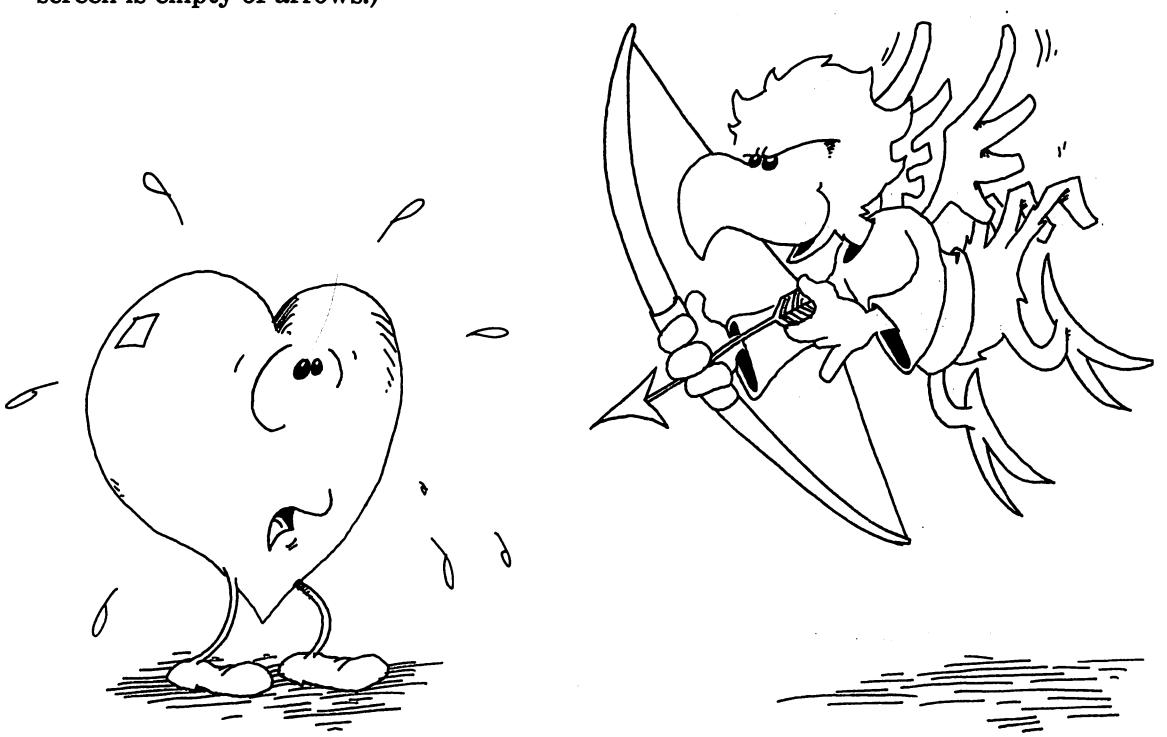
First, is this a new arrow or just one which already started across?

> 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=38, the arrow has moved 38 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=1024 + INT(RND(1)*23+1)*40
420 I=INT(RND(1)*36+1)
430 POKE I+J,83
450 NEXT L
499 RETURN
```

Save and then run it.

The arrows hit the hearts and erase them.

POPPING THE TARGETS

Add this line:

```
325 IF PEEK(X)=83 THEN S=0:POKE X,91:RETURN
```

PEEK to see if the arrow is about to hit a target.

PEEK(X) tells the number of the character at the spot the arrow is about to move on to. If the spot has number 83, it is a heart. Then pop it by 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 in which the diamond moves.

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 the diamond! Add missiles which try to hit the diamond.

INSTRUCTOR NOTES 29 MUSIC

This lesson continues the introduction of SID chip features with applications to making music.

Review the lesson on sound effects. DATA statements are used and may also need to be reviewed.

We treat the ADSR parameters (attack, decay, sustain and release) in some detail. The ADSR parameters define a loudness envelope or contour for the duration of the note.

Each parameter is one nybble in length and so two parameters can share one SID chip register. A and D share one register, S and R share another.

This is how it is done. A nybble is a number from 0 to 15. Memory is stored as bytes (a number from 0 to 255). So each byte holds exactly two nybbles and can be represented as 16*H + L where H is the high nybble and L is the low nybble.

Three of the ADSR parameters are times (A,D,R). The S, sustain, is a volume or loudness value. A table of time values for A,D, and R is given. Note that the time is not proportional to the parameter value. A value of 5 gives an attack rise time of 0.056 seconds, but a value of 10 gives 0.5 seconds. A table of time values is included in the lesson. Times up to 24 seconds are possible. Such long times are not useful in ordinary music but are useful in sound effects.

We refer the reader to the C-64 Programmer's Reference Guide for two of the most advanced features of SID: the programmable filter and the fact that voice 3 can be used to control some feature of voices 1 and 2, for example the frequency, for vibrato or glissando.

SID can also accept audio from an external source (tape recorder, etc.), process it with the filter and add it to the internal sound. The output can be re-recorded, so you can build up a rich polyphonic piece in this way.

QUESTIONS:

1. What kind of sound will you get if you set the attack and decay each to the value 2, and sustain and release to 0?

2. How do you store the pitch and length of each note of music in the program?

3. How do you put the attack and decay numbers into the same POKE statement?

4. How many voices does SID have?

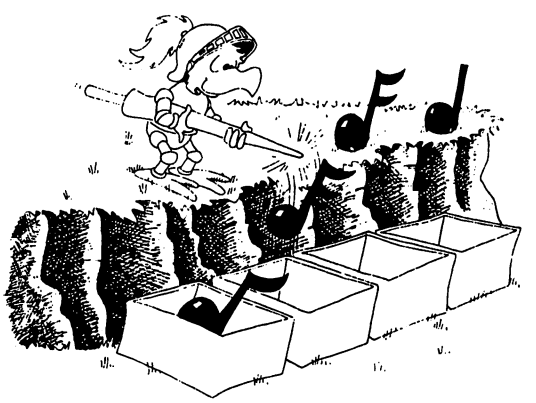
5. How would you change the program to use voice 2 instead of voice 1? Would it sound the same?

LESSON 29 MUSIC

Go back and review the SOUND EFFECTS lesson. It tells a lot about the SID sound chip. Also review DATA in Lesson 18.

SID PLAYS A TUNE

```
10 REM          ROW YOUR BOAT
12 C=54272:REM                          BASE ADDR
15 FOR I=C TO C+24:POKE I,0:NEXT
20 POKE C+5,9:POKE C+6,0:REM            ADSR
30 POKE C+24,15:REM                     VOLUME
40 READ H,L,D:REM                       GET NOTE
50 IF H<0 THEN POKE C+24,0:             END
60 POKE C+1,H:POKE C,L:REM              PITCH
70 POKE C+4,33:REM                      START NOTE
80 FOR T=1 TO D:NEXT:REM                HOLD NOTE
90 POKE C+4,32:REM                      END NOTE
95 GOTO 40
100 REM EACH LINE HOLDS TWO NOTES
105 REM NOTE: H, L, D
110 DATA 16,195,750,16,197,750
120 DATA 16,195,500,18,209,250
130 DATA 21, 31,750,21, 31,500
140 DATA 18,209,250,21, 31,500
150 DATA 22, 96,250,25, 30,1500
160 DATA 33,135,250,33,135,250
170 DATA 33,135,250,25, 30,250
180 DATA 25, 30,250,25, 30,250
190 DATA 21, 31,250,21, 31,250
200 DATA 21, 31,250,16,195,250
210 DATA 16,195,250,16,195,250
220 DATA 25, 30,500,22, 96,250
230 DATA 21, 31,500,18,209,250
240 DATA 16,195,1500,-1,-1, -1
```



Save the program before running.

THE SCALE

Here are two octaves of the scale. More are in the User's Guide which came with your C-64 computer.

		HI	LO
middle	C	16	195
	C#	17	195
	D	18	209
	D#	19	239
	E	21	31
	F	22	96
	F#	23	181
	G	25	30
	G#	26	156
concert	A	28	49
	A#	29	223
	B	31	165
	C	33	135
	C#	35	134
	D	37	162
	D#	39	223
	E	42	62
	F	44	193
	F#	47	107
	G	50	60
	G#	53	57
	A	56	99
	A#	59	190
	B	63	75
high	C	67	15

THE THREE VOICES OF SID

SID can play three instruments at once, each different from the others. You need to POKE information into the registers of each voice you use. We will not demonstrate the three voices here, but give you the register addresses

REGISTER ADDRESSES Base address is 54272

Add these register addresses to the base address:

Register	voice 1	2	3	contents
LO	0	7	14	0 to 255
HI	1	8	15	0 to 255
Pulse low	2	9	16	0 to 255
Pulse high	3	10	17	0 to 15
Waveform	4	11	18	16, 32, 64, 96
Attack-Decay	5	12	19	16*(0 to 15) + (0 to 15)
Sustain-Release	6	13	20	16*(0 to 15) + (0 to 15)
Volume (Loudness)	24			0 to 15

ATTACK-DECAY-SUSTAIN-RELEASE

Some registers are split into two uses.

ATTACK and DECAY share one register. Why don't they trample each other's feet? Well, ATTACK stays in the attic, DECAY in the cellar. (Meaning, multiply the ATTACK number by 16, then add on the DECAY number.)

Likewise, SUSTAIN and RELEASE share another register.

Add

```
16 INPUT "ATTACK-DECAY <EACH Ø TO 9>";A,D
17 INPUT "SUSTAIN <Ø TO 15>";S
18 INPUT "RELEASE <Ø TO 7> ";R
2Ø POKE C+5,16*A+D:POKE C+6,16*S+R
8Ø FOR T=1 TO D*.5:NEXT T:REM ATTACK,DECAY,SUSTAIN TIME
92 FOR T=1 TO D*.3:NEXT T:REM RELEASE TIME
```

Save, then run.

Now you can play the same tune sound with different instruments.

Try these combinations:

A	D	S	R
2	2	0	0
9	9	0	0
9	9	15	1
2	2	3	7
0	0	15	0

Try other combinations. Change lines 70 and 90 to have other waveforms (see page ::::: in the SOUND EFFECTS lesson) and try them with different ADSR values.

AS THE NOTE GROWS OLDER

When turned on, the note rises to full loudness. How long it takes to get there is controlled by the ATTACK number. Zero means a "click" start while 8 gives a very gradual start.

After getting to full loudness, the note decays to its SUSTAIN loudness. How long it takes to decay is controlled by the DECAY number. Of course, if the SUSTAIN number S is 15, the note does not decay at all!

The SUSTAIN number tells how loud (compared to the maximum loudness) the note will stay until turned off, 15 is loud, zero is off.

When the note is turned off, it dies away to zero loudness in a time controlled by the RELEASE number. Line 92 in the program gives the note time to die away.

Without line 92, the next note would start before the first died away. But if you do not start another note (for example in a sound effect), you can do other computing while the sound continues to die away! A laser shot sound can slowly die away while the computer moves graphics around.

Here are the times for ATTACK, DECAY and RELEASE:

Number	Attack	Decay and Release
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

"ms" means "millisecond" or a thousandth of a second. So 500 ms is a half a second.

Assignment 29:

1. Change the ROW YOUR BOAT program to play another tune.

2. Change the ROW YOUR BOAT program to let the user pick the waveform.

3. Write an ELECTRIC ORGAN program. Use the top row of the keyboard as the organ keys. Let the user pick stops of different kinds of sounds.

INSTRUCTOR NOTES 30 ARRAYS AND THE DIM COMMAND

This lesson introduces arrays. The DIM( ) statement is described. Up to 255 indices are allowed.

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

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

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

One can use them to store lists of information. Connected lists also can occur. The telephone number program uses two linear arrays: one for names, the other for numbers. They are indexed the same, so a single index number can retrive both the name and the number which goes with it.

Another general use of arrays is to store numbers which cannot neatly be obtained from an equation. An example would be the length in days of the 12 months.

Games often use arrays to store information about the playing board.

If you forget to DIMension an array before use, the BASIC interpreter gives it a dimension of 10. If you try to use an element larger than that assigned to the array, the

```
?BAD SUBSCRIPT ERROR IN XX
```

message is printed.

QUESTIONS:

1. What does the DIM BD(6) statement do?

2. Where do you put the DIM command in the program?

3. What two kinds of array families are there?

4. What is the "index" or "subscript" of an array?

LESSON 30 ARRAYS AND THE DIM COMMAND

MEET THE ARRAY FAMILY

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

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

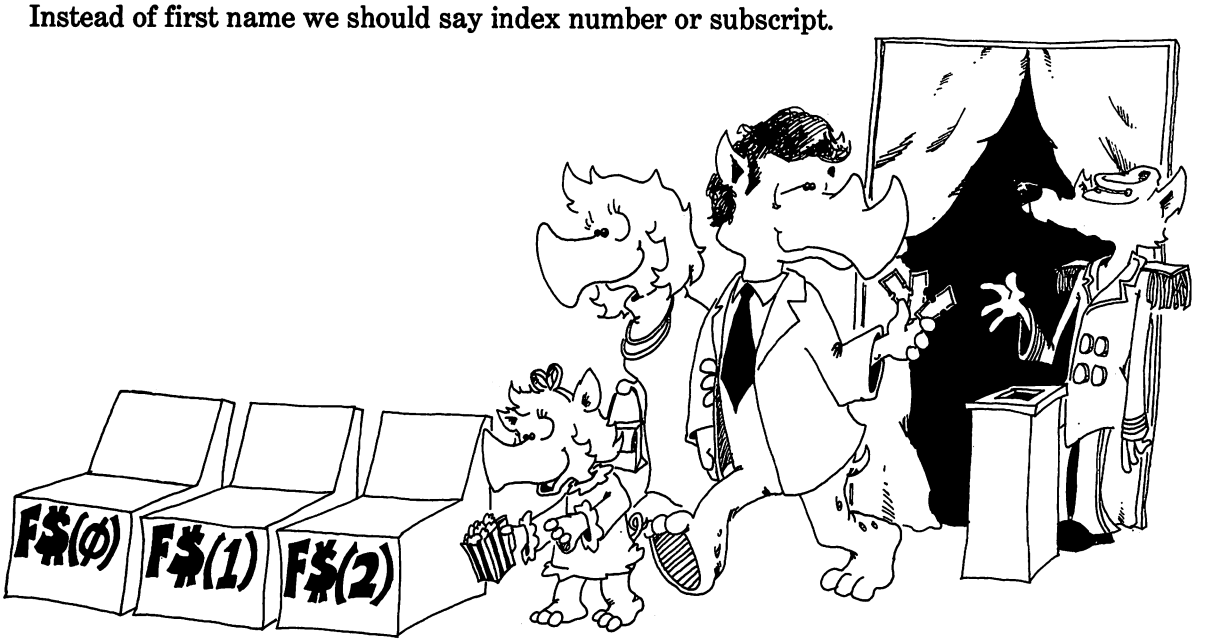
Here is a family of numeric variables:

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

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

```
18 DIM A(3)
30 DIM B(7),B$(4)
```

Line 18 saves four memory boxes, one each for the variables A(0), A(1), A(2) and A(3). These boxes are for numbers and contain the number "0" to start with.

Line 30 reserves eight boxes for the B() array and five for the string array B$(). The boxes named B$(0) through B$(4) are for strings and are empty to start with.

RULE: Put the DIM() statement early in the program, before the array is used in any other statement.

MAKING A LIST

Enter

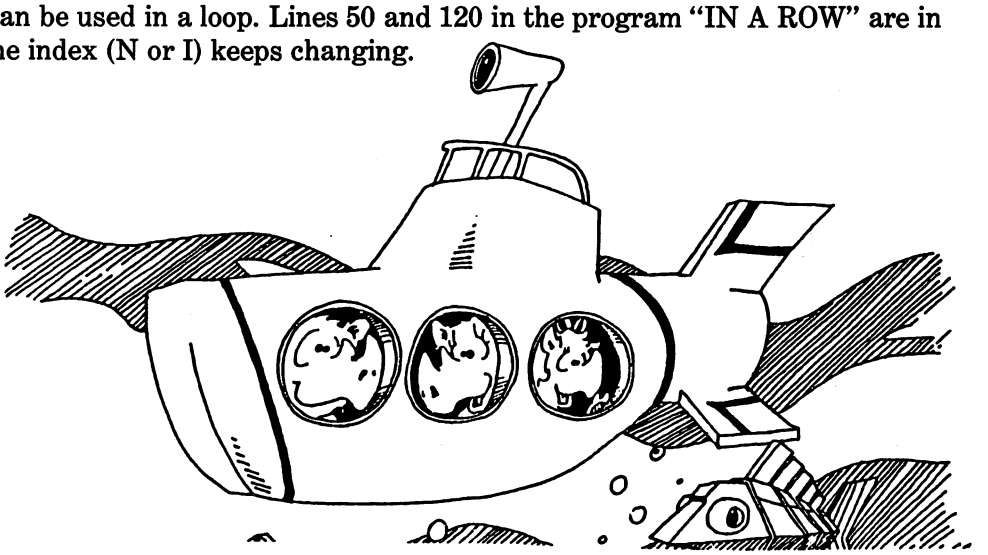
```
10 REM ++ IN A ROW ++
30 DIM A$(4)
35 PRINT"clr dn dn ENTER A WORD dn"
40 FOR N=0 TO 4
45 IF N>0 THEN PRINT"ANOTHER dn"
50 INPUT A$(N)
55 PRINT
60 NEXT N
100 PRINT"IN A ROW dn dn"
110 FOR I=0 TO 4
120 PRINT A$(I);TAB(10);I
130 NEXT I
```

SAVE and run.

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

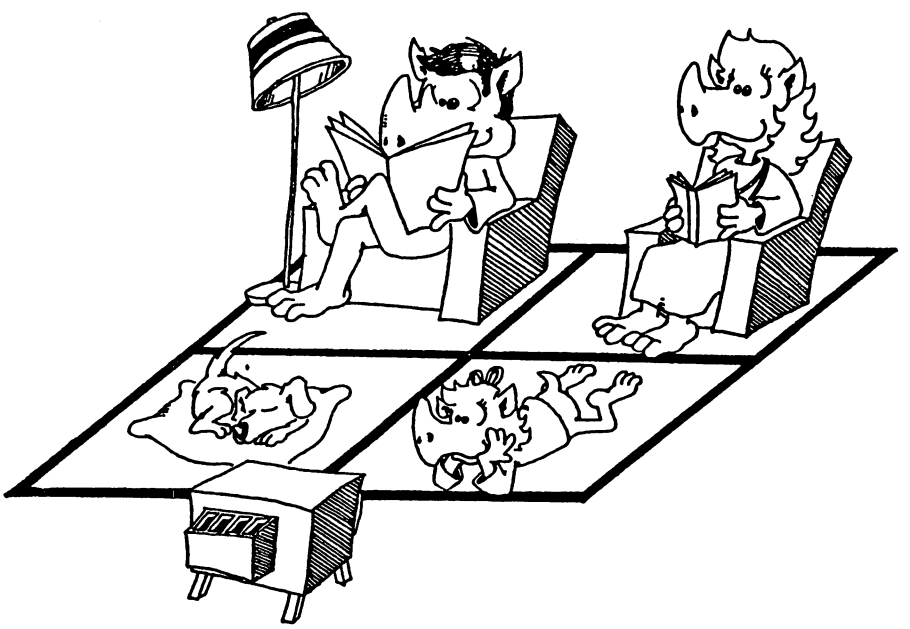
Enter
```
10 REM PHONE LIST
30 DIM NAME$(20), NUMBER$(20)
35 I=0
40 PRINT "clr dn ENTER DATA dn"
50 INPUT " NAME ";N$
55 IF N$="END" THEN END
56 NA$(I)=N$
60 INPUT " NUMBER ";NU$(I)
65 PRINT
70 I=I+1:GOTO 50
```

Run. Press STOP key to stop program. Save to tape.

ONE DIMENSION, TWO DIMENSION, ...

The arrays which have one index are called one dimensional arrays.

But arrays can have two or more indices. Two dimensional arrays have their "family members" put into a rectangle like the days in a month on a calendar.



EIGHT QUEENS

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

```
1 REM *** EIGHT QUEENS ***
2 GOTO 1000
100 REM MAIN LOOP
115 M=R(I)
120 M=M+1:IF M=9 THEN GOSUB 600:GOTO 115
130 IF B(I,M)=0 THEN 700
140 GOTO 120
200 REM
201 REM UPDATE ATTACKED SQUARES
202 REM
210 FOR L=1 TO 8
215 B(I,L)=B(I,L)+D
220 B(L,J)=B(L,J)+D
225 NEXT L
500 REM
501 REM DIAGONAL
502 REM
510 FOR K=1 TO 8
515 X=I+K
520 IF X>8 THEN 530
522 Y=J+K:IF Y>8 THEN 525
523 B(X,Y)=B(X,Y)+D
525 Y=J-K:IF Y<1 THEN 530
526 B(X,Y)=B(X,Y)+D
530 X=I-K
535 IF X<1 THEN 590
540 Y=J+K:IF Y>8 THEN 550
545 B(X,Y)=B(X,Y)+D
550 Y=J-K:IF Y<1 THEN 590
555 B(X,Y)=B(X,Y)+D
590 NEXT K
595 B(I,J)=Q
599 RETURN
600 REM
601 REM GO BACK
602 REM
610 R(I)=0
612 I=QN
615 IF I=0 THEN END
620 J=R(I)
630 D=-1:Q=0:GOSUB 200
640 QN=QN-1
690 REM GOSUB 900
699 RETURN
700 REM
701 REM GO AHEAD
```

```
702 REM
710 R(I)=M:J=M
715 QN=QN+1
720 D=1:Q=-1
730 GOSUB 200
735 NM=NM+1
740 IF QN=8 THEN GOTO 800
780 I=I+1
785 GET KB$
786 IF KB$="sp" THEN GOSUB 900
799 GOTO 115
800 REM
801 REM SOLUTION
802 REM
810 NA=NA+1
814 NS=NS+1
815 GOSUB 900
860 GOSUB 612
899 GOTO 115
900 REM
901 REM DISPLAY
902 REM
905 PRINT "clr"
910 FOR X=1 TO 8
915 FOR Y=1 TO 8
919 PRINT LEFT$(D$,5+2*X);
920 PRINT TAB(4=3*Y);
925 BB=B(X,Y)
930 IF BB=-1 THEN PRINT " Q ";
940 IF BB>-1 THEN PRINT " .";
950 IF BB<-1 THEN PRINT X,Y;:END
960 NEXT Y:NEXT X
980 PRINT "hm dn SOLUTIONS";NS,"MOVES";NM
999 RETURN
1000 REM
1001 REM INITIALIZE
1002 REM
1010 DIM R(8),B(8,8)
1020 I=1:QN=0:NS=0
1030 D$="hm dn dn ... dn"   (25 of them)
1040 PRINT "clr dn dn dn PRESS THE SPACE BAR TO SEE THE BOARD"
1050 FOR T=1 TO 2000:NEXT
1999 GOTO 100
```

Assignment 30:

1. Write a program which stores the number of days in each month in an array. Then when you ask the user to enter a number <1 to 12>, it prints out the number of days in that month.

2. Write a program so that the computer plays the card game "WAR" against the user. Have an array hold the 52 cards in the deck. Deal them at random into two arrays, one for each player. In each turn of play, each player plays the cards from her deck in order. If the cards played by user and computer don't match, they both are put in the "booty" pile. If they do match, there will be a BATTLE. In the battle, each player plays her next card. The high card wins the whole booty pile.

3. We wrote "Eight Queens" for a standard 8x8 chess board. Change the "Eight Queens" so the user can choose any size board.

4. Change "Eight Queens" into super queens. Each moves like a Queen and like a Knight. Are there any solutions?

INSTRUCTOR NOTES 31 SPRITES

The VIC chip in the Commodore 64 controls the video displays. In particular, it controls eight sprites, powerful bit-mapped graphics objects (24 bits wide and 21 bits high).

This lesson gives the student a program in which the sprite she draws on the screen is POKEd into memory without the drudgery of determining bit patterns and changing them to decimal numbers.

The sprite can be colored with one POKE and moved to another spot on the screen with one to three more POKEs. Each sprite is told which picture to display by a "pointer" updated with a single POKE. Many pictures can be stored ahead of time and rapidly displayed one after another by the same sprite. Conversely, several sprites can display the same picture in different colors and different spots on the screen at the same time.

Those portions of a sprite in which the bit is "0" are transparent, showing whatever objects (other sprites or screen graphics) are behind them. The frontmost sprite of an overlapping pair is the one with the lowest sprite number. For each sprite, the programmer can choose whether it passes in front of or behind objects in the static screen display.

Each sprite can be expanded to double size in the horizontal or vertical (or both) dimensions. Multicolored sprites can be made (three colors plus background) but this is not explained in this book.

There are two registers for recording collisions involving sprites. One records if a sprite has collided with any background drawing on the screen. The other records which sprites have collided with other sprites. These registers keep a record of the collision, even if the sprites have moved off and are no longer overlapping, until the register is read with a PEEK. Then the register is cleared to zero, "no collisions."

QUESTIONS:

1. How do you tell sprite 0 which picture to show?

2. How do you tell sprite 0 to move to the center of the screen?

3. How do you color sprite 1 red?

4. How do you tell sprite 2 to grow wider?

5. What does the "sprite hits sprite?" register contain if sprite 3 and sprite 5 are overlapping?

LESSON 31 SPRITES FOR ACTION GRAPHICS

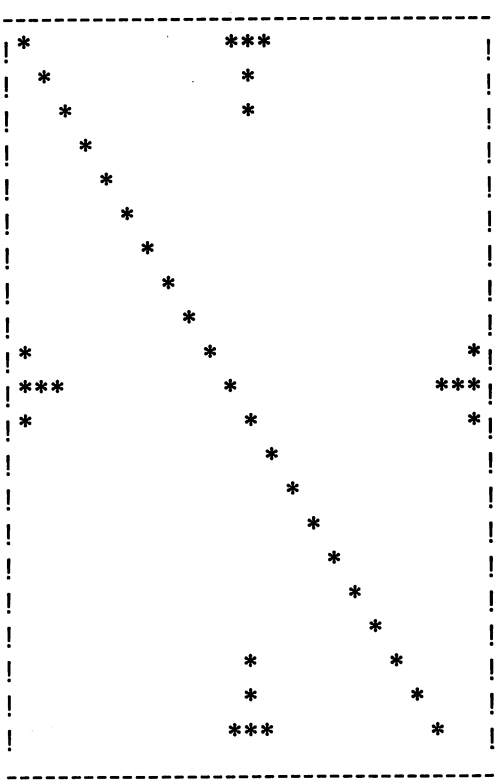
The VIC chip in the computer controls eight sprites.

A sprite is a little picture which moves on the screen. You draw the picture and put it into memory beforehand. Sprites let you put powerful moving graphics into your programs.

SPRITE ZERO, OBEY OUR WISHES!

```
10 REM ----- SPRITE -----
5 PRINT "clr blk"          :REM     BLACK LETTERS
16 POKE 53281,1            :REM     WHITE SCREEN
20 REM ---------------     SET UP VIC
22 V=53248                 :REM     VIC BASE ADDRESS
23 REM ---------------     PICTURE
24 B=200                   :REM     BLOCK NUMBER
26 POKE 2040,B             :REM     PUT IN BLOCK 200
28 GOSUB 200               :REM     STORE PICTURE
30 REM ---------------     ENABLE SPRITE 0
32 POKE V+21,1             :REM     TURN ON SPRITE 0
34 POKE V+39,0             :REM     BLACK COLOR
40 REM ---------------     USE TO SPRITE
42 FOR P=1 TO 200
44 POKE V+0,P              :REM     X POSITION
45 POKE V+1,P              :REM     Y POSITION
49 NEXT P
50 FOR I=7 TO 0 STEP -1
55 POKE V+39,I             :REM     ALL COLORS
60 FOR T=1 TO 1000:NEXT T
65 NEXT I
70 POKE V+29,1             :REM     FAT SPRITE
71 GOSUB 100
72 POKE V+23,1             :REM     FAT AND TALL
73 GOSUB 100
74 POKE V+29,0             :REM     TALL ONLY
75 GOSUB 100
76 POKE V+23,0             :REM     SMALL AGAIN
77 GOSUB 100
90 POKE V+21,0             :REM     TURN SPRITE OFF
96 LIST 300
100 FOR T=1 TO 1000:NEXT T:RETURN
200 REM --- STORE SPRITE --
201 L=0:PRINT " PLEASE WAIT"
202 FOR I=0 TO 20:READ R$
```

```
206 FOR J=0 TO 2:S$=MID$(R$,J*8+2,8)
210 N=0:P=128
215 FOR K=1 TO 8:D$=MID$(S$,K,1)
220 IF D$="*" THEN N=N+P
225 P=P/2:NEXT K
230 POKE B*64+L,N:L=L+1
280 NEXT J,I:PRINT "clr"
299 RETURN
300 REM
301 DATA
302 DATA
303 DATA
304 DATA
305 DATA
306 DATA
307 DATA
308 DATA
309 DATA
310 DATA
311 DATA
312 DATA
313 DATA
314 DATA
315 DATA
316 DATA
317 DATA
318 DATA
319 DATA
320 DATA
321 DATA
322 REM
```



SAVE before running!

HOW TO USE THE SPRITE PROGRAM

Draw your own picture in the DATA statements and enter the DATA lines into the program, like this:

List the program and see the DATA statements on the screen.

Move the cursor into the rectangle drawn in the DATA statements. Erase the stars which are already there. Draw whatever picture you like (a smiley face?) using the "*" key.

Then use the CRSR keys to move the cursor to line 301 at the top of the drawing. Then press the RETURN key 21 times to enter all the new DATA statements into the program.

Then run the program to see your sprite move.

HOW IT WORKS

The VIC chip has 47 memory boxes which tell the sprites what to do. Besides that, each sprite needs a picture stored in a block of 64 boxes in memory, and a "sprite pointer" box to tell which 64 boxes to use.

Line 22 The addresses of VIC's boxes start at 53248.

Line 26 The box at 2040 is the "sprite pointer" for sprite 0. It tells in which block of 64 boxes sprite 0 will find its picture. Block 200 has boxes 12800 to 12863. (12800 = 64*200).

CAREFUL! Use only blocks 200 to 255!

Line 32 Box V+21 is the sprite on-off box. Put number 0 into the box to turn off all the sprites. The number 1 turns on sprite 0.

Line 34 Box V+39 holds a color number (0 to 15) for sprite 0.

Line 44 Box V holds the X position of sprite 0. The number is between 0 and 255. Numbers near 0 put the sprite off the screen at the left.

Line 45 Box V+1 holds the Y position of sprite 0. Numbers near 0 put it off the screen at the top. Numbers near 255 put it off the screen at the bottom.

Line 200 The subroutine reads each DATA line, breaks it up into three "bytes" of eight characters each, and then looks to see which characters are stars. The stars tell which bits are "ones" in binary notation. Study pages 72 and 73 in the Commodore 64 User's Guide to learn more about making binary numbers into decimal numbers for sprites.

TWO STEPS TO SPRITES

You use sprites in two steps: First you draw and store a picture. Then you make VIC color it, turn it on and move it around.

Load the SPRITE program, then change it.

Remove all lines 20 to 96 and line 201. Add these lines:

```
10 REM ----- MAKE PICTURES ----
30 INPUT " BLOCK NUMBER";B
35 IF B<200 THEN 30
40 GOSUB 200
96 LIST 300
240 PRINT N,
280 NEXT J:PRINT:NEXT I
292 INPUT " NEXT PICTURE ";A$
```

How to use the program:

LIST it to see the box made of DATA statements.

Draw your new picture.

Then run the program and enter a block number from 200 to 255. The picture is stored in the block.

Also, the numbers draw the picture are shown on the screen so you can copy them down if you want. You can put these numbers into DATA statements if you want to write a program which uses that picture.

Finally, the DATA statements are LISTed again so you can draw a new picture to store in another block.

You can store many pictures for one sprite to use, and change from one picture to another by POKEing a new block number into the sprite pointer box. (Remember, for sprite 0 the pointer is in box 2040.) Change the pictures very fast and you can make a movie!

Assignment 31A:

1. Use MAKE PICTURES to store a smiley face in block 200. Then draw a frowning face and store it in block 201. (The smile in 200 is still there.)

   Then write a program to make first the smile, then the frown, show in the middle of the screen. (Hint: Use sprite 0 and change its pointer B in a loop.)

2. Now that you know how, store several faces and write a program which changes the face slowly from a big smile, to a small smile, then to a puzzled look, then to a frown, then to an angry face.

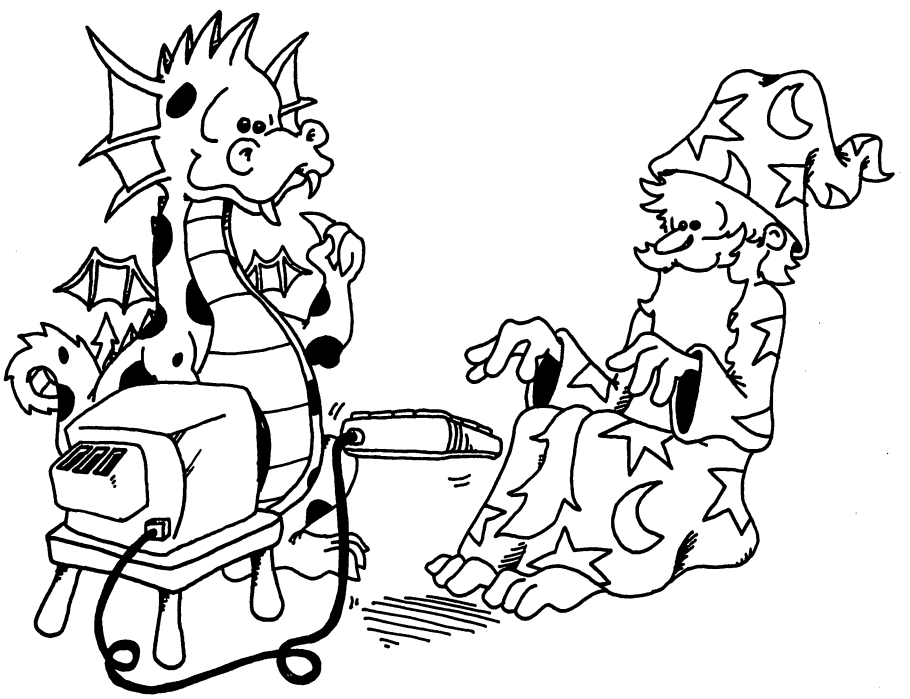
MANY SPRITES AT ONCE

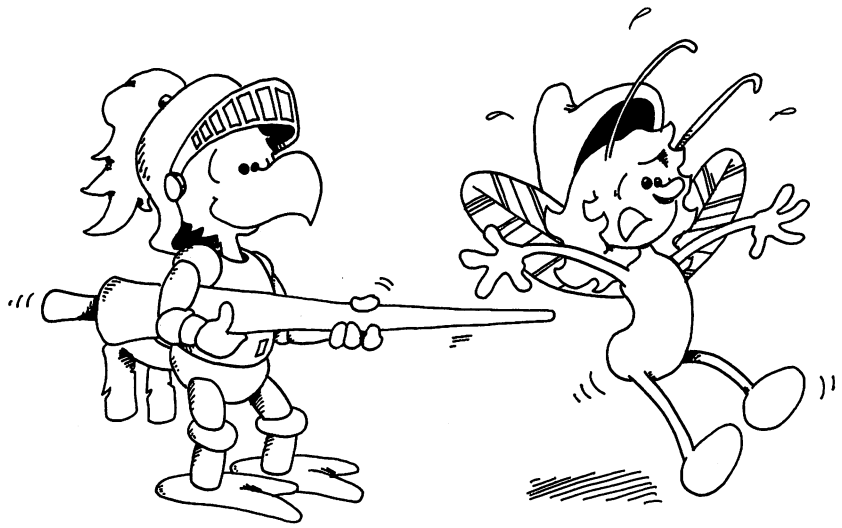
Here are the box addresses for all eight sprites.

Sprite	pointer	X	Y	color	bit value	
		V+	V+	V+		(V=53248)
0	2040	0	1	39	1	
1	2041	2	3.	40	2	
2	2042	4	5	41	4	
3	2043	6	7	42	8	
4	2044	8	9	43	16	
5	2045	10	11	44	32	
6	2046	12	13	45	64	
7	2047	14	15	46	128	

You already know how to use the pointer, X, Y, and color boxes for the 0 sprite. They are used in the same way for the other sprites.

The bit value for each sprite is a number used in the registers listed below.





Registers:	X extension?	V + 16
	sprites on?	V + 21
	tall?	V + 23
	wide?	V + 29
	behind background?	V + 27
	sprite hit sprite?	V + 30
	sprite hit background?	V + 31

Each register is a memory box in the VIC chip which holds the answers to eight yes-no questions, one answer for each sprite.

The bit values of each sprite are used to answer the questions. Example:

	33 POKE V + 21, 1+4+128	turns on sprites 0, 2, and 7
Then:	56 POKE V + 21, 1 + 128	turns off sprite 2 (because its bit value, 4, isn't there).

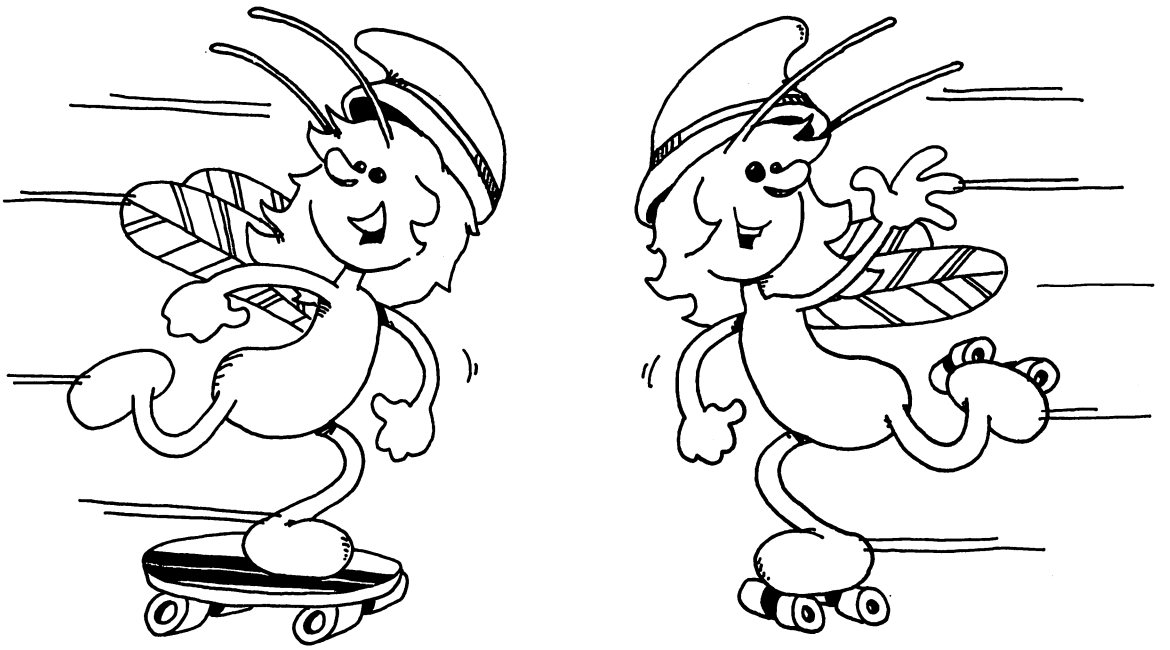
RULES: To answer no for all sprites, POKE 0.

To answer yes for one sprite, POKE its bit value. (This automatically answers no for all the other sprites).

To answer yes for several sprites, POKE the sum of their bit values. For all bit values you leave out (like 2, 8, 16, 32, and 64 in line 33) the answer is no.

The "X extension?" register helps you move a sprite to the right side of the screen. If you answer yes then the number 256 is added to the position you put into the sprite's X position register. Example:

```
35 POKE V + Ø,5Ø :REM     PUT SPRITE Ø AT 5Ø HORIZONATALLY
47 POKE V +16, 1 :REM     MOVE IT TO 256+5Ø=3Ø6
```



WATCH OUT, TWO SPRITES ARE GOING TO BUMP!

When two sprites overlap on the screen, their bit values added together appear in box V+30, the "sprite hit sprite?" register. Example:

```
87 H=PEEK(V+3Ø)
88 PRINT H
```

If "6" is printed, then you know that sprites 1 and 2 have hit each other because their bit values, 2+4, add to 6.

If a sprite hits something you have drawn on the screen, its bit value is added to box V+31, the "sprite hit background?" register.

Assignment 31B:

1. Make two sprites which can move horizontally. Start them at opposite ends of the screen. When they hit, make a "crash" sound.

2. Make a Bumper Car rink. Put a border around the screen. Have two sprites (different colors) for bumper cars. Make eight pictures of cars, going up, down left, right and toward each diagonal. Control the cars with keys for "turn left" and "turn right." When the cars hit, they have to back away.

INSTRUCTOR NOTES 32 USER FRIENDLY PROGRAMS

This lesson shows how to write clear programs which interact with the user in a "friendly" way.

They should also be clearly structured from the programmer's point of view. The "spaghetti" program should be discouraged. A format for writing programs is presented in this lesson. While methods of imposing order on the task are largely a matter of taste, the methods used in this lesson can serve to introduce the ideas.

"User friendly" means that the screen displays are easy to read, keyboard input is "RETURN key free" as much as possible and errors are "trapped." Ask if entries are OK. If not, give an opportunity to fix things.

Instructions and "HELP" should be available. Prompts need to be given. Beginners need complete prompts, but experienced users would rather have curt prompts.

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

Most young students will not progress very far toward fully friendly programming. To be acquainted with the desirability of friendly programming and to use some simple techniques toward accomplishing it are satisfactory achievements.

QUESTIONS:

1. Should your program give instructions whether the user wants them or not?

2. What is a "prompt"? Give two examples.

3. What is "scrolling"? How can you write to the screen without scrolling?

4. If you want the user to enter a single letter from the keyboard, what command is best? (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 REM
1001 REM                 *** program name ***
1002 REM
---
---                      REM's that give a description of the
---                      program, variable names, etc.
---
1999 REM
2000 REM STARTING STUFF
---
---                      ask for starting information
---                      set variable values
---                      give instructions
---
2999 GOTO 100
```

Save the outline and use it to start each new program that you write.

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.

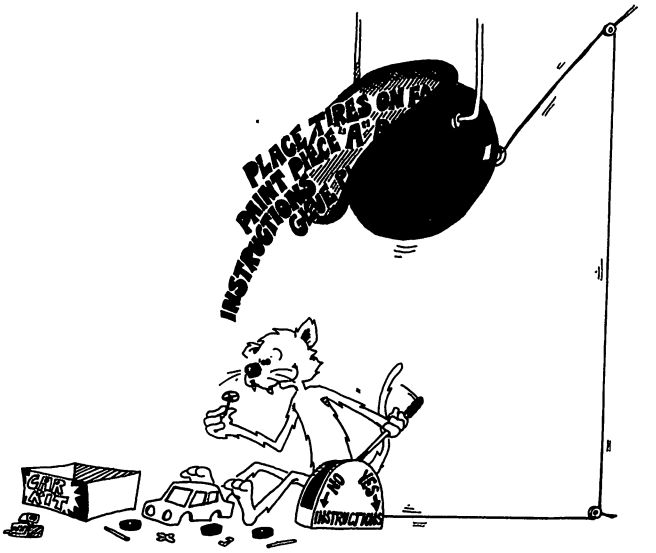
PUT SUBROUTINES IN THREE PLACES

between line 2 and line 99 for subroutines which must run fast after line 2999 for starting stuff subroutines between line 200 and 999 for the rest of the subroutines

INFORMATION PLEASE

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

This lets a beginner see instructions, and lets others say NO.



TIE A STRING AROUND THE USER'S FINGER

Use a prompt to remind users what choices they have.

Example: <Y/N> where the choice is "Y" for "YES" or "N" for "NO"

Beginners need long prompts. Other users like short prompts.

OUCH! MY FINGERS HURT

Use the GET command to enter single letters. This saves having to press RETURN.

```
380 PRINT "DO YOU WANT INSTRUCTIONS? <Y/N> "
382 GET R$:IF R$="" THEN 382
384 IF R$="Y" THEN GOTO 600
```

DON'T GIVE THE USER A HEADACHE

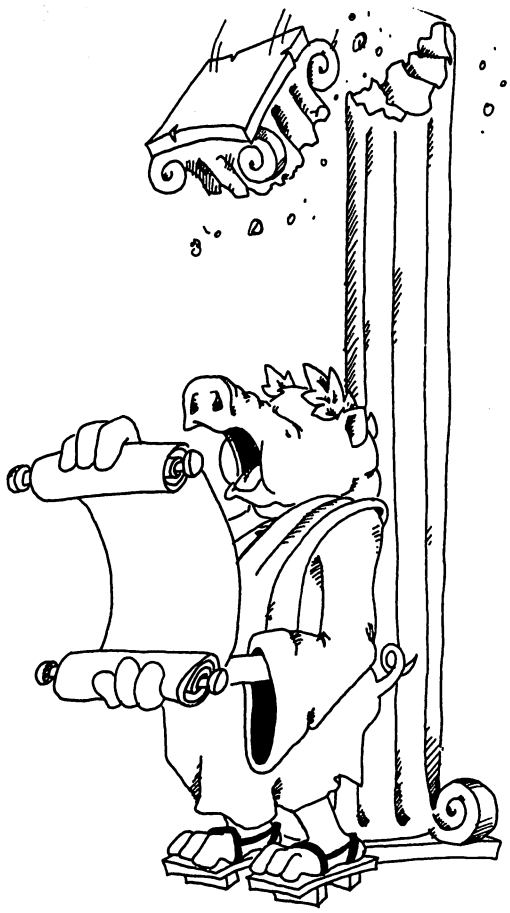
SCROLLING gives headaches!

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 key strings 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.



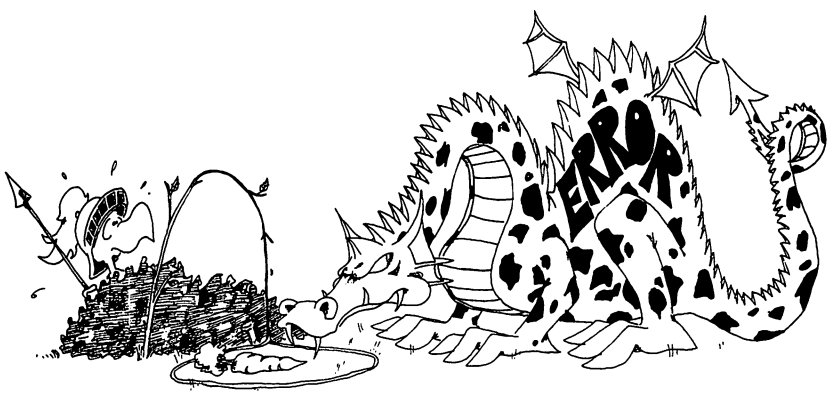
SET TRAPS FOR ERRORS

Example: Add this line to the above lines:

```
386 IF R$<>"N" THEN GOTO 380
```

Line 380 asked for only two choices, Y or N. If the user presses some other key, line 386 sends him back to line 380.

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



Assignment 32:

1. Make a program to write a very large number, 50 digits. Pick the digits at random. Put a comma between each set of three digits. Hint: Make the digits one at a time from RND(0) and glue them on the end of the number.

2. Write a secret cipher program. The user chooses a password and it is used to make a cipher alphabet like this:

   If the password is DRAGONETTE
   remove the repeated letters, get DRAGONET
   put it at the front of the alphabet and the rest of
   the letters after it in normal order

   DRAGONETBCFHIJKLMPQSUVWXYZ cipher alphabet

   ABCDEFGHIJKLMNOPQRSTUVWXYZ normal alphabet

The user chooses to code or decode from a menu.

INSTRUCTOR NOTES 33 DEBUGGING, STOP, CONT

The "sigh and moan" technique being a loser, our students need a bag of tricks which help isolate program bugs, and should practice on programs they are writing as they go through this book.

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

As programs grow in complexity, more of the bugs result from unforseen interactions between separated parts of the program. The bag of tricks we offer helps find these also. Delay loops, PRINT commands and STOP statements help the student see how the program is functioning.

Don't overlook those techniques you can use after the program is stopped with a STOP statement or a STOP key press. You can PRINT out any variable values you like so as to see what the program has done. You can also do arithmetic in the PRINT command to check what the program should be doing. You can even change variable values (for example, the value of a loop variable). When you are ready, CONTinue the program run.

QUESTIONS:

1. How can you make the computer print

   BREAK IN LINE 55

   by adding a line in the program?

2. How are the STOP and the END commands different?

3. How are the STOP command and the STOP key different?

4. What does the CONT command do?

5. Why would you put STOP commands into 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(8)*200)
30 FOR I=0 TO 200
40 IF I=R THEN STOP
50 NEXT I
```

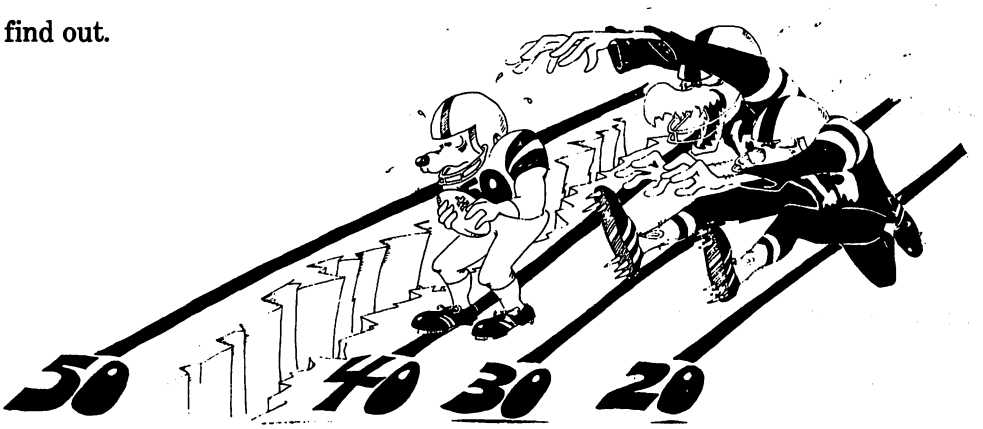
The program will stop, and the computer will peep and print a message:

```
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 prints the number of the line that the STOP is in.

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

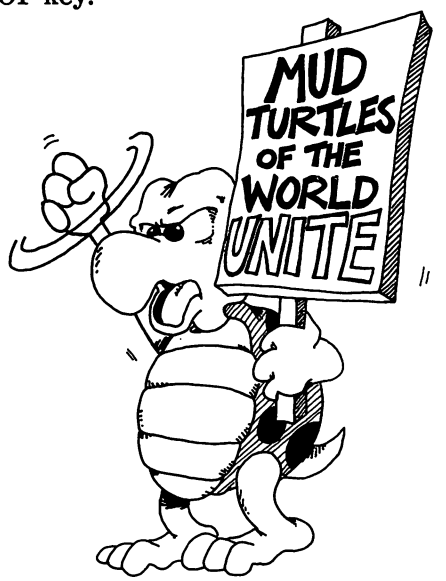
STOP is used for debugging your program.

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 dn"
16 GOSUB 90
20 PRINT "MUD dn"
21 GOSUB 90
22 PRINT "    TURTLES dn"
23 GOSUB 90
24 PRINT "       OF dn"
25 GOSUB 90
26 PRINT "          THE dn"
27 GOSUB 90
28 PRINT "             WORLD dn"
29 GOSUB 90
30 PRINT "                UNITE! dn"
31 GOSUB 90
40 GOTO 10
90 FOR T=1 TO 1000:NEXT T:RETURN
```



Pressing the STOP key stops the program at whatever spot it is. It prints:

BREAK IN LINE XX and enters the edit mode

(XX is the line number where it stops.)

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

The above program usually stops in line 90. Why? Try to make the above program stop in some other line.

WHAT DO YOU DO AFTER YOU STOP?

You put STOP into 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 spot where the trouble is.)

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

You are in the Edit mode. You can:

List parts of the program and study them.

Use the PRINT command to look at variables. Do they have the values you expected?

Do little calculations on the computer in the "calculator mode" (another name for the Edit mode) to check what the computer is doing.

Use the LET command to change the values of variables.

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

STARTING THE PROGRAM AGAIN

There are four ways to start a program. 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

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

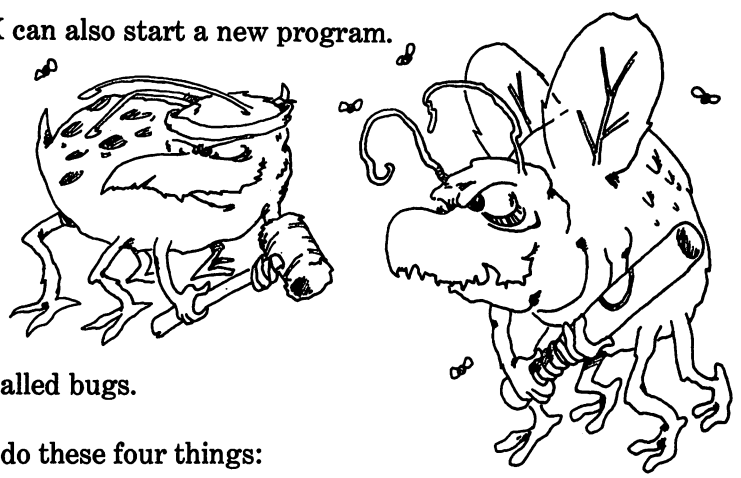
RUN RUN XX

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 restart only 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 in some PRINT lines which will tell what is happening.

3. Or you can put STOP commands into the program.

4. If the program runs so fast that you can't tell what is happening, put in some delay loops to slow it down.

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

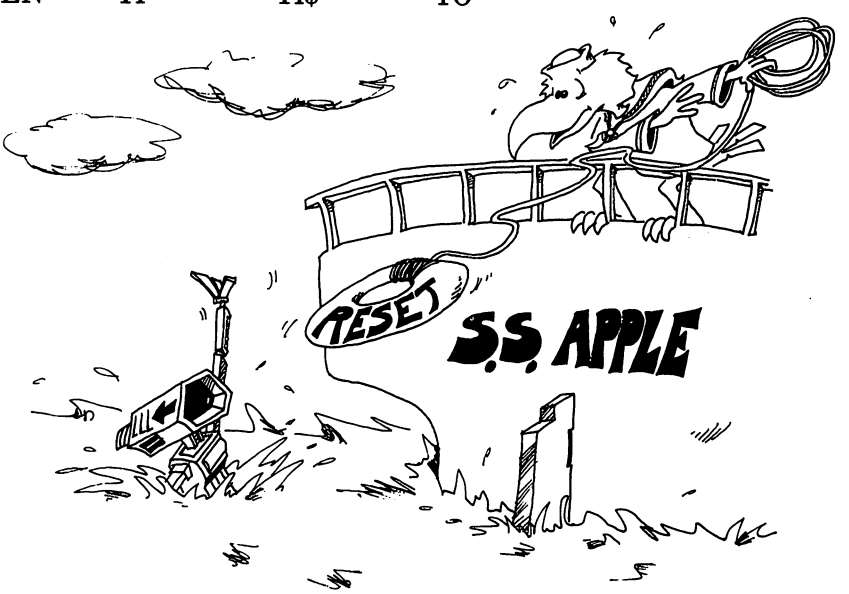
Assignment 33:

1. Go back to the SNAKE program and fix up some of the bugs. For example, the program "crashes" when the snake hits a wall. Add "food" for the snake. Add score keeping. Let the game end if the snake touches a wall.

2. Write a "shape shifter" program. A shape shifter is a demon which starts in one shape and shifts around its parts to make other shapes.

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#	INT		
LEFT$	LEN	LET	LIST	LOAD	LOG
MID$					
NEW	NEXT	NOT			
ON	OR	OPEN	PEEK	POKE	POS
	PRINT	PRINT#			
READ	REM	RESTORE	RETURN	RIGHT$	RND
	RUN				
SAVE	SGN	SIN	SPC(	SQR	STEP
	STOP	STR$	SYS		
TAB(	TAN	THEN	TI	TI$	TO
USR					
VAL	VERIFY				
WAIT					



GLOSSARY

argument The variable, number or string which appears in the parentheses of a function. Like:

INT(N)	has	N	as an argument
LEN(W$)	has	W$	as an argument

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

A(Ø)	is the first member of the array A
B$(7)	is the eighth member of the array B$
CD(3,M+1)	is a member of the array CD

arrow keys
There are two CRSR keys on the C-64. Each has two arrows. There are two other keys which have arrows and print them as characters.

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

assertion
The name of a phrase which can be TRUE or FALSE. The "phrase A" in an IF statement is an assertion. An assertion has a numeric value of 0 or -1. See expression, TRUE, FALSE, logic, IF, phrase A. Example:

the assertion	"A"<>"B"	is	TRUE
the assertion	3 = 4	is	FALSE

background
The part of the screen which is blank, not having characters on it.

base address
The SID chip has registers (memory boxes) from 54272 to 54301. So its base address is 54272, and one adds on numbers from 0 to 29 to get the addresses (box labels) of the various registers. Likewise, the VIC chip has a base address 53248.

BASIC
Beginners's All-purpose Symbolic Instruction Code. A computer language originated by John Kemeny and Thomas Kurtz at Dartmouth College in the early '60s.

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 which is a space.

boot
Means to start up the computer from scratch. An easy thing to do with modern computers which have start up programs stored permanently in ROM memory. It was an involved procedure in the early days. Now it usually means to read in the disk operating system programs (DOS) from a disk.

branch
A point in a program where there is a choice of which statement to execute next. An IF statement is a branch. So is an ON...GOTO statement. A branch is not the same as a jump where there is no choice. See jump.

buffer
A storage area in memory for temporary storage of information being input or output from the computer.

call
Using a GOSUB calls the subroutine. Putting a function into a statement calls the function. Call means the computer does the commands in the subroutine or does the calculation for the function, then returns to the calling spot.

carriage return
On a typewriter, you push the lever which 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 you see as pictures on some keys.

clear
Means erase. Used in "clear the screen" and "clear memory."

column
Things arranged vertically. See row.

command
In BASIC a command makes the computer do some action, such as erase the screen and memory by the NEW command. See statement, expression. Some commands need expressions to be complete. Example:

```
SAVE "doggie"
```

concatenation

Means sticking two strings together.

constant

A number or string which 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 which shows where the next character on the screen or in a storage buffer will be placed. Cursor means "runner." The cursor runs along the screen as you type. There are two kinds of cursors in the Commodore 64 computer:

INPUT cursor	a flashing square on the screen
PRINT cursor	invisible, "shows"

data

BASIC has two kinds of data: numeric and string. Logic data (TRUE, FALSE) is a type of numeric data.

debug

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

delay loop

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

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

edit

There are two kinds: editing a line and editing a program. In either kind, you are retyping parts of it to correct it.

Edit mode

When "READY." and the flashing cursor show, you are in the edit mode. The computer awaits commands or the entering of program lines.

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 which checks for mistakes in information which the user has entered, or checks to see if computed results are within reasonable bounds.

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

expression
A portion of a statement which has a single value, either a number or a string. See value. Examples:

```
7*X+1
"DOPE "<> N$
A$ + "HAT"
```

FALSE
The number 0. See logic, TRUE, assertion.

fork in the road
Describes a branch point in the program. See branch.

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

```
ASC, CHR$, INT, LEN, RND, LEFT$, MID$,
RIGHT$, STR$, VAL, TAB, PEEK
```

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

graphics
Means picture drawing.

index
An array name is followed by one or more numbers or numerical variables in parentheses. Each number is an index. Another word for index is subscript.

```
Q(7,J)+ 7 and J are indices
```

integers
The whole numbers, positive, negative and zero.

I/O
Input/Output. Input from keyboard, tape recorder, etc. Output to screen, printer, tape recorder, etc.

joystick
A device used in games. It is like the control stick used in early airplanes. It can detect eight different directions, as well as centered.

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

line
Lines start with a number followed by a statement which may contain expressions, etc.

Parts of a line:

```
16 IF 7<=INT(Z) THEN PRINT LEN(Q$+"R")+2;"RAT":GOTO 40
```

16	line number
IF 7<=INT(Z) THEN PRINT..."RAT"	statement
GOTO 40	statement
7<=INT(Z)	an assertion
7<=INT(Z)	an expression
LEN (Q$"R")+2;"RAT"	an expression
Q$;"R"	an expression
INT(Z)	a function
LEN(Q$)	a function
Z	argument
Q$+"R"	argument
7, "R", 2, "RAT"	constants
<= , +	operations

IF, INT, THEN, PRINT, LEN, GOTO reserved words

line buffer
The storage space which receives the characters you type in. See buffer.

line edit
Retyping parts of a line to correct it.

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

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

listing
A list of all the lines in a program.

load
To transfer the information in a file on tape or disk to the memory of the computer by using the LOAD command.

logic
The part of a program which compares numbers or strings. The relations =, <>, <, >, <=, and >= used. See assertion, phrase A, AND, OR, NOT.

loop
A part of the program which is done over and over again. There are two kinds of loops: FOR...NEXT loops, "home made" loops which use IF... commands with a loop variable and GOTO commands.

loop variable
Is the number which changes as the loop is repeated. For example:

```
40 FOR I= TO 5
50 NEXT I              I is the loop variable
```

memory
The part of the computer where information is stored. Memory is made of semiconductor chips, but we think of it as boxes with labels on the front and information inside.

menu
A list of choices shown on the screen. Each choice has a letter or number beside it. The program user presses a key to pick which choice is wanted.

message
A statement which tells what is expected in an INPUT statement. Example:

```
61 INPUT "AGE";A
```

monitor
Has two meanings. We use it to mean a box with a TV type screen which is connected to the computer. It displays text and graphics but cannot receive television programs. In machine language programming, a monitor is a control program.

nesting
When one thing is inside of another. In a program we nest loops. Inside a statement, we can nest expressions or functions.

```
L=INT(LEN(P$)+3)      nested functions
X=*(6++7*(8+K))       nested parentheses
```

number
Is one type of information in BASIC. The other is string. The numbers are generally decimal numbers. See integer, strings.

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

phrase A
Is a phrase in this book which stands for an assertion in an IF statement. See assertion. Example:

```
IF A>4 THEN 500      A>4 is "phrase A"
```

pitch
The numbers poked into the SID chip which tells the musical pitch of the sound. Pitch is the same as "note" and can be high or low.

pixel
PIcture Element. The smallest dot which is placed on the screen in a graphics mode.

pointer
A number in memory which tells where in the lists of DATA you are at the present moment. Also, number in a pointer box that tells a sprite which block contains its picture.

program
The usual program is a list of numbered lines containing statements. The computer executes the statements (commands) in order when the RUN command is entered. The program is stored in a special part of memory, and only one program can be stored at a time.

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

pseudo-random
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.

random
Numbers which cannot be predicted, like the numbers which show after the roll of dice, or the number of heads you get in tossing a coin 10 times.

register
A memory box contained in the VIC or the SID chip. The number you put in the box controls some action of the chip.

remark
A comment you make in the program by putting it into a REM statement. Example:

```
REM the graphics setup subroutine
```

reserved words
A list of words and abbreviations that BASIC recognizes as commands, statements, or functions. The reserved words cannot be used as variable names.

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

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

row
Things arranged horizontally (across).

save
To put the program which is in the computer's memory onto tape.

screen
The TV screen or a similar one in a monitor which is hooked up to the computer. See monitor.

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

simple variable
A variable which is not an array variable.

stack
Is a data type used in machine language programming. The data are arranged in a column and the last one put on is the first one taken off.

starting stuff
Is the name given in this book to initialization material in a program. It includes REM's for describing the program, input of initial values of variables, set up of array dimensions, drawing screen graphics, and any other things that need be done just once at the beginning of a program run.

statement
The smallest complete section of a program. It starts with a command. The command may have expressions in it.

store
To put information into memory or to save it onto tape.

string
A type of data in BASIC. It consists of a row of characters. See number.

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

subscript
A number in the parentheses of an array. It tells which member of the array is being used. See index.

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

timing loop
A loop which does nothing except use up a certain amount of time. See delay loop.

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

TRUE
Has the value -1. See logic, FALSE, assertion.

truncate
To cut off the decimal part of a number, leaving an integer. The INT( ) function does this for positive numbers.

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

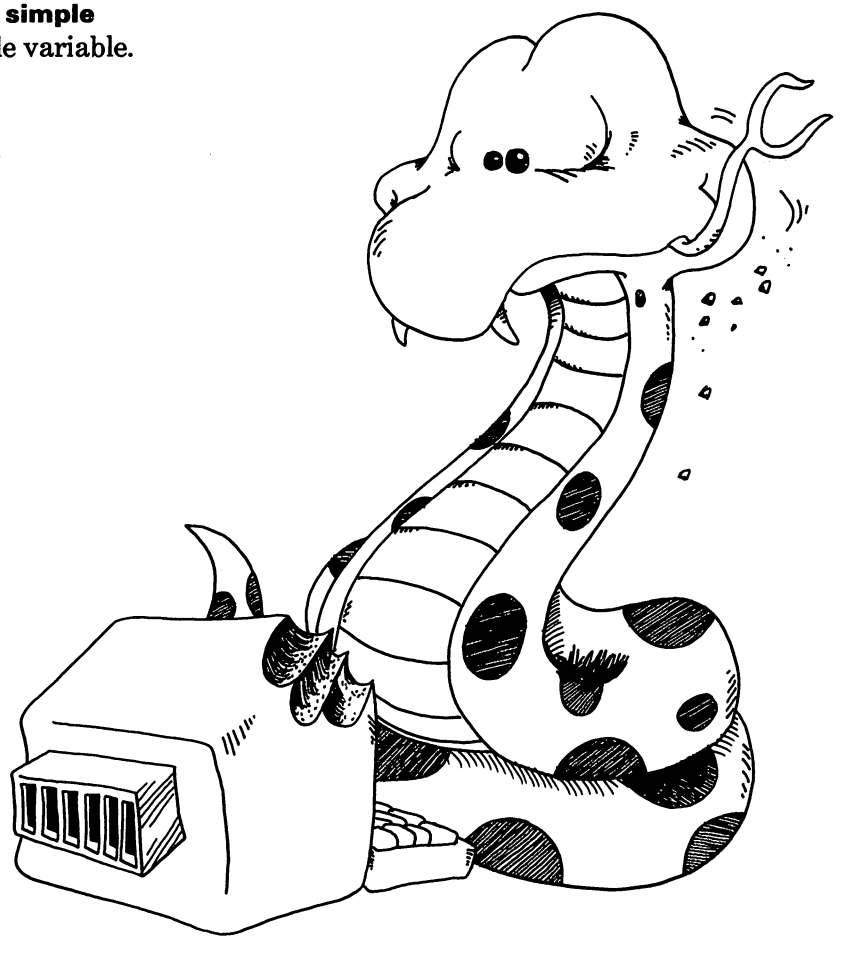
value
The value of a variable is the number or string stored in the memory box belonging to the variable. See variable.

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

variable name
A variable is either a string variable or a numeric variable. The name tells which. String variables have names ending in a "$" sign. Numeric variables do not. The variable name can have any number of characters but only the first are kept by the computer when making its variable table. The first character must be a letter, the rest may be letters or a numbers.

variable, array
See array.

variable, simple
See simple variable.



ERROR MESSAGES

BAD DATA

You read a tape file which was suposed to have numeric 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.) which 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 which needs to be split into parts.

ILLEGAL DIRECT

You used DATA, INPUT, GET or DEF FN in the Edit mode.

ILLEGAL QUANTITY

You made one of these errors:

You used a negative number as an array subscript, like

```
LET A(-1)=34
```

You used a function with the wrong kind of argument. Like:

wrong: L=VAL(R)	correct: L=VAL(R$)
wrong: TAB(-3)	correct: TAB(3)
wrong: SPEED 400	correct: SPEED 255

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 which 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 did 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 # into 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 which is not in your program.

VERIFY

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

ANSWERS TO ASSIGNMENTS

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

1 REM A2-2
10 REM *** COLORED NAMES ***
20 POKE 53281,0
30 PRINT "{wht} M {red} I {cyn} N {pur} D {grn} A"
35 PRINT "{yel} ANNE"
40 PRINT "{rvs}{red} CARLSON {wht}"

1 REM A3-5
10 REM *** BIRDS ***
15 POKE 53281,0
20 PRINT "{clr}"
22 PRINT
24 PRINT
30 PRINT "{cyn} UQI"
32 PRINT
34 PRINT
40 PRINT "{yel}          JQK"
42 PRINT
44 PRINT
46 PRINT
50 PRINT "{grn} CWC"
90 PRINT "{yel}"
```

```
1 REM A4-3
10 REM *** SMILE ***
13 POKE 53281,7
15 PRINT "{clr}"
20 PRINT
22 PRINT
24 PRINT
26 PRINT "{red}"
30 PRINT "    00  00      "
32 PRINT "    00  00      "
34 PRINT
35 PRINT
36 PRINT
37 PRINT
38 PRINT
40 PRINT
42 PRINT " *             * "
44 PRINT "  *           *  "
46 PRINT "   *         *   "
48 PRINT "     ****       "
```

```
1 REM A5-1
10 REM *** BY DAN CLARK, AGE 10 ***
15 PRINT "{clr}"
20 PRINT "NAME A MUSICAL GROUP"
30 INPUT A$
40 PRINT
50 PRINT "NAME ONE OF THEIR SONGS"
60 INPUT B$
70 PRINT
80 PRINT A$;" PLAYS ";B$
```

```
1 REM A5-2
10 REM *** 3 PRINTS ***
15 PRINT "{clr}"
20 PRINT "GIVE ME THE NAME OF A MUSICAL GROUP"
22 INPUT G$
25 PRINT
30 PRINT "WHAT IS ONE OF THEIR TUNES"
35 INPUT T$
40 PRINT "{dn}"
50 PRINT G$;
52 PRINT " PLAYS ";
54 PRINT T$
```

```
1 REM A5-3
10 REM *** THREE LITTLE WORDS ***
20 POKE 53281,0
21 PRINT "{wht}"
30 PRINT "GIVE ME A WORD"
31 INPUT W$
40 PRINT "GIVE ME ANOTHER"
41 INPUT X$
50 PRINT "ONE MORE"
51 INPUT Y$
60 PRINT "{clr}"
62 PRINT "{dn}{dn}{dn}"
65 PRINT "YOU MEAN"
66 PRINT "{dn}{dn}{dn}"
70 PRINT W$,X$,Y$

1 REM A6-1
10 REM *** SILLY GOOSE ***
15 PRINT "{clr}"
17 PRINT "{wht}"
20 PRINT
22 PRINT
24 PRINT
30 PRINT "HELLO,"
31 PRINT
32 PRINT "WHAT IS YOUR NAME?"
33 PRINT
35 INPUT N$
37 PRINT "{clr}{pur}"
38 PRINT
40 PRINT "WELL"
41 PRINT
42 PRINT N$
43 PRINT
50 PRINT "IT IS SILLY TO TALK"
51 PRINT
52 PRINT "TO A COMPUTER"
55 PRINT "{wht}"
```

```
1 REM A6-2
10 REM *** FAVORITE ***
15 PRINT "{clr}{red}"
20 PRINT "{dn}{dn} WHAT IS YOUR FAVORITE COLOR?"
25 PRINT
26 INPUT C$
30 PRINT "{dn}{blu} I PUT THAT IN BOX C$."
35 PRINT "{dn}{grn} NOW YOUR FAVORITE ANIMAL? {dn}"
40 INPUT C$
45 PRINT "{dn}{yel} I PUT THAT IN BOX C$ TOO"
50 PRINT "{dn}{cyn} NOW LET'S SEE WHAT IS IN BOX C$"
55 PRINT "{dn} IT IS:"
56 PRINT
60 PRINT C$
```

```
1 REM A7-2
10 REM *** FEELINGS ***
12 PRINT "{clr}"
20 PRINT
22 PRINT
24 PRINT " HOW IS THE WEATHER?"
26 PRINT
28 INPUT W$
29 PRINT
30 PRINT " AND HOW DO YOU FEEL?"
32 PRINT
34 INPUT F$
36 PRINT
38 PRINT " YOU MEAN:"
40 PRINT
45 LET S$=W$ +$ " AND " +$ F$
50 PRINT " ";S$
```

```
1 REM A8-2
10 REM *** T E E N  T I M E S ***
20 PRINT "      ";
21 PRINT "T  E  E  N   P  O  W  E  R"
22 PRINT
23 PRINT
24 PRINT
25 PRINT
30 GOTO 20
```

```
1 REM A8-4
10 REM *** COLOR FAST FRIENDS ***
15 POKE 53281,0
20 PRINT "{clr}"
22 PRINT
24 PRINT
30 PRINT "{yel} BRIAN"
31 PRINT
32 PRINT
33 PRINT
34 PRINT
35 PRINT
40 PRINT "{grn} STEVE"
41 PRINT
42 PRINT
43 PRINT
44 PRINT
45 PRINT
90 GOTO 30
```

```
1 REM A9A-1
10 REM *** BOYS AND GIRLS ***
15 POKE 53281,0
20 PRINT "{clr}{wht}"
22 PRINT
24 PRINT
30 PRINT " ARE YOU A BOY OR A GIRL?"
31 PRINT
32 INPUT W$
33 PRINT
34 PRINT
40 IF W$="BOY" THEN PRINT "{blu} SNIPS AND SNAILS"
50 IF W$="GIRL" THEN PRINT "{pur} SUGAR AND SPICE"
90 PRINT "{wht}"
```

```
1 REM A9B-2
10 REM *** WHAT COLOR ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
30 PRINT " PLAYER 1 TURN YOUR BACK {dn}"
32 PRINT " PLAYER 2 ENTER A COLOR NAME {dn}"
40 INPUT C$
45 PRINT "{clr}{dn}{dn}{dn}"
50 PRINT " PLAYER 1 TURN AROUND AND GUESS? {dn}"
55 INPUT G$
56 PRINT
60 IF G$=C$ THEN GOTO 80
65 IF G$<>C$ THEN PRINT "{yel} WRONG {wht}{dn}"
69 GOTO 55
80 REM CORRECT ANSWER
82 PRINT "{grn} RIGHT!"
90 PRINT "{wht}"
```

```
1 REM A10-1
10 REM *** BIRTHDAY ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
30 PRINT " HOW OLD ARE YOU? {dn}"
31 INPUT A
32 PRINT
33 PRINT " AND WHAT YEAR IS IT NOW? {dn}"
40 INPUT Y
45 LET B=Y-A
50 PRINT "{dn} HAS YOUR BIRTHDAY COME YET THIS YEAR"
51 PRINT " <Y OR N>"
52 PRINT
55 INPUT A$
57 PRINT
60 IF A$="N" THEN LET B=B-1
70 PRINT "{grn} YOU WERE BORN IN";B
90 PRINT "{wht}"
```

```
1 REM A10-2
10 REM *** MULTIPLICATION ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
30 PRINT " GIVE ME A NUMBER {dn}"
31 INPUT N1
32 PRINT
33 PRINT " GIVE ME ANOTHER {dn}"
40 INPUT N2
41 PRINT
45 LET A=N1*N2
50 PRINT " HERE IS THEIR PRODUCT: {dn}"
55 PRINT " ";A
90 PRINT "{wht}"
95 REM TRY N1=2000000 AND N2=3000000
96 REM THE ANSWER IS 6E+$12 IN SCIENTIFIC
97 REM NOTATION

1 REM A11A-1
10 REM *** NICKNAMES ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
24 PRINT " PLAYER 1: WHAT IS YOUR LAST NAME? {dn}"
25 INPUT LN$
26 PRINT: PRINT
30 PRINT " PLAYER 1: TURN YOUR BACK. {dn}"
31 PRINT " PLAYER 2: WHAT IS HIS NICKNAME? {dn}"
32 INPUT NN$
40 PRINT "{clr}{dn}{dn}{dn}"
41 PRINT " PLAYER 1: TURN AROUND AND HIT RETURN. {dn}"
42 INPUT A$
50 PRINT "{clr}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}"
55 PRINT " AROUND HERE YOU ARE CALLED: {dn}{yel}"
60 PRINT TAB(2);NN$;"{grn}";TAB(14);LN$
70 PRINT "{wht}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}"
80 PRINT "HIT RETURN TO PLAY AGAIN. {dn}"
90 INPUT A$
99 GOTO 20
```

```
1 REM A11A-2
10 REM !"#$% INSULTS %$#"!
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
25 PRINT " HEY YOU! WHAT IS YOUR NAME! {dn}"
30 INPUT N$
35 PRINT "{dn}{dn}{cyn}"
40 PRINT TAB(8);"BAH!";N$;" . . . "
50 PRINT "{dn}{dn}{dn}{yel}"
60 PRINT TAB(8);"YOUR FATHER EATS ONIONS!"
90 PRINT "{dn}{dn}{dn}{dn}{dn}{wht}"

1 REM A11B-1
10 REM *** SLOW POKE ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{{dn}{dn}"
24 PRINT " LIKE . . . {dn}{dn}{dn}"
25 FOR T=1 TO 1000: NEXT T
26 PRINT
30 PRINT " MOLASSAS . . . {dn}{dn}{dn}"
31 FOR T=1 TO 1000: NEXT T
33 PRINT
40 PRINT " IN . . . {dn}{dn}{dn}"
41 FOR T=1 TO 1000: NEXT T
42 PRINT
50 PRINT "JANUARY!"
90 PRINT "{dn}{wht}"
```

```
1 REM A11B-2
10 REM *** CLOCK ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
23 PRINT " PRESENT TIME: HR,MIN,SEC?"
24 PRINT
25 INPUT H,M,S
26 LET F=550
30 FOR I=1 TO F: NEXT I
31 PRINT "{hm}{dn}          "
32 PRINT "{hm}{dn}";H;TAB(3);M;TAB(6);S
33 LET S=S+$1
40 IF S<60 THEN GOTO 26
41 LET M=M+$1
42 LET S=0
45 IF M<60 THEN GOTO 26
50 LET H=H+$1
51 LET M=0
60 IF H>12 THEN LET H=1
90 GOTO 26

1 REM A12B-3
10 REM *** I GOT YOUR NUMBER ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}{dn}"
23 PRINT " GIVE ME A NUMBER 0 TO 10 {dn}"
25 INPUT N
26 PRINT
30 IF N=0 THEN PRINT " I GOT PLENTY OF NOTHING"
31 IF N=1 THEN PRINT " I'M NUMBER ONE!"
32 IF N=2 THEN PRINT " TWO IS COMPANY"
33 REM ETC.
40 IF N=10 THEN PRINT " TEN LITTLE INDIANS"
50 FOR T=1 TO 1000: NEXT T
60 IF N<10 THEN GOTO 20
80 PRINT "{dn}{dn}{dn}{dn}"
90 PRINT " THAT'S ALL, FOLKS"
```

```
1 REM A12B-4
10 REM *** PICK A CARD ***
15 POKE 53281,0: LET G=1
20 PRINT "{clr}{yel}{dn}{dn}{dn}"
22 PRINT " PLAYER2: TURN YOUR BACK {dn}{cyn}"
24 PRINT " PLAYER1: THINK OF A CARD, HIT RETURN {dn}"
26 INPUT A$
30 PRINT "{clr}{dn}{dn}{dn}"
32 PRINT " PLAYER1: TELL ME THE SUIT {dn}{pur}"
34 PRINT " HEARTS, DIAMONDS, CLUBS OR SPADES {dn}"
36 INPUT S$
40 PRINT "{cyn}{dn}{dn}{dn}"
42 PRINT " NOW, TELL ME THE VALUE 1 TO 13 {dn}"
44 INPUT V
50 PRINT "{clr}{yel}{dn}{dn}{dn}"
52 PRINT " PLAYER2: GUESS THE CARD. WHICH SUIT? {dn}{pur}"
54 PRINT " HEARTS, DIAMONDS, CLUBS OR SPADES {dn}"
60 INPUT GS$: IF GS$=S$ THEN GOTO 70
62 PRINT "{grn}{dn}{dn}"
64 PRINT " WRONG, GUESS AGAIN. WHICH SUIT? {dn}{pur}"
68 LET G=G+$1: GOTO 60
70 PRINT "{clr}{yel}{dn}{dn}{dn}"
72 PRINT " RIGHT! NOW, WHAT VALUE 1 TO 13? {dn}"
80 INPUT GV: IF GV=V THEN GOTO 90
82 PRINT "{grn}{dn}{dn}"
84 IF GV<V THEN PRINT " WRONG, GUESS HIGHER. WHAT VALUE? {dn}{yel}"
86 IF GV>V THEN PRINT " WRONG, GUESS LOWER. WHAT VALUE? {dn}{yel}"
88 LET G=G+$1: GOTO 80
90 PRINT "{clr}{yel}{dn}{dn}{dn}"
92 PRINT " CONGRATULATIONS!"
94 PRINT " YOU GUESSED IT IN ONLY";G;"TRIES {dn}{dn}"
96 PRINT " PLAY AGAIN";: INPUT A$: IF A$="YES" THEN GOTO 10
98 PRINT "{dn}{dn}{dn}{dn}"
99 PRINT " SEE YOU AGAIN SOON! {wht}"

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

```
1 REM A13-3
10 REM *** PAPER, SCISSORS AND ROCK ***
15 POKE 53281,0
16 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 PRINT " PLAY THE {dn}{dn}"
25 PRINT TAB(4);"P A P E R {dn}{dn}"
26 PRINT TAB(13);"S C I S S O R S {dn}{dn}"
28 PRINT TAB(22);"R O C K {dn}{dn}"
30 PRINT " GAME AGAINST THE COMPUTER {dn}"
44 REM MAKE YOUR CHOICE
45 PRINT "{dn}{dn} ENTER <P,S,R> {dn}{dn}"
46 INPUT Y$
47 FOR T=1 TO 100: NEXT T
49 REM COMPUTER MAKES CHOICE
50 LET C=INT(RND(8)*3)+$1
55 IF C=1 THEN LET C$="P"
56 IF C=2 THEN LET C$="S"
57 IF C=3 THEN LET C$="R"
60 REM WHO WINS?
61 IF Y$=C$ THEN GOTO 80
62 IF C$="P" THEN IF Y$="S" THEN GOTO 90
64 IF C$="S" THEN IF Y$="R" THEN GOTO 90
66 IF C$="R" THEN IF Y$="P" THEN GOTO 90
70 REM COMPUTER WINS
72 PRINT "{up}";TAB(4);"{cyn} COMPUTER WINS {wht}"
79 GOTO 46
80 REM TIE
82 PRINT "{up}";TAB(26);"{grn} IT'S A TIE {wht}"
89 GOTO 46
90 REM YOU WIN
92 PRINT "{up}";TAB(18);"{yel} YOU WIN {wht}"
99 GOTO 46
```

```
1 REM A15-2
10 REM !!! VACATION !!!
12 POKE 53281,0: PRINT "{clr}{wht}"
20 REM HEADING
22 PRINT "{dn} VACATION"
24 PRINT "{dn} CHOOSING"
26 PRINT "{dn} PROGRAM"
28 PRINT "{dn}{grn} CHOOSES YOUR VACATION BY THE AMOUNT"
30 PRINT "{dn}{yel} YOU WANT TO SPEND"
32 FOR T=1 TO 2000: NEXT T
35 REM INSTRUCTIONS
40 PRINT "{clr}{dn}{dn}{dn} ENTER THE AMOUNT IN DOLLARS"
42 PRINT "{dn}{yel} YOU WANT TO SPEND"
50 REM GET DOLLAR AMOUNT
51 PRINT
52 INPUT D
53 PRINT
60 M$=" FLIP PENNIES WITH YOUR KID BROTHER"
61 N$=" SPEND THE AFTERNOON IN BEAUTIFUL HOG      WALLOW, MI"
62 P$=" ENTER A PICKLE CONTEST IN                 SCRATCHYBACK, TN"
63 REM ETC.
69 V$=" BUY A COZY YACHT AND CRUISE THE      CARIBBEAN SEA"
70 IF D<.5 THEN PRINT M$: GOTO 90
71 IF D<5 THEN PRINT N$: GOTO 90
72 IF D<50 THEN PRINT P$: GOTO 90
79 PRINT V$
90 REM ENDING OF PROGRAM, SAY SOMETHING
91 REM FOR A CLOSING
```

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

```
1 REM A16-1
10 REM *** FLYING BALL ***
15 POKE 53281,0
20 PRINT "{clr}{wht}{dn}{dn}...{dn}"  {10 of them}
25 I=1
30 I=I+$1
35 PRINT "{sp} Q {lf}";
40 FOR T=1 TO 10: NEXT T
45 IF I<40 THEN GOTO 30
50 I=I-1
55 PRINT "Q {sp}{lf}{lf}{lf}";
60 FOR T=1 TO 10: NEXT T
65 IF I>310 THEN 50

1 REM A160-2
10 *** BLINKING NAME ***
15 POKE 53281,7
20 PRINT "{clr}"
20 N$="K A R E N"
30 PRINT "{hm}{grn}{dn}{dn}{dn}{dn}{dn}{dn}{dn} ";N$
35 FOR T=1 TO 500: NEXT T
40 PRINT "{hm}{red}{dn}{dn}{dn}{dn}{dn}{dn}{dn} ";N$
45 FOR T=1 TO 500: NEXT T
90 GOTO 30

1 REM A17A-1
10 REM *** RABBITS ***
15 POKE 53281,0: PRINT "{clr}{wht}"
20 FOR I=0 TO 100 STEP 5
30 PRINT I
40 FOR T=1 TO 200: NEXT T
50 NEXT I

1 REM A17B-3
10 REM *** CLIMBING NAME ***
12 PRINT "{clr}{dn}{dn}...{dn}"  {22 of them}
15 N$= " STANISLAUS MAZURSKY"
20 FOR I=1 TO 23
30 PRINT N$
35 PRINT "{up}{sp}{sp}...{sp}"  {20 spaces}
36 PRINT "{up}{up}{up}"
40 NEXT I
```

```
1 REM A17B-5
10 REM *** OPERA SOPRANO ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 FOR I=1 TO 3
21 POKE 53281,I+$5
22 PRINT "{dn} SING"
24 FOR T=1 TO 100: NEXT T
30 FOR J=1 TO 3
32 PRINT "{dn} TRA ";
34 FOR T=1 TO 100: NEXT T
40 FOR K=1 TO 3
42 PRINT "LA ";
44 FOR T=1 TO 100: NEXT T
50 NEXT K
52 PRINT
60 NEXT J
70 NEXT I
90 POKE 53281,0
```

```
1 REM A18-1
10 REM *** FAMILY ***
15 POKE 53281,1
20 PRINT "{clr}{blk}{dn}{dn}{dn} WHAT RELATION? {dn}"
22 PRINT "FATHER, MOTHER, SISTER, BROTHER, ETC. {dn}"
24 INPUT R$: PRINT
25 FLAG$="NO"
30 FOR I=1 TO 10
35 READ T$: READ N$
40 IF T$=R$ THEN PRINT " ";"{dn}": FLAG$="YES"
50 NEXT I
55 RESTORE
60 IF FLAG$="NO"THEN PRINT "{red} DON'T HAVE ONE {blk}{dn}"
70 FOR T=1 TO 2000: NEXT T
99 GOTO 20
100 DATA FATHER, EDWARD
101 DATA MOTHER, LOUISE
102 DATA SISTER, ANNE
103 DATA GRANDMOTHER, ADA
104 DATA GRANDMOTHER, CONSTANT
105 DATA GRANDFATHER, REGINALD
107 DATA AUNT, KAREN
108 DATA AUNT, SUSAN
109 DATA UNCLE, HARRY
110 DATA COUSIN, BOB
```

```
1 REM A19-2
10 REM *** FLY AWAY ***
12 C=54272: POKE C+$1,150
13 POKE C+$5,9: POKE C+$6,34
14 POKE C+$24,15
20 PRINT "{clr}{blk}{dn}{dn} . . . {dn}"; {22 of them}
25 PRINT "{pur}"
30 PRINT "UQI {lf}{lf}{lf}";
32 POKE C+$4,33
40 FOR T=1 TO 150: NEXT T
42 POKE C+$4,32
45 PRINT "{sp}{sp}{sp}{lf}{lf}{up}";
50 PRINT "JQK {lf}{lf}{lf}";
60 FOR T=1 TO 150: NEXT T
65 PRINT "{sp}{sp}{sp}{lf}{lf}{up}";
70 GOTO 30

1 REM A20-6
10 REM *** ARE YOU QUICK ***
11 PRINT "{clr}{dn}{dn}"
12 PRINT TAB(8);"HIT RETURN WHEN YOU SEE {dn}"
13 PRINT TAB(11);"THE QUESTION MARK {dn}{dn}"
14 PRINT TAB(13);"NOT TOO HARD!"
15 FOR I=1 TO 3000: NEXT I
16 PRINT "{clr}"
20 FOR I=1 TO RND(0)*9000+$500: NEXT I
30 PRINT "{hm}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}";TAB(19);
31 T=TI
32 INPUT A
40 T=TI-T
41 IF T<3 THEN PRINT "JUMPED THE GUN"
44 T=T/60
45 T=INT(T*100)/100
50 PRINT: PRINT
55 PRINT TAB(7);"YOUR TIME WAS";T;"SECONDS"
60 FOR I=1 TO 3000: NEXT I
99 GOTO 15

1 REM A21-2
10 REM *** ALL COLORS ***
15 S=53281
16 B=S-1
20 FOR I=0 TO 15
25 POKE B,I
30 FOR J=0 TO 15
40 POKE S,J
42 FOR T=1 TO 200: NEXT T
50 NEXT J,I
```

```
1 REM A22-1
10 REM *** COLORED BOX ***
12 POKE 53281,0: PRINT "{wht}"
20 INPUT "WHAT COLOR <1 TO 15>";C
30 INPUT "WHAT SIZE <2 TO 15>";S
32 T=INT(S*1.4)
35 POKE 646,C
40 PRINT "{clr}{dn}{dn}{dn}{dn}{dn}{dn}";TAB(10);
50 FOR I=1 TO T
52 PRINT "*";: NEXT I
53 PRINT: IF S<3 THEN GOTO 60
55 FOR I=1 TO S-2
57 PRINT TAB(10);"*";TAB(T+$9);"*": NEXT I
60 FOR I=1 TO T
62 PRINT TAB(10);"*";: NEXT I
90 PRINT "{wht}"

1 REM A23-1
10 REM *** MENU MAKER ***
12 POKE 53281,1
15 PRINT "{clr}{blk}{dn}{dn}{dn}"
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"
70 FOR T=1 TO 900: NEXT T
80 GOTO 15
99 REM MAKE A STAR OF THIS COLOR
```

```
1 REM A23-2
10 REM *** SILLY SENTENCES ***
11 S$=" "
12 PRINT "{clr}{dn}{dn}{dn}"
16 PRINT " SILLY SENTENCES {dn}"
17 PRINT " WANT INSTRUCTIONS? {dn}{dn}"
18 GET Y$: IF Y$="" THEN 18
19 IF Y$="Y" THEN GOSUB 100
20 PRINT " THE SUBJECT: {dn}"
22 PRINT " END WITH A PERIOD. {dn}{dn}"
24 GET L$: IF L$="" THEN 24
26 IF L$="." THEN 30
28 S$=S$+$L$
29 GOTO 24
30 S$=S$+$" "
32 PRINT " THE VERB: {dn}{dn}"
34 GET L$: IF L$="" THEN 34
36 IF L$="." THEN 40
38 S$=S$+$L$
39 GOTO 34
40 S$=S$+$" "
42 PRINT " THE OBJECT:"
44 GET L$: IF L$="" THEN 44
46 IF L$="." THEN 60
48 S$=S$+$L$
49 GOTO 44
60 S$=S$+$L$
80 PRINT "{clr}{dn}{dn}{dn}"
85 PRINT S$
99 END
100 REM INSTRUCTIONS
105 PRINT "{clr}{dn}{dn}{dn}"
110 PRINT " PARTS OF A SENTENCE {dn}"
115 PRINT " THREE PLAYER GAME {dn}"
120 PRINT " EACH PLAYER ENTERS PART OF A SENTENCE {dn}"
125 PRINT " NO ONE CAN SEE WHAT THE OTHERS ENTER {dn}"
130 PRINT " THE FIRST ENTERS THE SUBJECT {dn}"
135 PRINT " THE SECOND ENTERS THE VERB {dn}"
140 PRINT " THE THIRD ENTERS THE OBJECT {dn}"
150 PRINT " PRESS SPACE TO PLAY"
160 GET Y$: IF Y$="" THEN 160
190 PRINT "{clr}{dn}{dn}{dn}"
199 RETURN
```

```
1 REM A24B-1
10 REM *** SUBROUTINES ***
12 POKE 53281,0
15 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 REM MAIN PROGRAM
21 GOSUB 100
22 GOSUB 900
23 GOSUB 200
24 GOSUB 900
25 GOSUB 300
26 IF RND(0)<.5 THEN GOSUB 400
90 PRINT "{wht}"
99 END
100 REM
101 REM THE FIRST SUBROUTINE
102 REM
110 PRINT " NOW IS THE TIME {dn}"
199 RETURN
200 REM
201 REM THE SECOND SUBROUTINE
202 REM
210 PRINT " FOR {red} RED {wht} SMOKE {dn}"
299 RETURN
300 REM
301 REM THE THIRD SUBROUTINE
302 REM
310 PRINT " TO POUR OUT {dn}"
320 GOSUB 900
330 PRINT " OF YOUR COMPUTER!"
399 RETURN
400 REM
401 REM PERHAPS
402 REM
410 PRINT "{yel}{dn}{dn}{dn}"
420 PRINT " <I DON'T MEAN IT!>"
499 RETURN
900 REM
901 REM TIMER
902 REM
910 FOR T=1 TO 999: NEXT T
999 RETURN
```

```
1 REM A25-3
10 REM *** AIN'T GOT NO ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 PRINT " ENTER A SENTENCE {dn}"
22 PRINT " NO PUNCTUATION EXCEPT APOSTROPHE"
24 PRINT
32 INPUT S$
33 S$=S$+$" "
35 L=LEN(S$)
40 NN=0
42 S1=1: S2=1
45 FOR I=1 TO L
50 L$=MID$(S$,I,1)
55 IF L$=" " THEN S1=S2: S2=I+$1: GOSUB 200
60 NEXT I
65 PRINT "{dn}{dn}"
70 IF NN=0 THEN PRINT " NO NEGATIVE WORDS"
72 IF NN=1 THEN PRINT " A NEGATIVE SENTENCE"
74 IF NN=2 THEN PRINT " DOUBLE NEGATIVE"
76 IF NN>2 THEN PRINT " MANY NEGATIVES"
80 FOR T=1 TO 2000: NEXT T
198 GOTO 12
200 REM TEXT WORD
205 LW=S2-S1-1
210 W$=MID$(S$,S1,LW)
220 READ NW$
222 IF NW$="END" THEN RESTORE: RETURN
224 IF W$=NW$ THEN NN=NN+$1
230 GOTO 220
300 DATA NO, NOT, NEVER, NEGATIVE, DON'T, AIN'T, DOESN'T,
301 DATA NOTHING, SHOULDN'T, WOULDN'T, NONE, END

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

```
1 REM A26-2
10 REM *** ANSWERER ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 PRINT " ENTER A QUESTION {dn}" -
25 INPUT Q$
28 L=LEN(Q$)
30 REM TAKE OFF "?"
33 Q$=LEFT$(Q$,L-1)+$"."
36 REM LOOK FOR WORD END
39 FOR I=1 TO L
40 C$=MID$(Q$,I,1)
45 IF C$=" " THEN S1=I: I=L
46 NEXT
50 FOR I=S1+$1 TO L
52 C$=MID$(Q$,I,1)
54 IF C$=" " THEN S2=I: I=L
56 NEXT I
58 REM EXCHANGE FIRST AND SECOND WORDS
60 S$=MID$(Q$,S1+$1,S2-S1)
62 V$=LEFT$(Q$,S1)
63 PRINT
65 PRINT " ";S$+$V$+$RIGHT$(Q$,L-S2)

1 REM A26-3
10 REM *** PIG LATIN ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 PRINT " PIG LATIN {dn}"
25 INPUT " GIVE ME A WORD:";W$
35 L=LEN(W$)
40 FOR I=1 TO L
42 L$=MID$(W$,I,1)
45 T=L$="A" OR L$="E" OR L$="I" OR L$="O" OR L$="U"
50 IF T THEN GOTO 60
55 NEXT I
60 P=I-1
65 PL$=RIGHT$(W$,L-P)+$LEFT$(W$,P)+$"AY"
70 IF I=1 THEN PL$=W$+$"LAY"
80 PRINT: PRINT
85 PRINT " ";PL$;"{dn}"
90 GOTO 25
```

```
1 REM A26-4
10 REM *** DOUBLE DUTCH ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 PRINT " GIVE ME A SENTENCE: {dn}"
30 INPUT S$
35 L=LEN(S$)
40 PRINT: PRINT
45 SS$=""
47 REM CHECK LETTERS
50 FOR I=1 TO L
52 L$=MID$(S$,I,1)
60 IF L$="A" THEN 72
61 IF L$="E" THEN 72
62 IF L$="I" THEN 72
63 IF L$="O" THEN 72
64 IF L$="U" THEN 72
66 SS$=SS$+$L$
72 NEXT I
75 PRINT " IN DOUBLE DUTCH {dn}"
77 PRINT " ";SS$;"{dn}{dn}{dn}"
80 GOTO 20

1 REM A26-5
10 REM *** MENU ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 REM MAKE A MENU
30 PRINT " MAKE YOUR CHOICE: {dn}"
31 PRINT " <A> EAT AN APPLE {dn}"
32 PRINT " <B> TAKE A NAP {dn}"
33 PRINT " <C> CALL A FRIEND {dn}{dn}{dn}"
40 GET X$: IF X$="" THEN 40
41 X=ASC(X$)-64: IF X<0 THEN40
50 ON X GOTO 60,70,80
52 GOTO 40
60 PRINT " YOUR SISTER ATE THE LAST ONE"
61 END
70 PRINT " YOUR BED IS NOT MADE"
71 END
80 PRINT " YOUR FATHER IS ON THE PHONE"
81 END
```

```
1 REM A27-1
10 REM *** BACKWARD ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
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+$1-I,1)
50 NEXT I
55 B=VAL(B$)
57 PRINT "{clr}{dn}{dn}{dn}"
60 PRINT " ";N$
61 PRINT " +$";B$
65 PRINT " ";LEFT$("---------",L)
70 A=N+$B
72 A$=STR$(A)
75 IF LEN(A$)=L THEN PRINT " ";A$
76 IF LEN(A$)=L+$1 THEN PRINT " ";A$
80 END

1 REM A27-2
10 REM *** LEAP FROG ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 INPUT "GIVE ME A NUMBER";N
25 N$=STR$(N)
26 L=LEN(N$)
27 N$=RIGHT$(N$,L-1): L=L-1
30 FOR I=0 TO 39-L
32 PRINT "{hm}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}{dn}"
35 PRINT TAB(I);" ";N$
40 N$=RIGHT$(N$,L-1)+$LEFT$(N$,1)
45 FOR T=1 TO 300: NEXT T
50 NEXT I

1 REM A30-1
10 REM *** THIRTY DAYS HAS . . . ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
15 DIM D(12)
20 FOR I=1 TO 12
22 READ D(I)
25 NEXT I
30 INPUT " WHICH MONTH (1 TO 12)";M
31 PRINT: PRINT
35 PRINT " MONTH NUMBER";M;"HAS";D(M);"DAYS"
99 DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

```
1 REM A32-1
10 REM *** BIG, BIG NUMBERS ***
12 PRINT "{clr}{wht}{dn}{dn}{dn}"
20 FOR I=1 TO 17
25 FOR J=1 TO 3
27 D$=STR$(INT(RND(0)*10))
28 D$=RIGHT$(D$,1)
30 N$=N$+$D$
35 NEXT J
39 IF I=17 THEN 45
40 N$=N$+$","
45 NEXT I
50 N$=" "+$RIGHT$(N$,LEN(N$)-1)
60 PRINT N$

1 REM A32-2
10 REM *** CIPHER PROGRAM ***
20 GOSUB 1000
100 REM
101 REM MAIN LOOP
102 REM
110 GOSUB 400
120 PRINT " CODE OR DECODE <C/D> {dn}"
130 GET Y$: IF Y$="" THEN 130
140 IF Y$="C" THEN GOSUB 500: END
150 IF Y$="D" THEN GOSUB 600: END
199 GOTO 130
400 REM
401 REM GET PASSWORD
402 REM
405 INPUT " INPUT PASSWORD ";PW$
407 REM REMOVE REPEATED LETTERS
410 F$=LEFT$(PW$,1)
411 FOR I=2 TO LEN(PW$)
412 L1$=MID$(F$,I,1)
413 FOR J=1 TO LEN(F$)
414 L2$=MID$(F$,J,1)
415 IF L1$=L2$ THEN 418
416 NEXT J
417 F$=F$+$L1$
418 NEXT I
419 PW$=F$
430 PRINT: PRINT " SHORTENED PASSWORD: ";PW$;"{dn}{dn}"
435 REM FORM THE CODE ALPHABET
437 A$=PW$+$A$
440 F$=LEFT$(A$,1)
```

```
441 FOR I=2 TO LEN(A$)
442 L1$=MID$(F$,I,1)
443 FOR J=1 TO LEN(F$)
444 L2$=MID$(F$,J,1)
445 IF L1$=L2$ THEN 448
446 NEXT J
447 F$=F$+$L1$
448 NEXT I
449 A$=F$
470 PRINT " CIPHER ALPHABET (dn)"
471 PRINT " ";A$
472 PRINT " ";B$;"(dn)"
473 PRINT TAB(14);"PLAIN ALPHABET (dn)(dn)(dn)"
499 RETURN
500 REM
501 REM CODE A MESSAGE
502 REM
505 PRINT " INPUT MESSAGE, END WITH '*' SIGN (dn)"
510 GET L$: IF L$="" THEN 510
511 L=ASC(L$)
515 IF L$="*" THEN 590
520 IF L<65 OR L>91 THEN P$=P$+$L$: GOTO 540
530 P$=P$+$MID$(A$,L-64,1)
540 PRINT L$;
550 GOTO 510
590 PRINT
591 PRINT P$
599 RETURN
600 REM
601 REM DECODE A MESSAGE
602 REM
605 PRINT " TYPE CODED MESSAGE, END WITH '*' SIGN (dn)"
610 GET L$: IF L$="" THEN 610
615 IF L$="*" THEN 699
620 FOR I=1 TO 26
625 IF L$=MID$(A$,I,1) THEN PRINT MID$(B$,I,1);: GOTO 610
630 NEXT I
640 PRINT L$;
650 GOTO 610
699 RETURN
1000 REM STARTING STUFF
1010 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
1020 B$=A$
1030 PRINT "(clr)(wht)(dn)"
1999 RETURN
```

INDEX OF COMMANDS AND FUNCTIONS

KEYS EXPLAINED IN THIS BOOK

INDEX OF TOPICS

KIDS AND THE COMMODORE 64

A KID'S BOOK FOR ADULTS TOO? YOU BET!

The title of this book may be KIDS & THE COMMODORE 64, but don't let it fool you. Designed for children ages 10 to 14, it also provides valuable information for the adult purchasing his or her first Commodore 64. KIDS & THE COMMODORE 64 was created to lead you gently yet quickly into the world of Commodore BASIC.

You'll learn how to write action games, board games and word games. Guidance, explanations, exercises, reviews and quizzes are given in an easy going, non-technical style. Study guides precede each chapter, and every new concept is tied to simple everyday experiences. KIDS & THE COMMODORE 64 is illustrated with dozens of clever cartoons — so you'll laugh as you learn!

Computers are here to stay. KIDS & THE COMMODORE 64 prepares the computer generation by solving the mysteries of the Commodore 64 in entertaining and enjoyable ways!

ABOUT THE AUTHOR: Edward H. Carlson has been a Professor of Physics at Michigan State University since 1965. His interest in computers began in 1960, and he has since been involved in many University projects. He recently helped establish a Computer Camp for Children.

OTHER POPULAR COMPUTER BOOKS BY DATAMOST:

Kids & the Apple
Kids & the Atari
Kids & the Panasonic
Kids & the TI 99/4A
Kids & the VIC 20
by Ed Carlson

How to Write an Apple Program
How to Write an IBM-PC Program
How to Write a TRS-80 Program
How to Write a Program Vol. II
by Ed Faulk

Using 6502 Assembly Language
p-Source
by Randy Hyde

The Elementary Apple
The Elementary Atari 400, 800, 1200
The Elementary Commodore 64
The Elementary TI 99/4A
The Elementary Timex/Sinclair
The Elementary VIC 20
by William Sanders

Computer Playground Apple
Computer Playground Atari
Computer Playground C-64/VIC 20
Computer Playground TI 99/4A
by M.J. Winter

Games Apples Play
by Mike Weinstock & Mark Capella

ISBN 0-88190-172-5

DATAMOST™
8943 Fullbright Ave., Chatsworth, CA 91311
(213) 709-1202 (800) 692-1649

Carlson
KIDS AND THE COMMODORE 64
DATAMOST