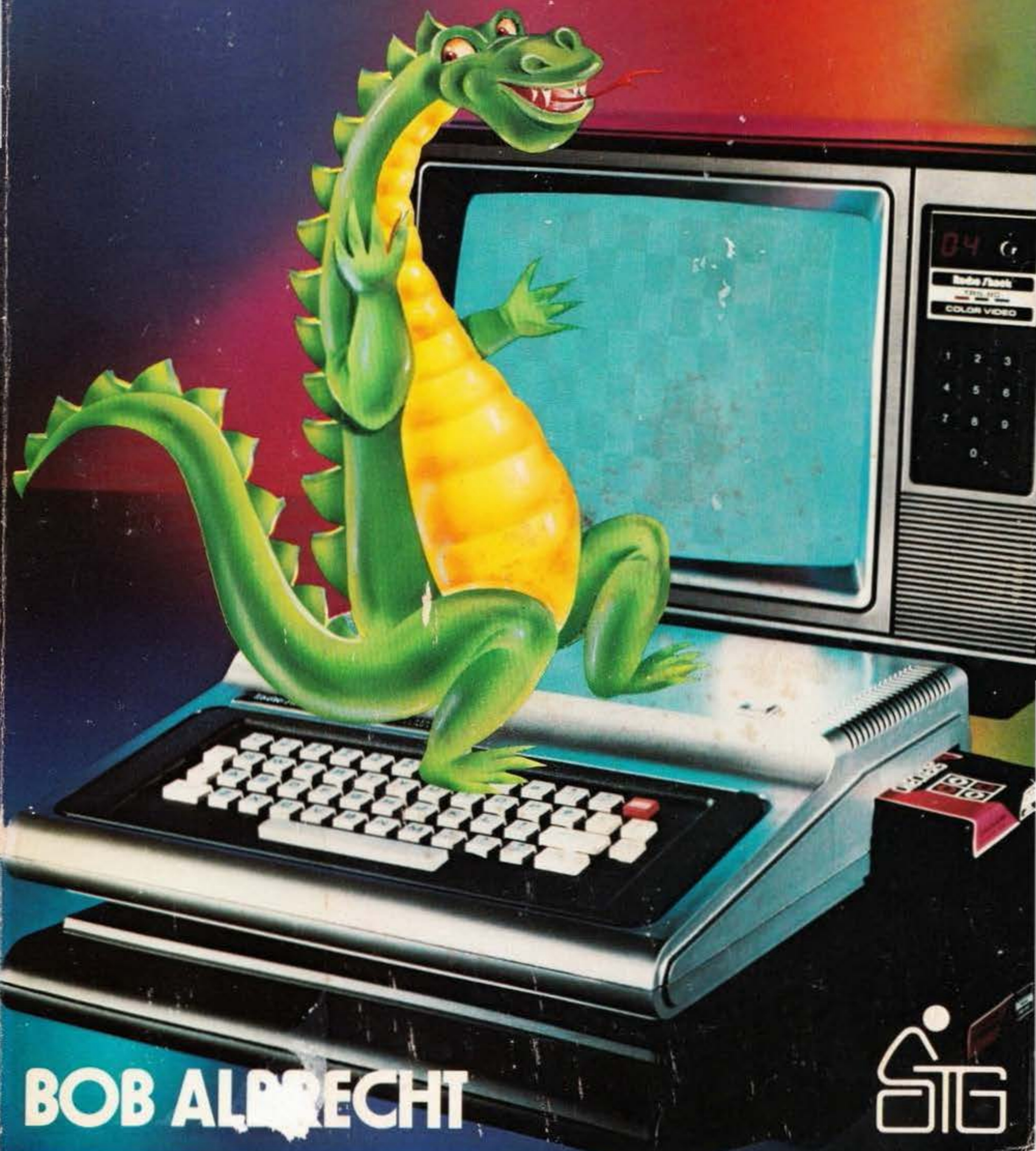


TRS-80TM COLOR BASIC



BOB ALBRECHT



TRS-80TM Color BASIC

BOB ALBRECHT

Dymax Corporation
Menlo Park, California



John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Judy V. Wilson
Editor: Dianne Littwin
Composition and Make-up: Cobb/Dunlop, Inc.

TRS-80™ is a trademark of Tandy Corp.

Copyright © 1982, by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data

Albrecht, Bob, 1930—

TRS-80 color BASIC.

(Wiley self-teaching guides)

Includes index.

1. TRS-80 (Computer)—Programming—Programmed instruction. 2. Basic (Computer program language)—Programmed instruction. I. Title. II. Title: T.R.S.-80 color B.A.S.I.C. III. Title: TRS-eighty color BASIC. IV. Title: T.R.S.-eighty color B.A.S.I.C.

QA76.8.T18A423

001.64'1

81-16286

ISBN 0-471-09644-X

AACR2

Printed in the United States of America

82 83 10 9 8 7 6 5 4 3 2

Contents

	To the Reader	iv
	How to Use This Book	v
Chapter 1	The TRS-80 Color Computer	1
Chapter 2	Easy Stuff	11
Chapter 3	BASIC Programs	35
Chapter 4	Number Boxes	59
Chapter 5	String Boxes	85
Chapter 6	Skipping Round the Screen	115
Chapter 7	Graphics Galore	143
Chapter 8	Meandering	177
Chapter 9	Playtime Junction	205
Chapter 10	String Functions	239
Chapter 11	Subscripted Variables	273
Chapter 12	Computing Problems and Challenges	299
Chapter 13	The Color BASIC Toolbox	323
Appendix A	Hooking Up Your Color Computer	346
Appendix B	Using the Tape Cassette Recorder	350
Appendix C	Error Messages	354
Appendix D	Arithmetic	357
Appendix E	Reserved Words	360
Appendix F	Screen Maps	361
Appendix G	Sound and Music	364
Appendix H	Color Codes and Graphics Characters	366
Appendix I	ASCII Codes	368
Appendix J	Joysticks	370
Appendix K	Look Here First	375
	Index	377

To the Reader

This book is for people who want to learn how to use, program, and enjoy the Radio Shack Color Computer. Using this book, you can teach yourself how to read and understand BASIC, the language of the Color Computer and many other computers. No previous computer experience is required.

The Color Computer is friendly, fun, easy to use by kids and adults, at home, school, or elsewhere. It is a superb low-cost computer for educational and recreational use.

That's really what this book is all about—to help you learn to use the Color Computer for your own recreation and education. So play and learn your way through this book—have a good time!

You will learn BASIC, a language you use to tell the computer what you want it to do. Since BASIC is a language, learn it as you might learn any language, such as English, Spanish, or Swahili.

- Learn a little bit and use it.
- Learn a little more and use it.
- And so on. Be patient. There's no hurry. Becoming fluent in BASIC takes some time, but you can enjoy every moment.
- Be confident. BASIC is a simple language, much more easily learned than English, Spanish, or Swahili.

This book is *not* a reference manual. It is *not* a textbook. It doesn't even try to cover *all* of Color Computer BASIC. *That* would take three or four books this size, if we explained everything as slowly and carefully as we do in this book.

If you can read a newspaper or a comic book, you can use this book to help you teach yourself to read, understand, and use BASIC. Before you begin, browse through Appendix K, "Look Here First," for sources of additional information to help you learn.

Wait! Before plunging into Chapter 1, read "How to Use This Book."

How to Use This Book

This is a Self-Teaching Guide. You can use this book to teach *yourself*. Each chapter of this book (except Chapters 12 and 13) is composed of short numbered sections called *frames*. Each frame is a single idea, topic, or problem. At the end of most frames are questions for *you* to answer or things for *you* to do on the Color Computer.

If a frame asks you to do something on the computer, *do it!* The Color Computer itself is your best teacher. Experiment—try it and find out what happens.

If a frame asks questions, answer them. We've left some space for your answers throughout the book, but if you need more, use a separate sheet of paper. Don't peek at *our* answers until you have written *your* answers. Then compare. You will find our answers below the dotted line. Sometimes our answer is the "right" answer; sometimes our answer is one of many "right" answers. Yours may be just as "right" or even better. You decide.

We encourage you to use this book while seated comfortably in front of a Radio Shack Color Computer. Try our examples and exercises. You and the Color Computer will know what works and what doesn't work.

The first page of each chapter briefly lists what the chapter covers. Scan that list. If you feel you already know it, skip to the back of the chapter and take the Self-Test. The answers to the Self-Test list the frame numbers within the chapter that relate to the question. If you miss a Self-Test question, review those frames. You may also wish to browse through the chapter to find the little treasures (variations, challenges, puns, and so on) that we have buried there for you to find.

Things get more challenging as you progress through the book. If you are a beginner, start with the first chapter and work (play!) your way through the book. If you already know some BASIC, feel free to browse, skip around, meander through the book. Use the beginning-of-chapter objectives list and the end-of-chapter Self-Test as your guides.

Of course, look at the table of Contents. You will see the titles of 13 chapters and 11 appendixes. Curious? Go ahead—explore.

This book is full of challenges. They range from easy to hard to awful. You will find challenges *with* solutions and challenges *without* solutions. Many of our challenges are unusual. And many of our solutions are unusual, especially in Chapter 12. If you are a puzzler—a person who likes to solve problems (especially in unusual ways!)—we think you will like our challenges.

Chapter 13 is a "mini-reference manual." It contains brief descriptions of all important words in the language of BASIC for the lowest-cost Radio Shack Color Computer (Catalog Number 26-3001.) It does *not* contain words in Extended Color BASIC for the more expensive Color Computer (Catalog Number 26-3002). Use Chapter 13 for quick reference.

One more thing. As you use this book and your Color Computer, know this:

YOU CAN DO NOTHING WRONG!

You can't harm the computer by typing stuff into it. You may make mistakes, but that is a natural part of exploring and learning. Risk it! Try it and find out what happens. You can learn more from your own patient exploration than from this or any book.

So, explore, enjoy, and tell us about your discoveries as you teach yourself how to use, program, and enjoy your Color Computer.

Bob Albrecht and Friends
Dymax Gazette
P.O. Box 310
Menlo Park, CA 94025

CHAPTER ONE

The TRS-80 Color Computer

Relax, make yourself comfortable. The first chapter is especially easy! In this chapter, you begin learning how to use, program, and enjoy the Radio Shack TRS-80 Color Computer.* Get ready for an adventure in sound, color, and convivial computing!

The Color Computer is friendly, fun, easy to learn and use by kids and adults, at home or at school. Although designed primarily for educational and recreational uses, it can also do "serious stuff" in the worlds of math, science, home and personal management, and small business.

You will also begin to learn *computerese*, the jargon or terminology of computers. You will be able to better understand the literature of the computer age and enjoy conversing about the exciting things you are doing with your Color Computer.

When you finish this chapter, you will know a few things about the TRS-80 Color Computer and be able to use the following words and phrases:

- TRS-80 Color Computer System
- Keyboard
- Television (TV) output
- Microcomputer and microcomputer chip
- Program PAKS^{™†}
- Cassette recorder
- Joysticks
- BASIC (TRS-80 Color BASIC)
- Memory
- Read-Only Memory (ROM)
- Random Access Memory (RAM)
- BASIC program

*This book is about the TRS-80 Color Computer, Radio Shack catalog Number 26-3001. We do NOT cover *Extended BASIC*. However, everything in this book will work on the Color Computer with *Extended BASIC* (Catalog Number 26-3002).

†Program PAKS are a product of the Tandy Company.

And at the end of this chapter, you will be ready to continue with Chapter 2, in which you begin "talking" to the Color Computer!

1. You can start with only a Color Computer and a television. You will have lots more fun if the television is a *color* TV!



TRS-80 color computer



What are the parts of a minimum Color Computer system?

- (a) _____
- (b) _____

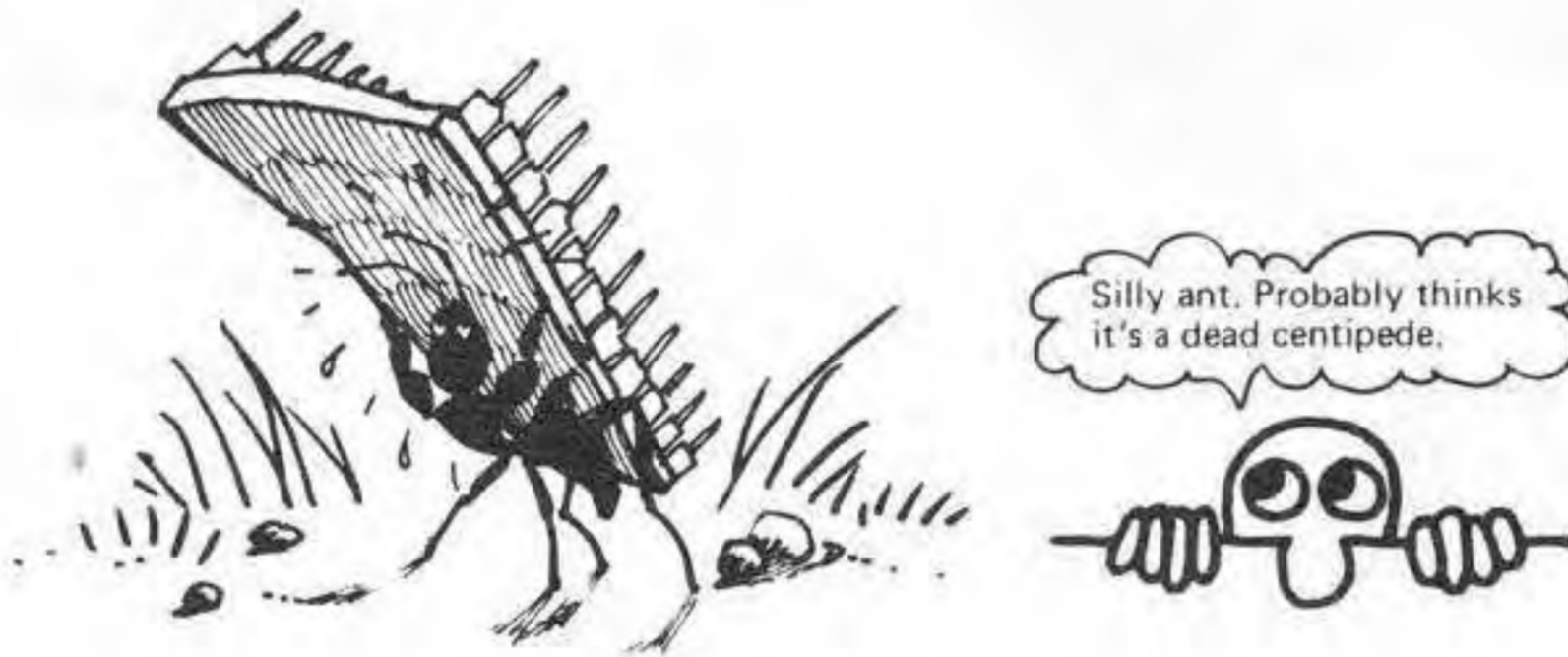
- (a) Color Computer
- (b) color TV (preferred) or black and white TV

Of course, these two parts must be *connected*. How to do this is explained in Appendix A and also in the *TRS-80 Color Computer Operation Manual* that accompanies the Color Computer. We assume that you already have this. If not, ask about it at your friendly neighborhood Radio Shack store or write to Radio Shack, Fort Worth, TX 76102.

2. Inside the Color Computer case is . . . the computer! It consists of a bunch of "chips." A chip is a tiny wafer containing thousands of *microscopic* circuits. Some people decided to call this a *microcomputer*. Others call it a *microprocessor*.

The tiny wafer is about this big:

Well, that's too small for clumsy human fingers to handle. So the tiny wafer is put into a larger package with lots of legs that can be used to connect it to the other parts of the computer.



Check all appropriate answers. A microcomputer chip, in its centipede-like package is:

- (a) smaller than a tennis ball.
 (b) about the size of a microscope.
 (c) too small to be seen by human eyes.
 (d) about the shape and size of a centipede.
 (e) delicious with avocado dip.
-

We suspect you checked (a) and (d). What? You also checked (e)? Oh well, tastes vary.

3. As you work (or play) through this book, you will learn a computer language called BASIC. A computer language is simply a language you use to communicate with a computer. Compared to natural languages (English, Spanish, Swahili, etc.), a computer language is very simple. BASIC has a simple vocabulary (the list of words it knows) and a very formal *syntax* (rules of grammar that it follows). This book will help you enjoy teaching yourself how to "talk" to computers, using the language BASIC.

What is BASIC? _____

a simple computer language, or a language used in communicating with a computer

There are many variations, or *dialects*, of BASIC. In this book, you will learn TRS-80 Color BASIC for a minimum TRS-80 Color Computer (Radio Shack Catalog Number 26-3001).

IMPORTANT NOTICE

This book is about TRS-80 Color BASIC on a *minimum* TRS-80 Color Computer. It is NOT about Extended BASIC on the larger, more expensive TRS-80 Color Computer (Catalog Number 26-3002). And now the good news. Everything in this book will also work in Extended Color BASIC!

4. As you learn BASIC, you will learn to read and understand *computer programs* written in BASIC. As a child learns to read and understand a natural language such as English, you will learn to read and understand BASIC.

Simple things first.
Then a little more.
Then a little more.
And so on, forever.

Like the child, as you begin to understand, you may wish to express yourself in the language you are learning.

You may write your own, original, never-before-seen-on-Earth-or-anywhere-else programs—*your* programs!

Program? A program is simply a procedure, a set of instructions, a plan for doing something. You may have already used, or created, or been frustrated by programs written in English (or another language). For example:

- A recipe for baking a cake.
- Instructions for opening a combination lock.
- Directions on how to get to your house from the airport.
- And, of course, those maddening instructions for assembling toys, tricycles, playpens, furniture, and so on—probably at the last minute on Christmas Eve.

A BASIC program is a set of instructions that tells the computer what to do and how to do it in the language the computer understands—BASIC.

Your instructions to make the computer do what you want it to do, following the rules of BASIC, is called a _____.

program or BASIC program

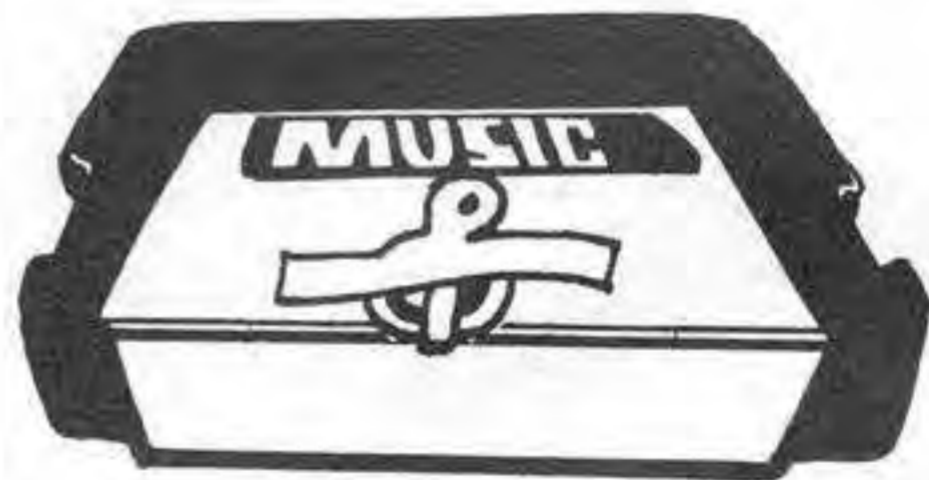
5. BASIC is “built-in” to the TRS-80 Color Computer. The vocabulary and syntax (rules of grammar) are stored in the memory of the Color Computer. BASIC is stored (or “remembered”) in a bunch of chips called ROM, or Read Only Memory. Information in ROM is permanently stored, much like the information in a printed dictionary or on a phonograph record. Have you ever used a pocket calculator? In a calculator, the rules for arithmetic and other functions are permanently stored in ROM. When you turn on the TRS-80 Color Computer, it immediately “knows” BASIC—simply by looking into its ROM.

What is ROM? _____

Read Only Memory

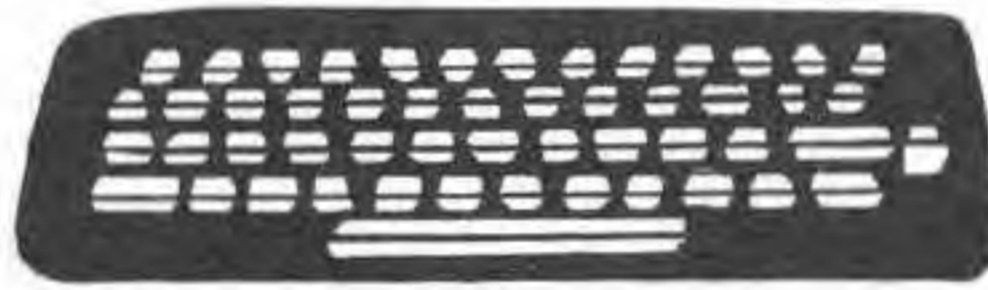
The Color Computer can read information from ROM but cannot erase it or change it in any way. That is why this kind of memory is called *Read Only Memory*.

6. The Color Computer has a slot for plug-in ROM cartridges called Program PAKs.TM A Program PAK contains a program that is permanently stored in its ROM. The ROM chips are inside the cartridge. You simply plug it in and go! In this book, you will not use Program PAKs. For information on using Program PAKs, look in the *TRS-80 Color Computer Operation Manual*, which comes with the Color Computer.



Program PAK

7. You will use the keyboard to communicate with the Color Computer. The keyboard is your control center. You use the keyboard to type information into the computer. As you type, the information that you type is stored in the computer's memory. This type of memory is called RAM, which means *Random Access Memory*. RAM is different from ROM. You can put information into RAM, but not into ROM. You can also erase or change information in RAM. RAM is like a blackboard or a scratch pad. Yes, RAM is just more chips. Good grief! Will those chips ever stop? Actually, RAM should have been called *Read/Write Memory*, or RWM, because information can be read from it, or written into it. Unfortunately, RWM is hard to pronounce.



Answer the following questions by writing RAM and/or ROM.

- (a) Which can be erased or changed? _____
- (b) Which is permanent and can't be changed? _____
- (c) From which can information be read? _____
- (d) Which is used to store information you type on the keyboard? _____

(a) RAM; (b) ROM; (c) RAM and ROM; (d) RAM

8. What you type on the keyboard is stored in RAM. It also appears on the TV screen so that you can see what you type. As you will soon see, the computer also prints information on the TV screen. Together, the keyboard and the TV screen provide two-way communication with the Color Computer.

When you type on the keyboard, the stuff that you type is stored in the _____ of the Color Computer and is also displayed, or printed, on the _____.

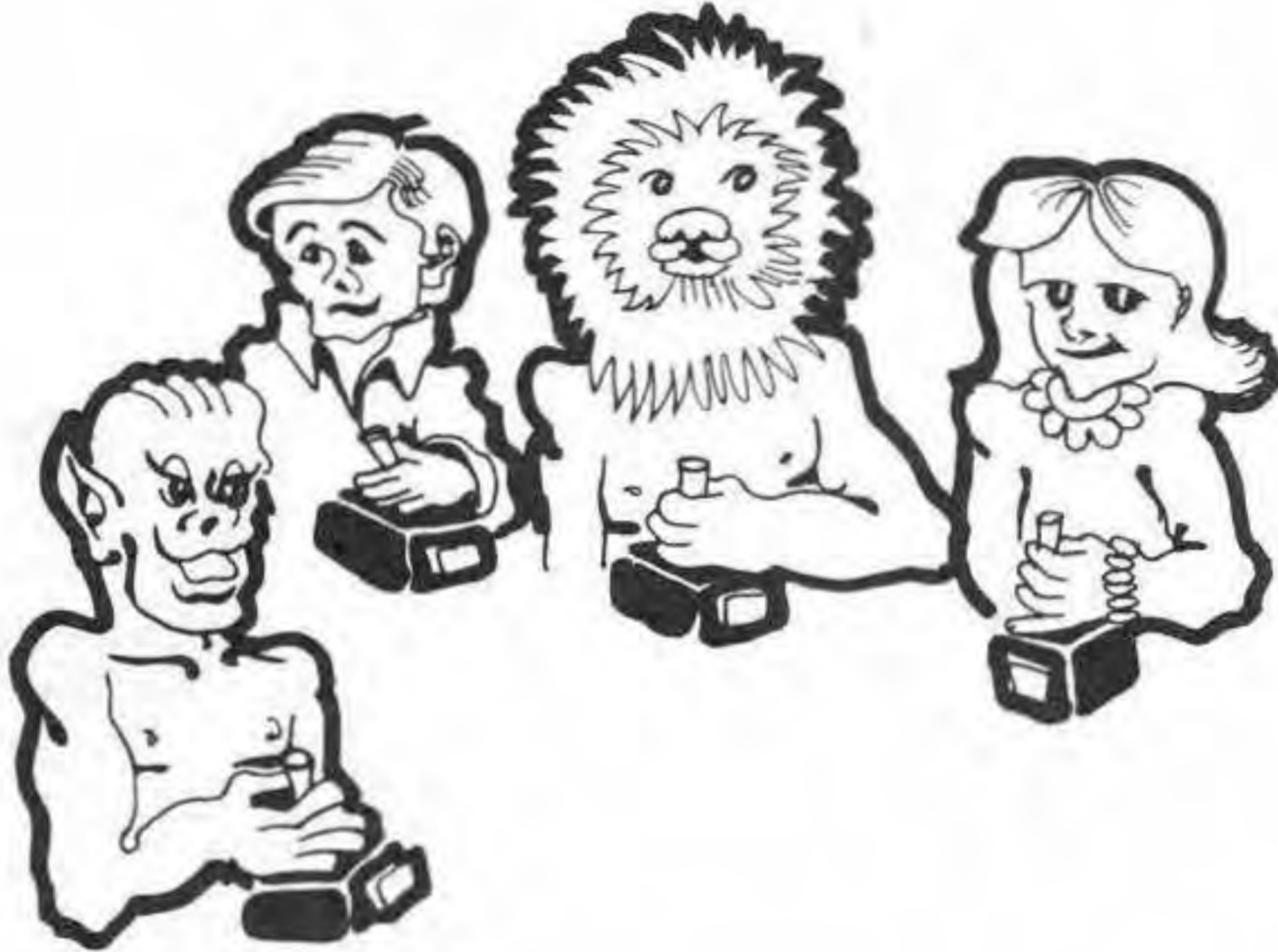
memory (or RAM); TV screen

9. Many programs are available on prerecorded tape cassettes. A single cassette, which might cost from \$3 to \$30, might contain one large program or several small programs. These programs can be entered into the Color Computer by means of a cassette recorder.



Using the cassette recorder, you can quickly load a program into your Color Computer, then enjoy the use of a program written by an expert. As you learn to program in Color BASIC, you can use the cassette recorder to record *your* programs on tape cassettes. Then, when you wish to use your program again, you use the cassette recorder to read the program back into the computer. Reduce finger fatigue! Use the cassette recorder. Appendix B tells you how. Relax! No questions. Read on.

10. Did you get a pair of joysticks with your Color Computer? These are fun and useful control devices for games and other activities in which things are happening on the screen.



11. OK, you have now had a brief introduction to the TRS-80 Color Computer and to the jargon of computers, *computerese*. If you have not already done so, hook up your Color Computer and plunge into the next chapter. If you don't know how to hook up your computer, then:

- read Appendix A, “Hooking Up Your Color Computer”; or
- read “Installation” in the *TRS-80 Color Computer Operation Manual*, available at most Radio Shack Stores; or
- yell for help! There are so many Color Computers in use, someone might hear you.

Before you move on to Chapter 2, you may wish to take the following Self-Test so that you can amaze and delight yourself by how much you have already learned.

Self-Test

1. The Color Computer is a microcomputer made up of a bunch of chips. A microcomputer chip (check the one that applies best):
 - (a) can be carried by a strong ant.
 - (b) contains thousands of very tiny electronic circuits in a small space.
 - (c) can be seen only by mountain sheep (RAMs?) with telescopic eyesight.
 - (d) is manufactured by leprechauns using miniature tools.
 - (e) requires only a droplet of avocado dip.
2. You "talk" to the Color Computer by using a _____
 — called BASIC. The vocabulary and syntax (rules) for this language are located in a special kind of memory. What is this memory called?
3. How do you enter information into the Color Computer? _____

4. When you type information into the Color Computer, where is it stored? _____

5. What is a BASIC program? _____

6. What is the function and purpose of the cassette recorder?

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. The most appropriate answer is probably (b) contains thousands of very tiny electronic circuits in a small space. However, if you are a microcomputer pirate using trained ants to pilfer parts, answer (a) might be more appropriate. (frame 2)
2. language; ROM
 When you turn off the Color Computer (or trip over the power cord), information in RAM disappears. Information in ROM is still there. (frames 3, 4, 5)
3. Type it on the keyboard. Well, you can also plug in a Program PAK or read information using the cassette recorder. (frames 6, 7, 8, 9)
4. RAM. (And as you type, your information, replete with typos, also appears on the TV screen.) (frames 7, 8)
5. A set of instructions that tells the computer what to do and how to do it. (It is the program that makes the computer appear smart or dumb, friendly or arrogant, helpful or stand-offish, passive or interactive.) (frame 4)

6. You use it to get information into the computer's memory (RAM) or to save information already in the memory, on tape. Your aching fingers will testify to the usefulness of devices such as the cassette recorder. Your impatient mind will sometimes grumble at its slowness. (frame 9)
-

CHAPTER TWO

Easy Stuff

Now your fun begins. In this chapter, you will begin learning how to “talk” to your Color Computer. You will do this by typing instructions called *direct statements*. Direct statements are also called *immediate statements*. These statements are direct or immediate because the computer executes them (does, or obeys them) immediately after you type a statement and then press the **ENTER** key on the keyboard.

A direct statement tells the Color Computer to do something. The computer does it immediately, then waits for your next instruction.

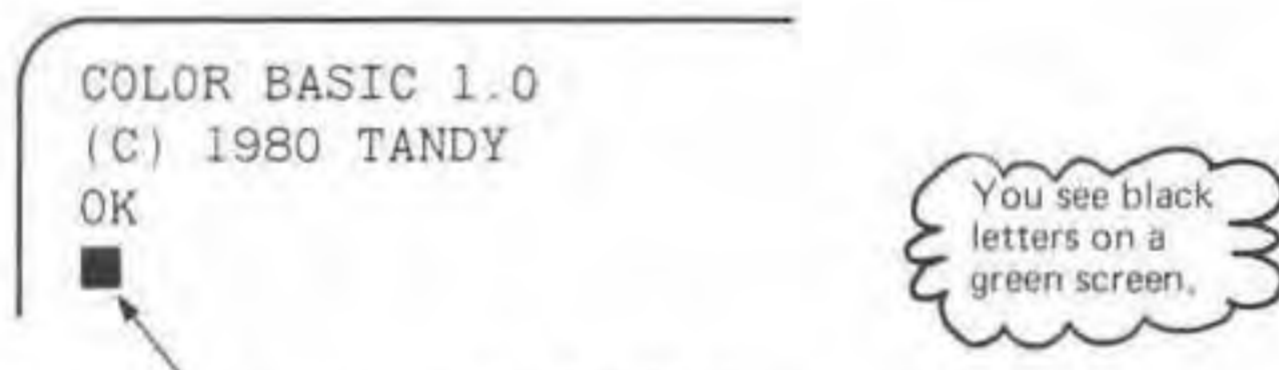
When you finish this chapter, you will be able to:

- Clear the TV screen;
- Change the color of the TV screen;
- Tell the computer to make musical sounds;
- Use direct PRINT statements to tell the computer to print information on the TV screen;
- Recognize error messages from the computer (in case you make an error or tell the computer to do something it doesn't understand);
- Correct typing errors or delete a direct statement that contains errors;
- Look forward with confidence to the next exciting chapter.

We assume that your Color Computer is set up and ready to go. If not, hook it up now. You can find directions on how to do it in Appendix A of this book or in your *TRS-80 Color Computer Operation Manual*.



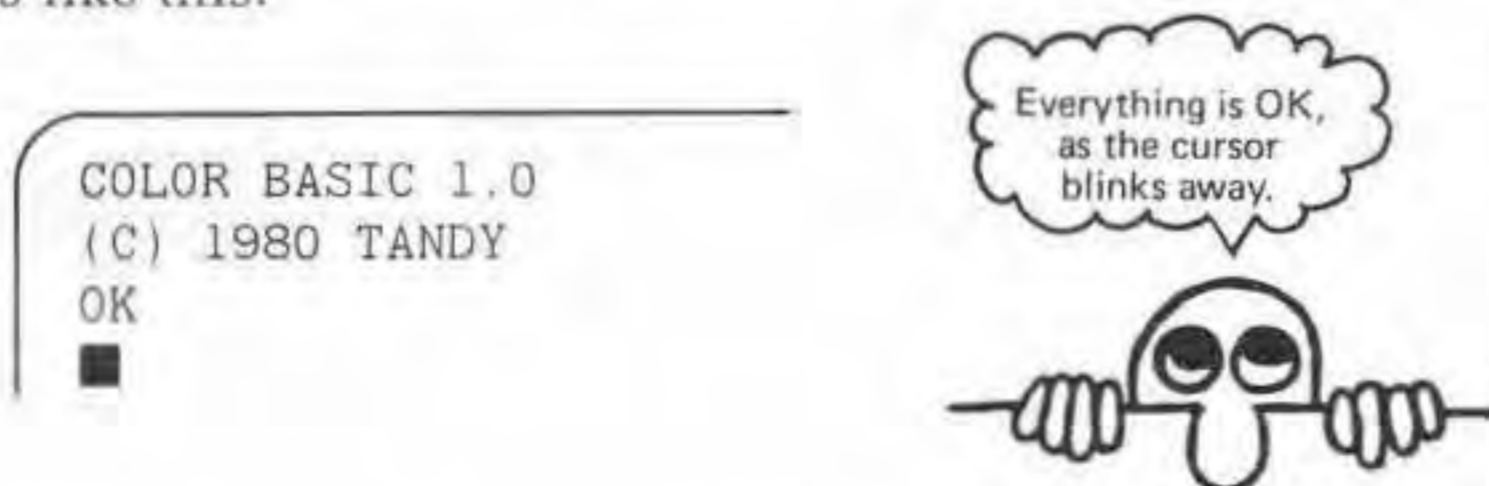
1. Ready? Begin. Turn on the TV and the Color Computer.* This is what you should see:



This is the cursor, which blinks on and off.

If you don't see this (or something quite similar), turn everything off and carefully check to see if the Color Computer and the TV are properly hooked up (see Appendix A). Then try again!

2. Welcome back. We now assume that everything is OK and the screen looks like this:



*The power switch for the Color Computer is in the back, to the left as you face the keyboard.

If your screen says COLOR BASIC 1.2 or COLOR BASIC 2.3 or something similar—relax. You simply have a later model than the one we used to write this book. The important things are the word OK and the cursor, which blinks merrily, continually changing colors.

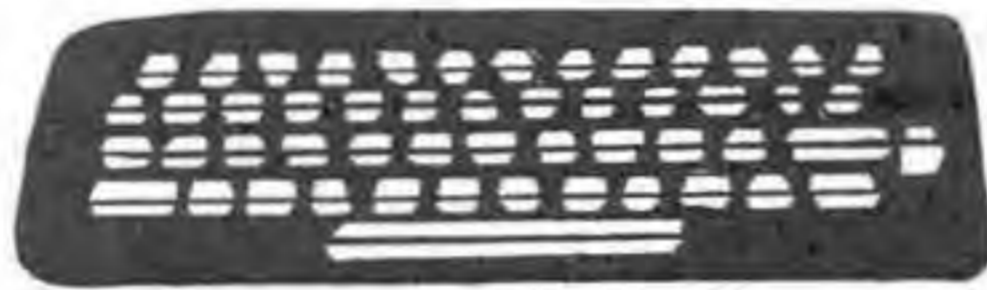
Whenever you see the cursor, you know that it is your turn to do something. If you don't do something, the computer will simply wait patiently until you are ready to use it.

What does the cursor look like? _____

a rectangle that blinks and changes color

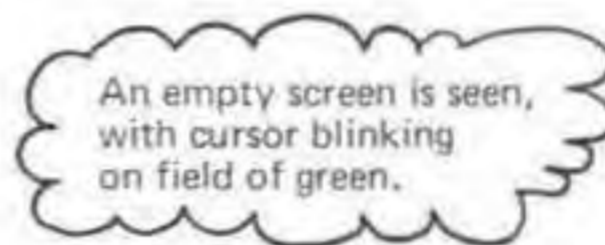
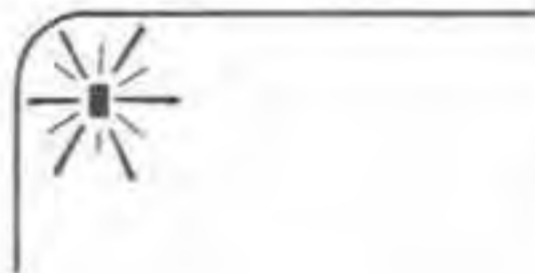
3. The keyboard is your control center.

Let your eyes wander over the keyboard. Find the **CLEAR** key on the right side of the keyboard.



Press the **CLEAR** key.

The screen now looks like this:

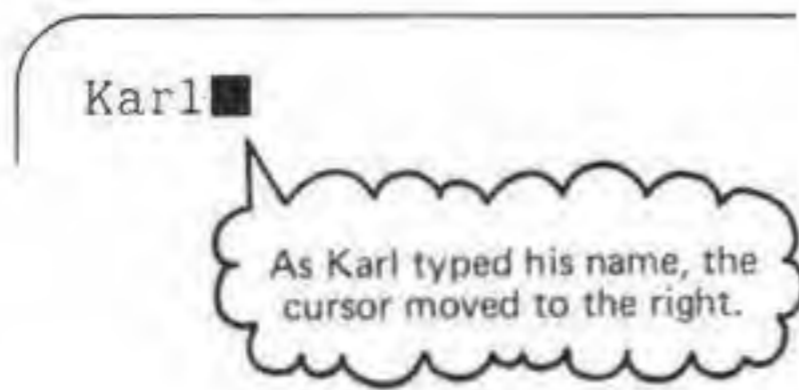


Pressing the **CLEAR** key clears, or erases, the screen. Only the blinking cursor remains on a green screen. This is *all* that happens. Pressing the **CLEAR** key does not clear or erase anything *inside* the computer.

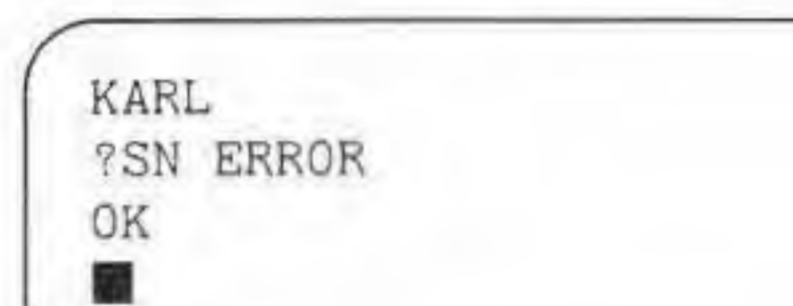
You know that it is your turn to do something because you can see the _____ on the screen.

cursor

4. So do something. Type your name and press the **ENTER** key. Here is what happened when Karl typed his name:



Then Karl pressed the **ENTER** key:



The strange message ?SN ERROR means that the Color Computer did not understand the word KARL. Karl (who is thirteen and proud of his name) was quite surprised. After all, how could *his* name be an error, much less an SN ERROR, whatever that is?

We explained to Karl that the computer didn't understand him. The word KARL is not one of those special BASIC words that the computer understands. "Aha!" exclaimed Karl, and he, as you will also, began to learn about those few special words that the Color Computer *does* understand.

- (a) If you type something, then press the **ENTER** key, and the computer types ?SN ERROR, what is it trying to tell you? _____
- (b) After typing ?SN ERROR, what does the computer do? _____
-

- (a) It does not understand you.
- (b) It types OK, then turns on the cursor.

The OK and the cursor let you know that everything is OK and it is again your turn. SN ERROR means SYNTAX ERROR. You will get this message when the computer does not understand you. Many other error messages are possible. You will meet others as you experiment with your Color Computer. See Appendix C for a complete list of error messages.

5. The Color Computer understands *Color BASIC*. In Color BASIC, there are special words to tell the computer to do things.

A special BASIC word → **PRINT**

PRINT tells the computer to print something on the TV screen. Here is how to get the computer to print a name on the screen. Lizzie will demonstrate. First, Lizzie clears the screen by pressing the **CLEAR** key.

The screen is clear, except for the cursor.



Then she types:

PRINT "LIZZIE" ■

Note that LIZZIE is enclosed in quotation marks.

To type a quotation mark ("), hold down the **SHIFT** key and press **2**.



Then she presses the **ENTER** key.

Lizzie typed this.
The computer did the rest.

```
PRINT "LIZZIE"
LIZZIE
OK
■
```

No questions. Read on!

6. Try it yourself. Since we don't know your name, try it with Lizzie's name.

You press: **CLEAR**


You type: PRINT "LIZZIE"

You press: **ENTER**

The screen should look like this:

```
PRINT "LIZZIE"  
LIZZIE  
OK  
■
```

If the screen doesn't look as above, try again. Be careful—spell PRINT correctly and *do* remember to enclose LIZZIE in quotation marks.

To type a quotation mark, hold down the **SHIFT** key and press the  key.

Now pretend that you are the computer and show what the screen will look like after the PRINT statement shown below has been typed.

```
PRINT "TAKE A DRAGON TO LUNCH"  
_____  
_____  
_____
```

```
TAKE A DRAGON TO LUNCH  
OK  
■
```

7. The statement: PRINT "TAKE A DRAGON TO LUNCH" is a *direct* PRINT statement. It tells the computer to print something on the screen. The computer prints whatever is enclosed in quotation marks following the word PRINT.

PRINT " TAKE A DRAGON TO LUNCH "

This is a *string*.

In a PRINT statement, a string is enclosed in quotation marks. A string is any bunch of keyboard characters, typed one after another.

A string can be a name: KARL

A string can be a telephone number: 415-323-6117

A string can be a message: TAKE A DRAGON TO LUNCH

A string can be gibberish: AB#J%FD+Z

A string can be almost anything you can type on the keyboard.

Since quotation marks (") mark the beginning and end of a string in a PRINT statement, do you think quotation marks can be part of a string?—

No, they cannot. (That would *really* confuse the computer!) However, single quotes (') can be used in a string. For example:

You type: PRINT "THEY SAID, 'ALL RIGHT.' "
 It prints: THEY SAID, 'ALL RIGHT.'
 OK

8. Complete the PRINT statement so that the computer prints as shown.

```
PRINT _____
I'M A TV STAR!
OK
■
```

```
PRINT "I'M A TV STAR!"
```

You did this (we hope), including the quotation marks.

9. Turn off the computer, then turn it back on again.

```
COLOR BASIC 1.0
(C) 1980 TANDY
OK
■
```

Now hold down the **SHIFT** key and press the zero key **0**. Then let go of the **SHIFT** key.

Type a bunch of letters. They will be green on a black background. This is called *reverse color*.

Press the **ENTER** key. You will probably get a ?SN ERROR. This will happen even if you type something the computer knows, such as PRINT.

Here is what happened when we (1) turned on the computer, (2) pressed **SHIFT** and **0** together, (3) typed PRINT "LIZZIE", and (4) pressed the **ENTER** key.

```
COLOR BASIC 1.0
(C) 1980 TANDY
OK
PRINT "LIZZIE"
?SN ERROR
OK
■
```

Green letters on
a black background.

To get back to normal color, hold down the **SHIFT** key and press the **0** key.

Remember:

To go from normal color to reverse color, hold down the **SHIFT** key and press **0**.

To go from reverse color to normal color, hold down the **SHIFT** key and press **0**.


10. Have you made a typing mistake yet? If you do, BASIC provides a simple way to fix it. Watch while we make a typing error.


```
We type: PTINT "OLAF"
It prints: ?SN ERROR
OK
```



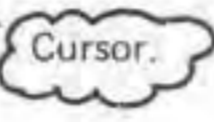









Huh? What's a PTINT?

We misspelled PRINT, so the computer doesn't know what we want. If we had noticed that we hit T when we meant to hit R, we could have corrected our mistake by using the backspace key.



The backspace key: 

Each time you press the backspace key () , the cursor moves back (to the left) one space and erases the character in that space.

DO THIS	THIS IS WHAT YOU SEE
Press CLEAR	
Type ABCDE	ABCDE  
Press 	ABCD 
Press 	ABC 
Press 	AB 

What will happen if you press  two more times? _____


The cursor will move two places to the left, erasing A and B. Use the backspace key at any time to erase an incorrect character, then type the correct character.

Remember: To erase the character to the left of the cursor, press  once. To erase two characters, press  twice. And so on.

11. The backspace key is great for erasing errors you have just made. But suppose you are typing a long line and are almost to the end when, alas, out of the left corner of your left eye, you spot a mistake way back at the beginning. If you complete the line and press **ENTER**, you will get a ?SN ERROR. You would like to just scrub the line, erase it entirely, and start over.

You can! Hold down the **SHIFT** key and press the  key.

You type: PTINT "ONCE UPON A TIME TH

Oops! PRINT is misspelled. Hold down the **SHIFT** key and press the  key. Poof! The line disappears. Start over. Type it again, correctly.

12. You know how to clear the screen by pressing the **CLEAR** key. Here is another way; you can use it whenever the cursor is flashing.

You type: **C** **L** **S** and press the **ENTER** key.

This is what you see:



CLS is a direct statement. It tells the computer to *C*lear the *S*creen.

- To clear the screen, press the **CLEAR** key.
- To clear the screen, type CLS and press the **ENTER** key.

What is different in the appearance of the screen? _____

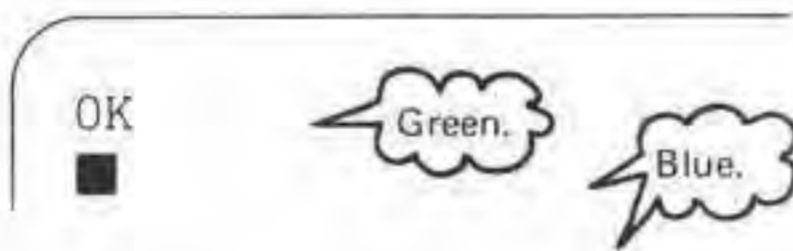
Pressing **CLEAR** leaves only the cursor on the screen. After typing CLS, the word OK and the cursor are on the screen.

Hmmmm . . . it seems that OK always appears after the computer obeys a direct statement.

13. Tired of a green screen? Try other colors.

You type: CLS 3 and press **ENTER**.

The screen is now mostly blue, except for a green line at the top.



The actual color, of course, will depend on your TV set. Fiddle with the color settings until you get a blue screen with the top line green.

Another color? Of course.

You type: CLS 8 and press **ENTER**.

This time you get an orange screen, except for the top line, which is green. What? You got a dirty brown? Well, fiddle with the color controls. CLS 8 is supposed to give an orange screen.

Try other colors. Use numbers from 0 to 8. Then complete the following screen color chart.

<u>Statement Color</u>	<u>Statement Color</u>
CLS 0 _____	CLS 5 _____
CLS 1 _____	CLS 6 _____
CLS 2 _____	CLS 7 _____
CLS 3 _____	CLS 8 _____
CLS 4 _____	

You are supposed to get these colors:

CLS 0 Black	CLS 5 Buff
CLS 1 Green	CLS 6 Cyan (a light blue)
CLS 2 Yellow	CLS 7 Magenta
CLS 3 Blue	CLS 8 Orange
CLS 4 Red	

Actual colors will depend on your TV and how you set its color controls. Set them any way you want and make up your own color chart!

14. What would happen if:

- You type CLS 9 and press **ENTER**?
- You type CLS 3.14 and press **ENTER**?
- You type CLS(3) and press **ENTER**?
- You type CLS(-1) and press **ENTER**?
- You type CLS 7 and press **ENTER**?
- You type CLS 4 and press **ENTER**?

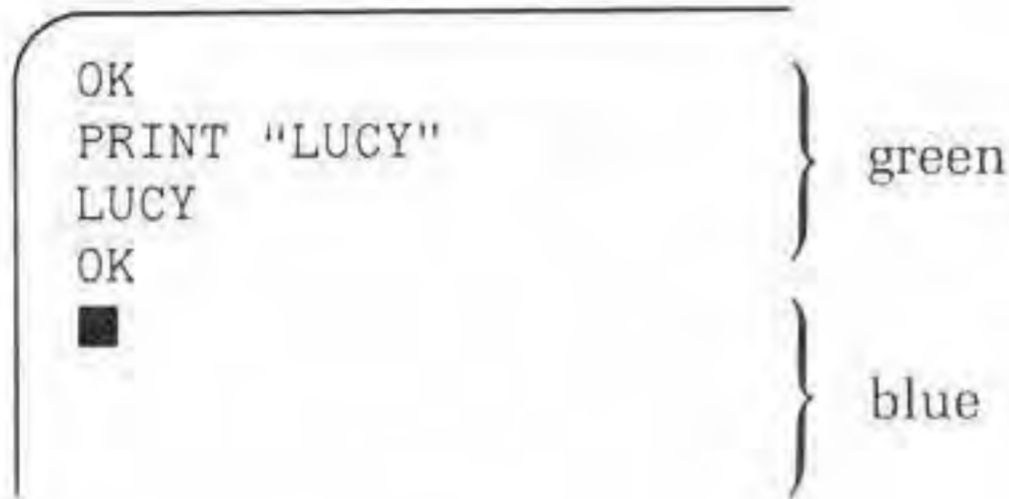
Several spaces.

Try it and find out. Experiment!

15. Can you put a name on a blue screen?

You type: CLS 3
You type: PRINT "LUCY"

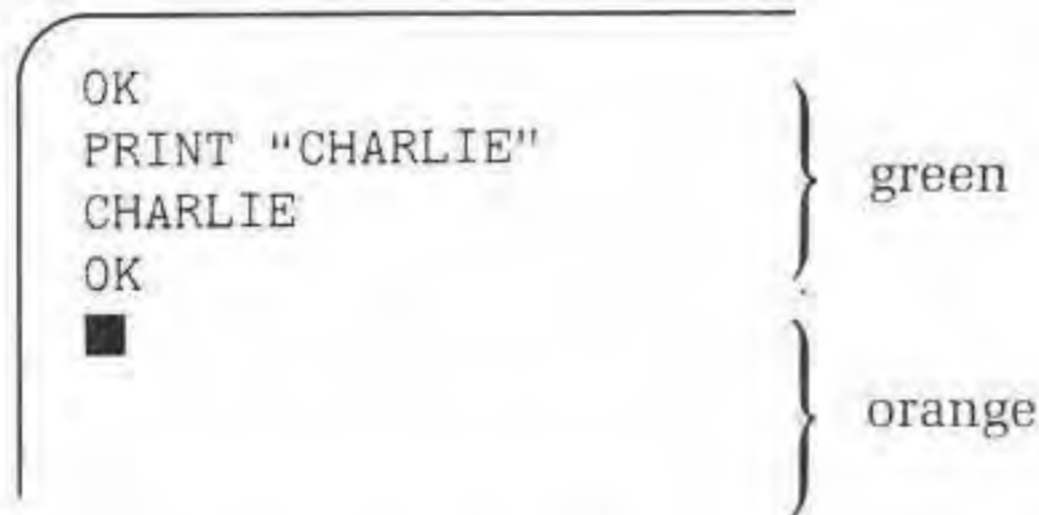
And this is what you see:



Next, try orange.

You type: CLS 8
You type: PRINT "CHARLIE"

This is what you see:



Can you put a name on a blue or orange background? _____

No. Whenever the Color Computer prints something, it prints black characters on a green background. Well, if you are in reverse color, you get green characters on a black background (see frame 9). If this happens, hold down the **SHIFT** key and press **0** to get back to normal mode.



16. For a slightly different effect, try this:

You type: CLS 3 : PRINT "LUCY"



This is what you see:

```
LUCY
OK
```

Now type: CLS 8 : PRINT "CHARLIE" And you will get:

```
CHARLIE
OK
```

What would you type to make the screen look like this?

```
SNOOPY
OK
```

You type: _____

```
CLS 4 : PRINT "SNOOPY"
```

IMPORTANT NOTICE: The computer ignores spaces, except inside quotation marks. Try these:

```
CLS 4:PRINT "SNOOPY"
CLS4:PRINT "SNOOPY"
CLS 4 : PRINT " SNOOPY"
```



17. In the last frame, we put two statements on a single line.

CL S 4 : PRINT "LUCY"
1st statement 2nd statement

CL S 8 : PRINT "CHARLIE"
1st statement 2nd statement

CL S 4 : PRINT "SNOOPY"
1st statement 2nd statement

When you put two statements on a single line, what do you put *between* the two statements? _____

a colon (:)

Are you wondering if you can put more than two statements on a line?
EXPERIMENT!

18. Til now, your Color Computer adventure has been colorful but quiet. Add some musical accompaniment.

You type: SOUND 89, 20

Did you hear it? If not, turn up the volume on your TV and do it again. When you type SOUND 89, 20 and press **ENTER**, the Color Computer plays a tone on the sound system of the TV.

Now turn the volume to a comfortable level, tune in your ears, and try some tones.

You type: SOUND 147, 20

(a) A different tone! What's different? _____

You type: SOUND 89, 50

(b) How is this different from SOUND 89, 20? _____

-
- (a) The first number following SOUND is different, and a different tone is heard. The tone is higher than the one you heard before.
- (b) The second number following SOUND is different. This time the tone is heard for a longer time.

In a SOUND statement, the computer ignores spaces. It accepts all of the following as the same:

SOUND89,20 SOUND 89, 20 SOUND 89.20 SOUND 89 , 20

↑

We prefer this one. It is the easiest for *people* to read.

19. Aha! The first number tells the computer what tone to play; different numbers give different tones. The second number is the length, or *duration*, of the tone; bigger numbers give longer tones.

SOUND 89, 20

↑ ↑

Play this tone for this long.

TONE DURATION

A tone can be very low.

SOUND 1, 20



A tone can be quite high.

SOUND 255, 20



Guess! What is the lowest tone number you can use in a SOUND statement? _____ The highest? _____

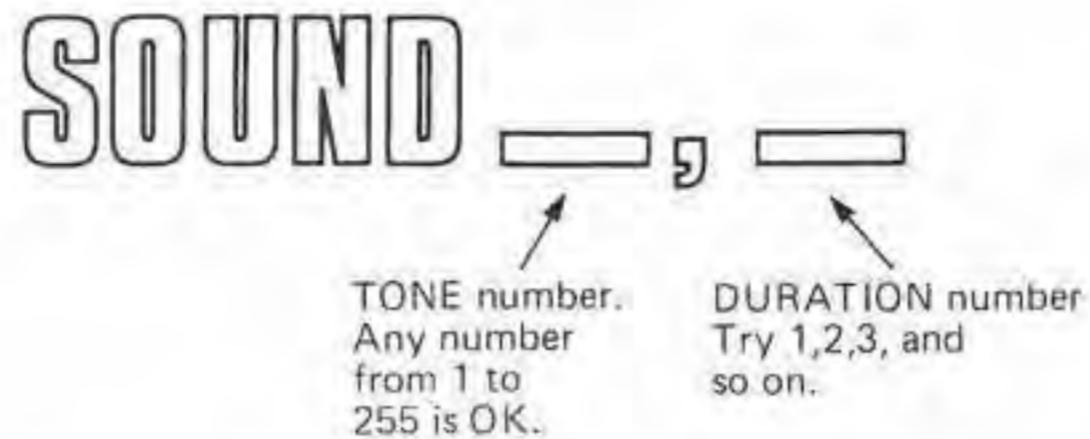
Lowest: 1
Highest: 255

Hmmm . . . wonder what note on the musical scale tone number 89 is? Here is a hint.



If you don't understand the hint, send a self-addressed stamped envelope to: BAD HINT, P.O. Box 310, Menlo Park, CA 94025. (We collect self-addressed stamped envelopes.)

20. OK, trundle over to your Color Computer and make some sounds. Make low sounds, high sounds, in-between sounds. Make little quick sounds, longer sounds, and sounds that make everyone yell, "Turn that thing off!"



Hmmm . . . what would happen if? Try some of these.

SOUND 0, 0	SOUND 0, 20
SOUND 89, 0	SOUND 256, 20
SOUND 3.14, 20	SOUND -1, 20
SOUND 89, -1	SOUND 89, 1000000
SOUND 89, 256	SOUND 89, 255

A tone number may be any number from 1 to 255. What numbers may be used for a duration number (go ahead—guess!)? From _____ to _____.

1; 255 (From 1 to 255)

See Appendix G for more information on SOUND.

21. Your turn. Write some SOUND statements.
- (a) Write a SOUND statement using tone number 58 and duration number 50. _____
 - (b) Write a SOUND statement using tone number 133 and duration number 10. _____
 - (c) Write a SOUND statement using tone number 176 and duration number 25. _____
 - (d) Write a SOUND statement using tone number 218 and duration number 37. _____

- | | |
|-------------------|-------------------|
| (a) SOUND 58, 50 | (b) SOUND 133, 10 |
| (c) SOUND 176, 25 | (d) SOUND 218, 37 |

Don't write a SOUND statement using tone number 256 or duration number -37. Why not? You will get an FC ERROR. Oh? You like FC ERRORS?! OK, do write a SOUND statement using. . . .

22. Combine color and sound.

You type: CLS 3 : SOUND 89, 100



For several seconds, the screen is blue and tone number 89 sounds. Then the computer stops with:

```

OK      } green
■       } blue
    
```

Tenors type: CLS 8: SOUND 204, 100

Baritones type: CLS 1: SOUND 5, 100

Sorry, basses. The Color Computer doesn't get down to your best tones. Use a stopwatch, or the sweep second on a clock, or simply count (1001, 1002, etc.) to find out how long the tone and blue screen are on. How long? _____

About 6 seconds. A duration number of one (1) gives a tone about $\frac{1}{100} = .06$ seconds long. Therefore a duration number of 100 will give a tone about $100 \times .06 = 6$ seconds long.

23. Combine SOUND and SOUND.

You type: SOUND 89, 20 : SOUND 176, 30

1st sound
2nd sound

Colon.



Combine PRINT and SOUND.

Press the **CLEAR** key

You type: PRINT "HMMMMMMMMMM" : SOUND 89,20

Combine color and PRINT and SOUND.

You type: CLS 8 : PRINT "LA LA LA" : SOUND 159,30

How about a rousing do re mi? Turn up the volume and . . .

You type: SOUND 89,10 : SOUND 108, 10 : SOUND 125,10

do
re
mi

Well, the obvious thing to fa lo is
SOUND 133,10: SOUND 147,10



Oh well, if you didn't understand that pun, send a self-addressed stamped envelope to Bad Pun, P.O. Box 310, Menlo Park CA 94025 and our captive . . . er, resident . . . musician will explain.

While you are waiting for a reply, increase your confidence by waltzing through our Chapter 2 Self-Test. You may also wish to browse a bit in Appendix G, which has a less frivolous approach to the SOUND statement.

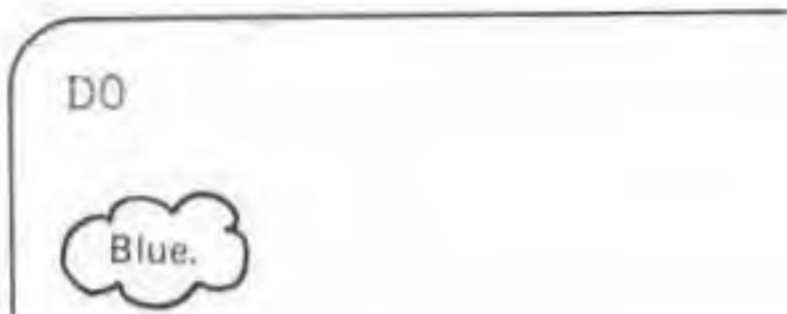
Self-Test

Before zooming on to Chapter 3, dally for awhile with this self-help Self-Test.

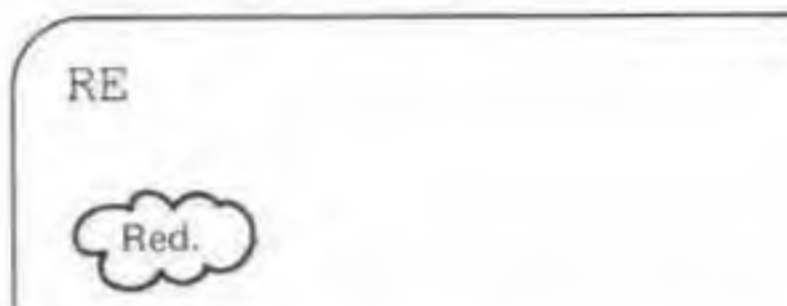
1. When you first turn on the TRS-80 Color Computer, you see a few words on the screen and a rectangular blob that blinks and changes color. What is the rectangular blob called?_____
2. The quickest way to clear the screen is to press the **CLEAR** key. After you press the **CLEAR** key, what does the screen look like?_____
3. You can also clear the screen by using the CLS statement as a *direct*, or *immediate*, statement. How?_____
4. After you use CLS to clear the screen, what does the screen look like?_____
5. If you type: DO THE HOMEWORK ON PAGE 157
The computer will type: _____
6. If you type: PRINT "DO THE HOMEWORK ON PAGE 157"
It will type:_____
7. To make the screen blue (well, mostly blue), what would you type?_____

8. You can tell the computer to make SOUNDS which vary from a robust, moderately low tone to a fragile, tinkly high tone. Complete the following SOUND statements for these two *extremes*.
(a) ROBUST, MODERATELY LOW: SOUND____, 20
(b) FRAGILE, TINKLY HIGH SOUND____, 20
9. A Color Computer SOUND can be very, very short or quite long. Complete the following SOUND statements so that good old tone 89 is as short as it can be or as long as it can be.
(a) Shortest: SOUND 89,____
(b) Longest: SOUND 89,____

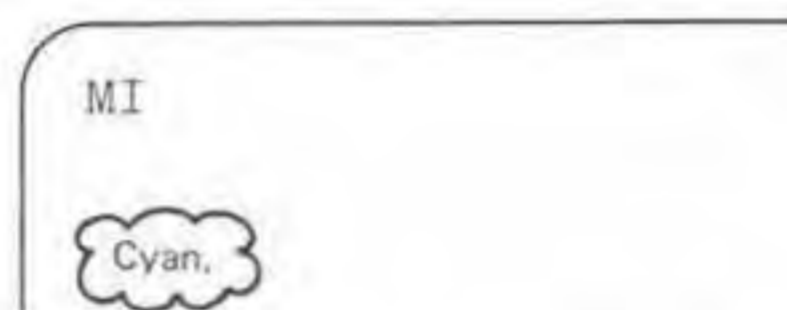
10. Hang on! This is a big one. So take three deep breaths—relax; you can do it!
1st deep breath
2nd deep breath
3rd deep breath
Write one *multiple* statement (with statements separated by colons, of course) so that, after you press **ENTER**, this will happen:

(1) The diagram shows a rectangular screen with a top-left corner cut off. Inside the screen, the text "DO" is positioned at the top left. Below it, there is a cloud-shaped icon containing the text "Blue."

The word DO appears, tone number 89 sounds for a little while, screen is mostly blue.

(2) The diagram shows a rectangular screen with a top-left corner cut off. Inside the screen, the text "RE" is positioned at the top left. Below it, there is a cloud-shaped icon containing the text "Red."

The word RE appears, tone number 108 sounds for a little while, screen is mostly red.

(3) The diagram shows a rectangular screen with a top-left corner cut off. Inside the screen, the text "MI" is positioned at the top left. Below it, there is a cloud-shaped icon containing the text "Cyan."

The word MI appears, tone number 125 sounds for a little while, screen is mostly cyan.

11. EXPERIMENT! To learn more about ways of PRINTing stuff on the screen, try these. Avoid clutter—press **CLEAR** before you type each PRINT statement.
- (a) PRINT 1, 2
 - (b) PRINT 1; 2
 - (c) PRINT 1, 2, 3
 - (d) PRINT 1; 2; 3
 - (e) PRINT 1, 22, 333, 4444, 55555
 - (f) PRINT 1; 22; 333; 4444; 55555
 - (g) PRINT "1", "2"
 - (h) PRINT "1"; "2"
 - (i) PRINT "1", "2", "3"
 - (j) PRINT "1"; "2"; "3"
 - (k) PRINT 1 + 2
 - (l) PRINT "1" + "2"
 - (m) PRINT 1 + 2 + 3
 - (n) PRINT "1" + "2" + "3"
 - (o) PRINT "1" "2" "3"
 - (p) PRINT "1""2""3"
 - (q) PRINT "GREEN", "SLEEVES"
 - (r) PRINT "GREEN"; "SLEEVES"
 - (s) PRINT "GREEN" + "SLEEVES"
 - (t) PRINT "GREEN" "SLEEVES"
 - (u) PRINT "SLEEVES" " OF " "GREEN"

We hope the above suggests more experiments. Design your own experiments, then try them. Write down what happened. Write yourself a book on how to use the Color Computer.

Answers to Self-Test

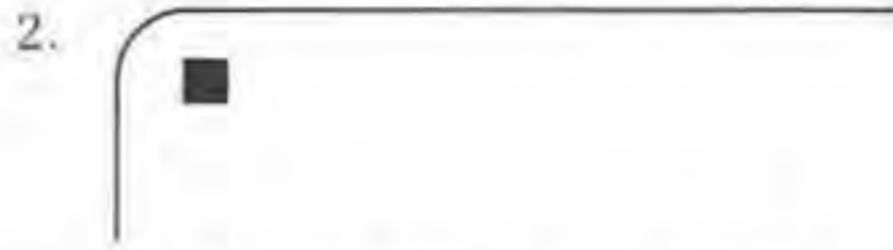
The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed. Here are our answers. Sometimes, our answer is the "right" answer; other times, our answer is, well . . . *our answer* . . . an answer that works.

As you go through this book, you will find that there is rarely a "right" answer. Increasingly, as you go from chapter to chapter, you will find that each Self-Test question has many answers. Ours, yours, others.

As you gain confidence, you will more and more often find that *your* answer is better for you than our answer. *That is the Joy of Computing!*

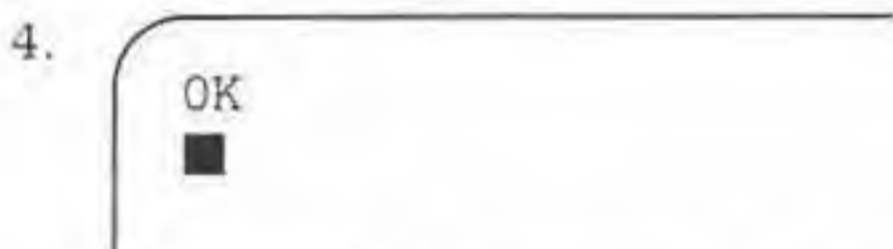
And so we give you our answers to the questions in Chapter 2 Self-Test. (Frame numbers are shown for each answer.)

1. The CURSOR. Whenever you see the cursor patiently blinking and winking at you, you know it is *your* turn to do something. (frames 1–3)



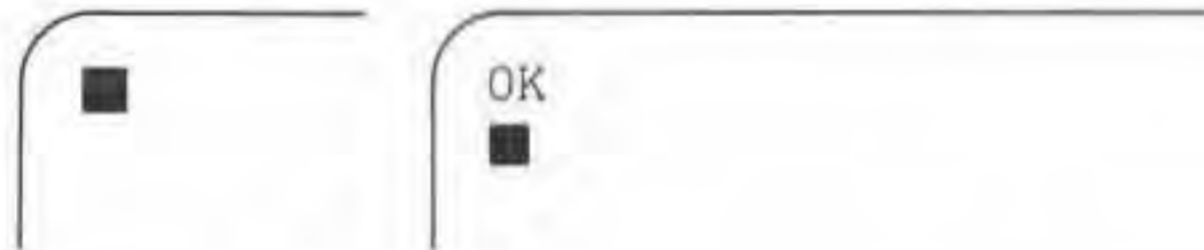
The entire screen is green, except for the cursor, blinking and changing color in the upper left corner of the screen. (frame 3)

3. Type `C` `L` `S` and press the **ENTER** key. (frame 12)



The word OK appears in the upper left corner of the screen. The winking, blinking cursor appears immediately below OK. (frame 12)

Press **CLEAR** Type `CLS` and press **ENTER**



5. ?SN ERROR
Shame on you! Trying to get the computer to do your homework! Even a computer knows it is a SiN to do homework for a human. Computers know that HOMEWORK (such as vacuuming, mopping floors, cleaning one's room, washing dishes, and so on) is for *humans*, not computers. (frame 4)
 6. DO THE HOMEWORK ON PAGE 157
Ha! Of course! The computer tells *you* to do the homework . . . because you told it to tell you to do homework . . . oh me, oh my . . . if the computer doesn't do the homework, and *you* don't do the homework . . . we are really sorry we included this question—please forget about it and go on to the next question. (frames 5–7)
-

7. We tried: CLS BLUE
Alas, that didn't work.
Voila! We tried: CLS BLEU
Alors . . . that didn't work either.
Undaunted, we approached our keyboard and struck these keys: CLS BLEW.
To no avail! The computer printed OK on a green stripe on a mostly black screen. And we thought we could faintly hear a message from the computer: You BLEW it!
Then we remembered. To tell the computer to make a mostly blue screen,
Type: CLS 3 and press **ENTER**
We did. It did. (frame 13)

8. (a) SOUND 1,20
(b) SOUND 255,20
(frames 18, 19)



9. (a) SOUND 89,1
(b) SOUND 89, 255 (frame 20)
- 10, 11. Sorry, no answers. We will leave one or more questions in each chapter for you to answer. For hints and/or answers to similar questions, or for lots more computing problems and solutions, read issues of the following periodicals:

The Computing Teacher,
Department of Computer and Information Science
University of Oregon
Eugene, Or. 97403
\$14.50 a year (9 issues)

Dymax Gazette,
P.O. Box 310,
Menlo Park, CA 94025
\$12 a year (6 issues)

Popular Computing,
P.O. Box 397
Hancock, N.H. 03449
\$15 a year (12 issues)



CHAPTER THREE

BASIC Programs

In this chapter, you will learn to understand and use simple BASIC programs. These programs will include statements that you already know how to use (PRINT, CLS, SOUND) and two new statements (GOTO and REMARK).

When you finish this chapter, you will be able to:

- Read and understand short programs that include PRINT, CLS, SOUND, GOTO, and REMARK statements;
- Use NEW to erase an old, unwanted program from the computer's memory;
- Enter a new program into the computer's memory;
- Tell the computer to LIST a program on the TV screen;
- Tell the computer to RUN (execute) a program in its memory;
- Write simple programs using PRINT, SOUND, CLS, GOTO and REMARK statements;
- Edit, correct, or delete statements in a program.

As usual, EXPERIMENT as you boogie through this chapter.

1. Now you will learn how to *enter a program* into the Color Computer's memory. The following program causes the computer to put Karl's name on every line of the screen.

```
10 CLS
20 PRINT "KARL"
30 SOUND 89, 10
40 GOTO 20
```



The above program consists of four *statements*.

This is a statement:	10 CLS
This is a statement:	20 PRINT "KARL"
This is a statement:	30 SOUND 89, 10
This is a statement:	40 GOTO 20

- (a) Which statements did you already use in Chapter 2? CLS, PRINT, and _____.
- (b) Which statement is brand new? _____
-

- (a) SOUND
- (b) GOTO or GOTO 20 (GOTO can also be written as two words: GO TO)

In the next few frames, we will describe the program, show you how to enter it into the memory, and show you how to tell the computer to RUN, or execute (obey, carry out), the program.

2. Each statement in the program begins with a *line number*.

Line number →	10 CLS
Line number →	20 PRINT "KARL"
Line number →	30 SOUND 89, 10
Line number →	40 GOTO 20

When you type a statement that begins with a line number, the statement is *not* executed immediately after you press the **ENTER** key (as in Chapter 2). Instead, the statement is stored in the computer's memory for later execution.

A statement that does not begin with a line number is called a *direct statement* or *immediate statement*. You learned about direct statements in Chapter 2.

- (a) What happens when you type a direct statement (without a line number) and press **ENTER**? _____

- (b) What happens when you type a statement that begins with a line number, then press **ENTER**? _____
-

(a) The computer executes the statement immediately, then forgets the statement. Statements without line numbers do not remain stored for later execution.

(b) The statement is stored in the computer's memory for later execution. That is, the computer "remembers" the statement.

3. Line numbers tell the computer the order in which to follow statements in the program. Line numbers don't have to be consecutive integers such as 1, 2, 3, 4, 5, and so on. Instead, it is better to number by tens as we do in the following program. Then, if you wish, you can easily insert or add more statements between ones you already have.

```
10 CLS
20 PRINT "KARL"
30 SOUND 89,10
40 GOTO 20
```

How many additional statements can you add *between* line 20 and line 30?

Nine statements (lines 21, 22, 23, 24, 25, 26, 27, 28, and 29)


Of course, you don't *have* to number by tens. If you prefer numbering by thirteens or fives or jumping around, do it!

4. Before you enter a program, you must first remove or erase any programs that may already be stored in the memory (in RAM, that is). If you don't do this, the new program may become intertwined with an old program, resulting in . . . confusion!

Here is how to erase old programs and get the Color Computer ready to accept a new program:

You type: NEW and press the **ENTER** key.

The screen should
look like this:



```
NEW
OK
■
```

The computer has erased the portion of its memory that stores BASIC programs. It is ready to accept a new program.

How do you erase, remove, or delete an old program from the computer's memory? _____

Type NEW and press the **ENTER** key. If you misspell NEW, you may get an error message. For example:

We type: GNU
It prints: ?SN ERROR

Apparently the computer doesn't appreciate puns.



5. Now you are ready to enter the four-line program shown in frames 1 and 3. Here it is again.

```
10 CLS  
20 PRINT "KARL"  
30 SOUND 89,10  
40 GOTO 20
```


Before you enter the program, you must first erase any old program by typing and pressing the **ENTER** key. Then enter the program, one line at a time. (a) You type: 10 CLS and press **ENTER**.

(b) You type: 20 PRINT "KARL" and press _____.

(c) You type: _____.

(d) You type: _____.

(a) NEW

(b) **ENTER**

(c) 30 SOUND 89, 10, and press **ENTER**

(d) 40 GOTO 20 and press **ENTER**

6. Do it! Enter the program. First, clear the screen by pressing the **CLEAR** key. Then:

You type: NEW


It prints: OK



Type each line of the program. After typing each line, press the **ENTER** key.

You type:

```
10 CLS
20 PRINT "KARL"
30 SOUND 89, 10
40 GOTO 20
```

If you make a typing error, correct it by using the  key. If you have already pressed **ENTER**, simply retype the line correctly. The program is now stored in the computer. The computer patiently awaits your next instruction. No questions. Scurry on to the next frame.

7. Hmmmm . . . is the program *really* stored in memory? Find out. First, clear the screen by pressing the **CLEAR** key. Then type LIST and press **ENTER**.

You type: LIST and press **ENTER**.

The screen should look like this:

```
LIST
10 CLS
20 PRINT "KARL"
30 SOUND 89, 10
40 GOTO 20
OK
■
```

LIST tells the computer to print on the screen all of the statements in the program stored in the computer's memory. After listing the program, the computer prints OK and turns on the cursor to let you know it is your turn again.

How do you tell the computer to print the program that is stored in its memory?_____

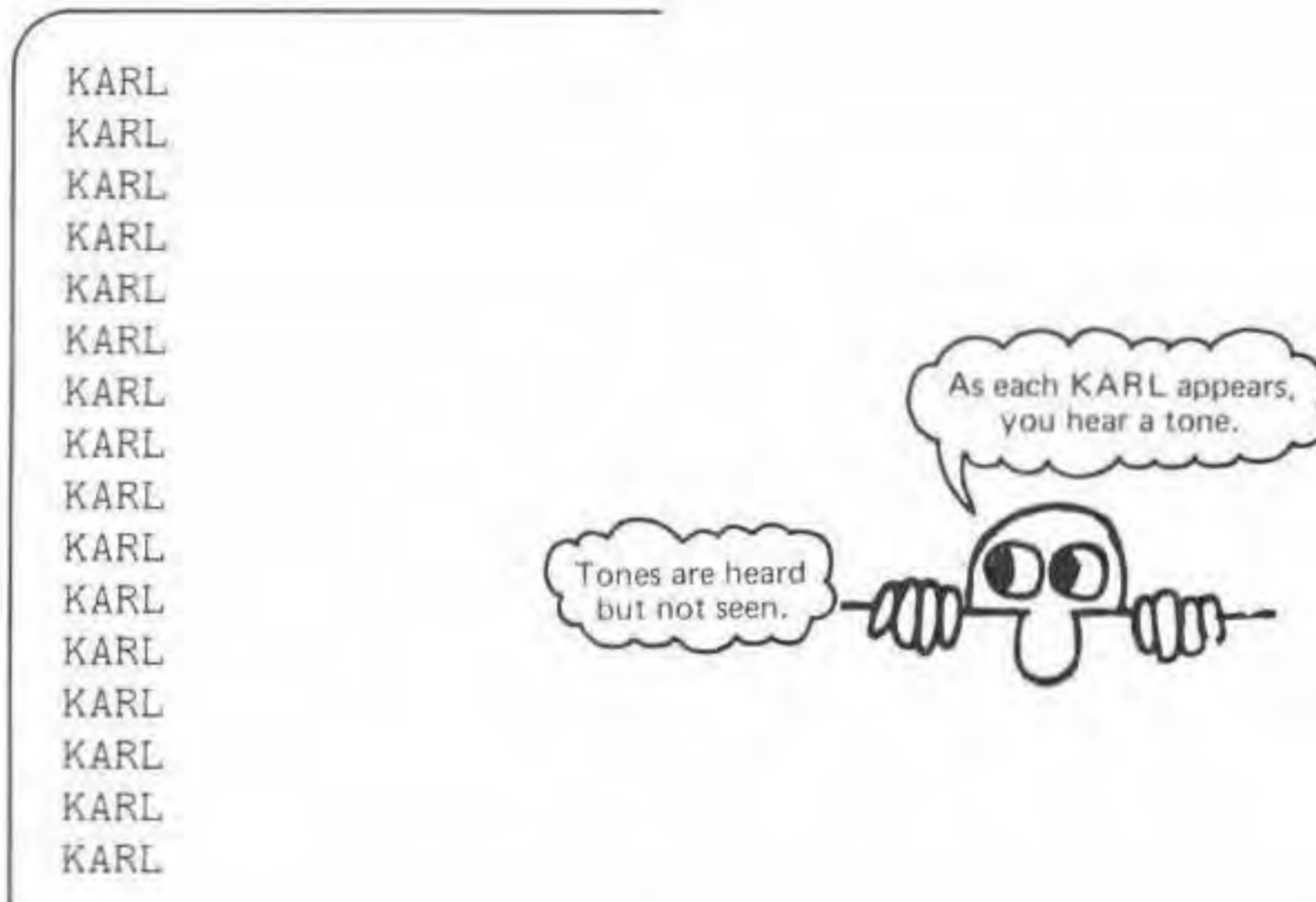
Type LIST and press the **ENTER** key.

The computer then prints a copy of the program on the TV screen. If there are mistakes in the program, correct them by retyping any line (10, 20, 30, 40) that has a mistake. Then LIST the program again. Repeat until all is well!

8. Finally! The program is stored in memory (RAM, of course) and ready to go. It's time to RUN the program.

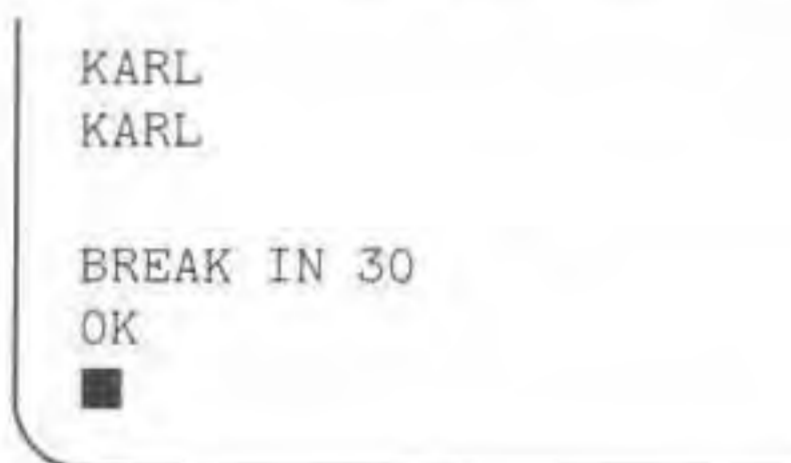
You type: RUN and press the **ENTER** key.

The screen is momentarily clear, then KARL appears on the top line, then on the second line, then on the third line, and so on until the screen looks like this:



Nothing more seems to happen except that the tone keeps sounding (and soon gets mono-tone-ous, we think). Actually, the computer keeps printing KARL on the bottom line. Each time it does, all the other KARLs are pushed up one place; the top KARL is "pushed off" the top of the screen. This happens so quickly, however, that only superheroes with ultrafast eyes can see it happen.

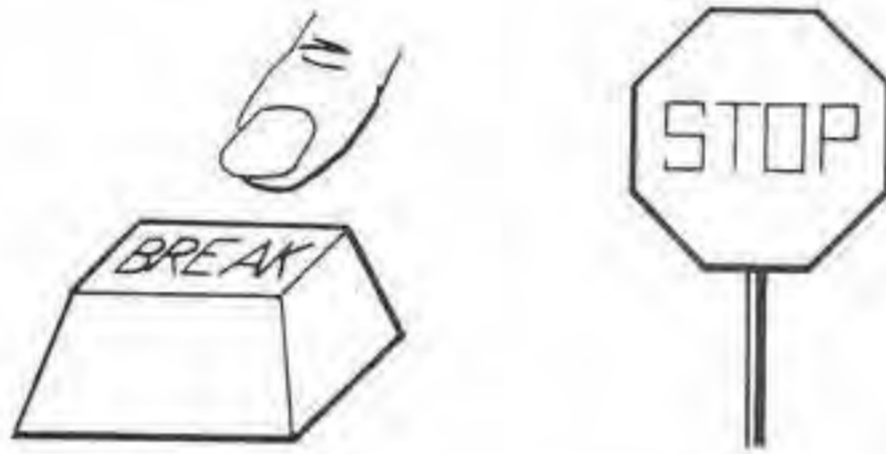
How do you stop the computer? Easy. Find the **BREAK** key. Press it. Congratulations! You have stopped the computer. The bottom part of the screen probably looks like this:



How do you stop a runaway computer?

- _____ (a) Turn off the computer (press the **ON-OFF** button).
 - _____ (b) Pull the power cord out of the electrical outlet.
 - _____ (c) Yell **STOP!** in a loud voice three times.
 - _____ (d) Press the **BREAK** key.
-

(a), (b), and (d) will work. However, (a) and (b) cause the computer to erase its memory, so the program would be erased. The best choice is (d)—to stop the computer, press the **BREAK** key.



9. Well, maybe the program wasn't stored correctly. Maybe **PRINT** was misspelled, as shown below.

```
10 CLS
20 PTINT "KARL"
30 SOUND 89,10
40 GOTO 20
```



Alas, the error went unnoticed. You typed **RUN** and . . .

```
?SN ERROR IN 20
OK
```



Error? Where? Oh, in line 20. Might as well take a look at it. Clear the screen, then:

You type: LIST and press **ENTER**

```
LIST
10 CLS
20 PTINT "KARL"
30 SOUND 89,10
40 GOTO 20
OK
■
```



How can you correct the error? _____

Retype line 20 correctly and press **ENTER**. That is, type *20 PRINT "KARL"* and press **ENTER**. Then LIST the program to see if everything is OK. If so, RUN the program.

NOTE: When you enter a program, errors are not detected by the computer until you tell the computer to RUN the program. During a RUN, if the computer tries to execute a statement with an error (such as line 20, above), it will stop and print an error message.

10. How does the program work? The statement:

```
10 CLS
```

tells the computer to clear the screen. The statement:

```
20 PRINT "KARL"
```

tells the computer to print the string KARL on the screen. The statement:

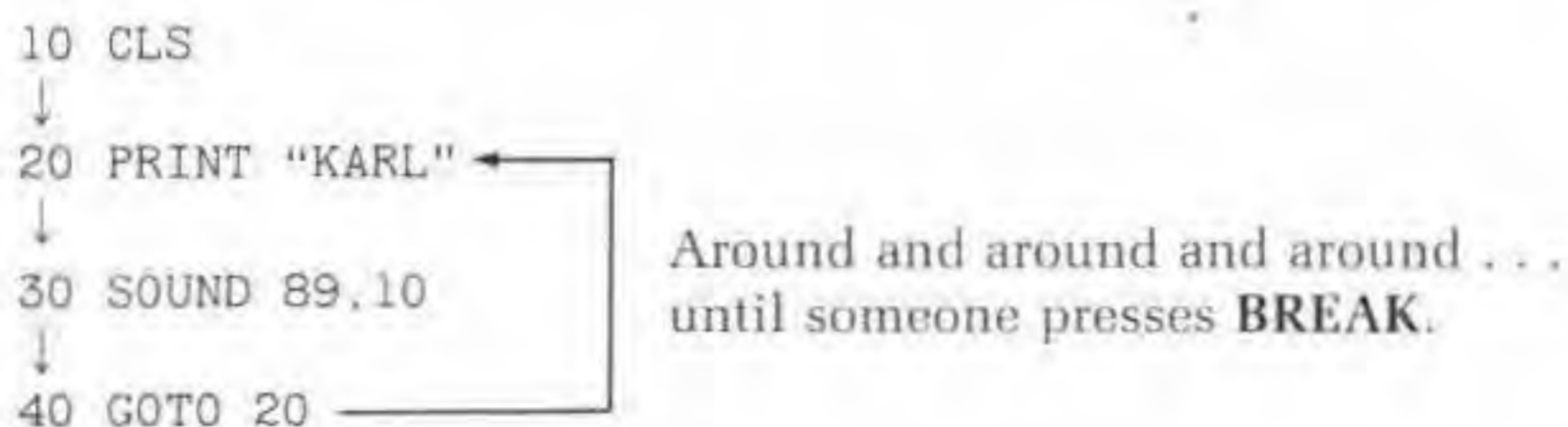
```
30 SOUND 89,10
```

tells the computer to play tone number 89 for a short time. The statement:

```
40 GOTO 20
```

tells the computer to go to line 20 and continue.

When you type RUN and press **ENTER**, the computer begins executing the program, *beginning with the smallest line number*. The computer does line 10, then line 20, then line 30, then line 40, then line 20, then line 30, then line 40, then line 20, and so on . . . until someone presses **BREAK**. Follow the arrows.



No questions. Hurry on to the next frame, or EXPERIMENT!

Change line 40 to: 40 GOTO 10
 or 40 GOTO 30
 or 40 GOTO 40

Change line 30 to: 30 SOUND 133,1
 or 30 SOUND 176,20

Change line 20 to: 20 PRINT "TRA LA LA"
 or 20 PRINT "OM"
 or 20 PRINT "TO STOP ME, PRESS BREAK"

11. We assume that the program of frames 1–9 is still in the computer's memory. If not, please enter it as shown in frame 6. Then clear the screen and LIST the program.

```

LIST
10 CLS
20 PRINT "KARL"
30 SOUND 89, 10
40 GOTO 20
OK
■

```

Now change line 20, but only line 20. Do not (repeat, do not) type NEW. Instead, do this:

You type: 20 PRINT "KARL"; (semicolon at end)

You have changed line 20. Clear the screen and LIST the modified program.

```
LIST
10 CLS                               Old line 10.
20 PRINT "KARL";                     New line 20.
30 SOUND 89, 10                      Old line 30.
40 GOTO 20                            Old line 40.
OK
■
```

The new line 20, with a semicolon on the right end, replaces the old line 20, which did not have a semicolon. What does the semicolon do? As usual with computers, the best way to find out is to experiment—try it and see what happens.

So type RUN, press **ENTER**, and watch.

```
KARLKARLKARLKARLKARLKARLKARL
KARLKARLKARLKARLKARLKARLKARL
KARLKARLKARLKARLKARLKARLKARL
KARLKARLKARL
```

and so on.

Each KARL is accompanied by the sound of tone 89.



What does the semicolon on the right end of the PRINT statement do? (Go ahead, guess!)

Causes the computer to print across the screen from left to right. When it gets to the right edge of the screen, it continues on the left side of the next line down. Without the semicolon, the computer prints whatever is between quotation marks in the PRINT statement, then moves immediately to the left side of the next line. More about this later in the book.

12. Change line 30, as follows:

30 SOUND 89, 1 (a very short sound)

Then try these variations for line 20. For each new line 20, RUN the program to see what happens.

20 PRINT "MARY ";	Put one space after MARY, inside the quotation marks.
20 PRINT "JUDY ";	Put two spaces after JUDY, inside the quotation marks.
20 PRINT "'..";	Huh? Put an apostrophe (<input type="text" value="7"/>) and two periods between quotation marks.
	<input type="text" value="7"/> <input type="text" value="."/> <input type="text" value="."/> ← Hold down the SHIFT key to get the apostrophe.
20 PRINT ">";	Put one space after <input "="" type="text" value=" "/> <input type="text" value=">"/>

Illusion: That's what is happening. Things seem to move on the screen. The computer is simply printing what is between quotation marks (a string—remember?) across the screen, beginning at the top left. Watch the bottom line of the screen. When the screen fills, the computer keeps right on printing new lines across the bottom line. Each new line pushes all the old lines up one place (computer people call it "scrolling").

So . . . play! Put anything you want in quotation marks in line 20. Try various numbers of spaces. Enjoy!

20 PRINT "  ;

13. What would happen if you changed line 10 to one of the following? Try it and find out. Experiment!

```
10 CLS 0
10 CLS 1
10 CLS 2
10 CLS 3
10 CLS 4
10 CLS 5
10 CLS 6
10 CLS 7
10 CLS 8
```

14. Why not be a TV star with a flair? Try your version of this program:

```

NEW
100 CLS
110 PRINT "INTERGALACTIC"
120 SOUND 89, 20
130 PRINT "BROADCASTING"
140 SOUND 125, 20
150 PRINT "COMPANY"
160 SOUND 147, 20
170 PRINT "BRINGS TO YOU . . ."
180 SOUND 176, 40
200 CLS
210 PRINT "LUCY ";
220 SOUND 89, 1
230 GOTO 210

```

Of course, you may want to put *your* name in line 210.

In the above program, we added some line spaces to make the program easier for you to read and also to group parts of the program that go together. For example, lines 110 and 120 work together; lines 130 and 140 work together; and so on. If you LIST the program, you will not see these extra line spaces. We will frequently include them in this book, however, to make programs easier for you to read and understand.

RUN the program. It goes like this:

```

(1) INTERGALACTIC
(2) INTERGALACTIC
    BROADCASTING
(3) INTERGALACTIC
    BROADCASTING
    COMPANY
(4) INTERGALACTIC
    BROADCASTING
    COMPANY
    BRINGS TO YOU . . .
(5) LUCY LUCY LUCY etc.

```


15. EXPERIMENT. Stop the program in frame 14 (press **BREAK**), then add these lines. Don't type NEW. Instead, just type these three lines:

```
125 PRINT
145 PRINT
165 PRINT
```

Now LIST the program. It should look like this:

```
LIST
100 CLS
110 PRINT "INTERGALACTIC"
120 SOUND 89, 20
125 PRINT
130 PRINT "BROADCASTING"
140 SOUND 125, 20
145 PRINT
150 PRINT "COMPANY"
160 SOUND 147, 20
165 PRINT
170 PRINT "BRINGS TO YOU . . ."
180 SOUND 176, 40
200 CLS
210 PRINT "LUCY ";
220 SOUND 89, 1
230 GOTO 210
```

The computer has inserted lines 125, 145, and 165 into the program so that all statements are in line-number order.

RUN the program. How does it differ from the original program?
What does an "empty" PRINT statement do? _____
This is an "empty" PRINT statement → 125 PRINT

An empty PRINT statement prints an empty line. In the above program, the empty PRINT statements in lines 125, 145, and 165 cause the information printed on the screen to appear "double-spaced," as follows:

```
INTERGALACTIC
BROADCASTING
COMPANY
BRINGS TO YOU . . .
```

16. Here is a new program for you to try:

```
10 CLS
20 SOUND 133, 10
30 SOUND 89, 10
40 GOTO 20
```

Enter this program. Remember: First type NEW to erase any old program. After you enter the program, LIST it and verify that it is correctly stored in memory.

RUN it. You will see an empty green screen and hear the two tones repeated again and again and again, and so on until you press **BREAK**. Hmmmmm . . . is it a fire truck, ambulance, or police car rushing to the scene?

Draw arrows showing the order in which statements are done by the computer.

```
10 CLS
20 SOUND 133, 10
30 SOUND 89, 10
40 GOTO 20
```

If you are not sure how to do this, see frame 10.

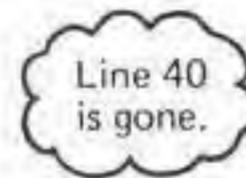
```
10 CLS
  ↓
20 SOUND 133, 10 ←
  ↓
30 SOUND 89, 10
  ↓
40 GOTO 20 ←
```

Around and around and around . . . until someone presses **BREAK**.

17. Stop the program (press **BREAK**) and remove *only* line 40 from the program. To do this, simply type the line number (40) and press **ENTER**.

To remove a line from the program, type only the line number and press **ENTER**.

You type: 40 and press **ENTER**
 You type: LIST and press **ENTER**
 It prints: 10 CLS
 20 SOUND 133, 10
 30 SOUND 89, 10
 OK
 ■



Now **RUN** the program. The computer will play two tones, then stop. The screen should look like this:

OK
 ■

Add a tone.

You type: 40 SOUND 125, 10

RUN the program. The computer should play three tones, then stop. **LIST** the program.

LIST
 10 CLS
 20 SOUND 133, 10
 30 SOUND 89, 10
 40 SOUND 125, 10
 OK
 ■

EXPERIMENT

- Add a fourth tone (line 50).
- Add still more tones.
- Change any of the tones.
- Change line 10 to CLS 2 or CLS 3 or
- Change the durations of the tones.

Try various combinations of the following tone numbers:

89 108 125 133 147 159 170 176

Make the computer play a melody! You may also wish to consult Appendix G.

18. Combine sound and color.

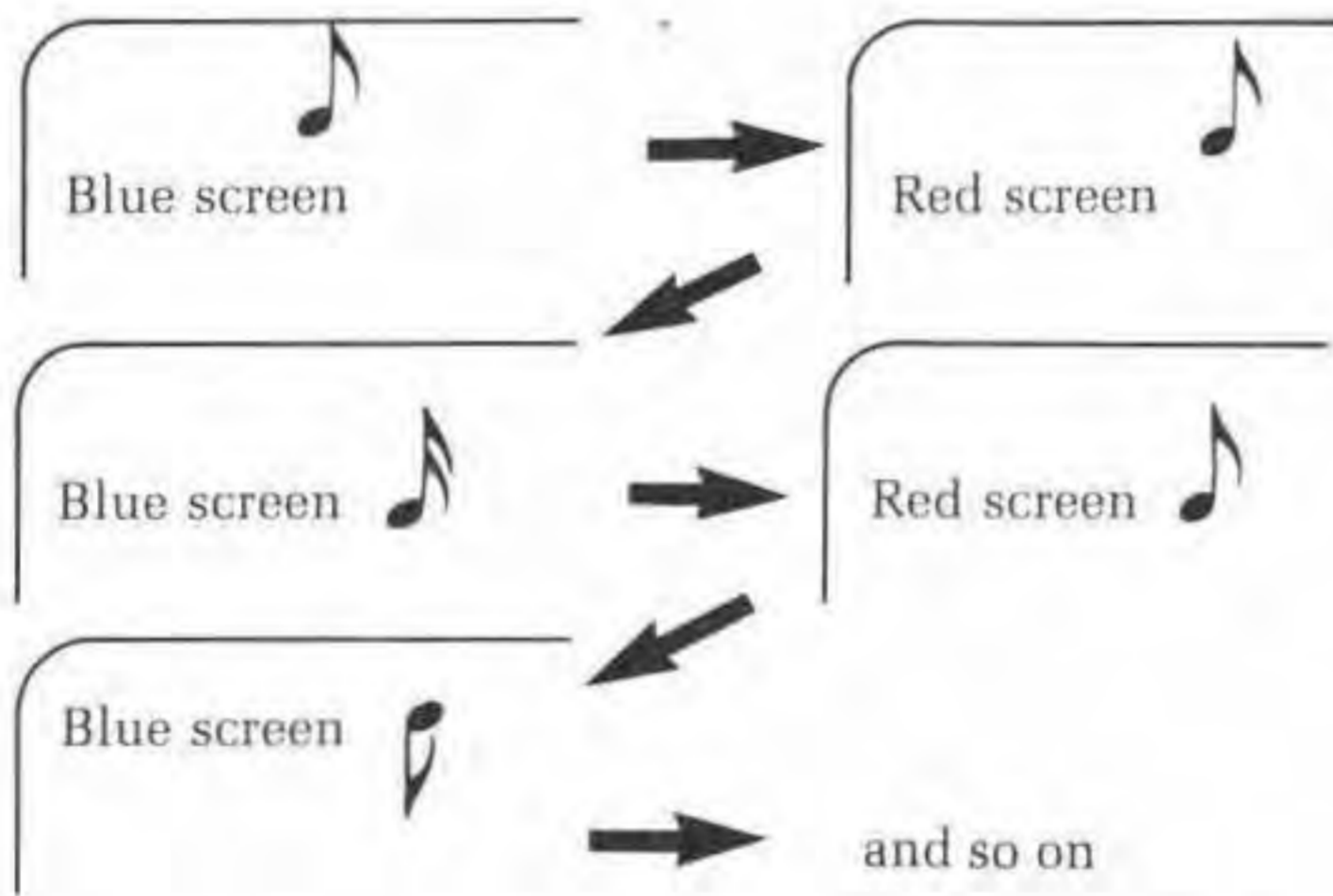
```

NEW
100 CLS 3
110 SOUND 133, 10
200 CLS 4
210 SOUND 89, 10
300 GOTO 100
    
```

Blue tone.

Red tone.

Now our emergency vehicle has a ululating siren and a flashing blue and red light!



Add a yellow tone to the program. Use tone number 176.

```

300 _____
310 _____
400 _____
    
```

```

300 CLS 2
310 SOUND 176, 10
400 GOTO 100
    
```

19. Our programs are getting longer. In order to make them more understandable to *people*, from now on we will frequently include REMARK statements. REMARK statements are for people. They explain what is happening in the program. The computer ignores REMARK statements. Here again is the program from the preceding frame. REMARK statements explain what each part of the program does.

```
100 REMARK  PLAY BLUE TONE
110 CLS 3
120 SOUND 133, 10

200 REMARK  PLAY RED TONE
210 CLS 4
220 SOUND 89, 10

300 REMARK  GO BACK AND PLAY IT AGAIN
310 GOTO 110
```

The following statements add a yellow tone. You write the REMARK statements.

```
400 _____
410 CLS 2
420 SOUND 176, 10

500 _____
510 GOTO 110
```

```
400 REMARK PLAY YELLOW TONE
500 REMARK GO BACK AND PLAY IT AGAIN
```

The REMARK statement is frequently abbreviated as REM. For example:

```
100 REM PLAY BLUE NOTE
200 REM PLAY RED NOTE
```

To call attention to REMark statements, we might sometimes write them like this:

```
100 REM ** PLAY BLUE NOTE
200 REM ** PLAY RED NOTE
```

20. If you play a musical instrument, you know about scales. You may have spent many joyful (?) hours practicing scales on a piano, guitar, flute, or other instrument. And, if you saw *The Sound of Music*, you know that it all begins with DO, RE, MI!

Here is a DO, RE, MI program for your Color Computer.

```

100 REM ** DO, RE, MI PROGRAM
110 CLS

200 REM ** PRINT AND PLAY 'DO'
210 PRINT "DO"
220 SOUND 89, 10

300 REM ** PRINT AND PLAY 'RE'
310 PRINT "RE"
320 SOUND 108, 10

400 REM ** PRINT AND PLAY 'MI'
410 PRINT "MI"
420 SOUND 125, 10

```

These tones will be close to Middle C, D, and E on a piano. However, they may not *exactly* match those tones.

Enter and RUN this program. Then add FA. The tone number for FA (of DO, RE, MI, FA fame) is 133.

```

500 REM **PRINT AND PLAY 'FA'
510 _____
520 _____

```

```

-----

510 PRINT "FA"
520 SOUND 133, 10

```

If you want to do the entire scale, here are the tone numbers and the musical letter notes.

DO	89	Middle C
RE	108	D
MI	125	E
FA	133	F
SO	147	G
LA	159	A
TI	170	B
DO	176	C

For more information, see Appendix G.

21. Add color. In the program of the preceding frame, replace lines 210, 310, and 410 as follows:

210 CLS 1: PRINT "DO"	Two statements on each line.
310 CLS 2: PRINT "RE"	Remember? If not, review frame 17
410 CLS 3: PRINT "MI"	in Chapter 2.

Hmmm . . . now try it this way.

```
210 CLS 1: PRINT "DO";  
310 CLS 2: PRINT "RE";  
410 CLS 3: PRINT "MI";
```



Or this way:

```
210 CLS 8: PRINT "DO";  
310          PRINT "RE";  
410          PRINT "RE";
```

Or yet another way. Experiment!

SELF-TEST

Congratulations! You have muddled through—oops, sorry; of course, you have studiously consumed—no, you have—well, anyhow, welcome to the Self-Test.

1. Before entering a BASIC program, you usually type NEW and press **ENTER**. Why? _____
 2. How do you tell the computer to print on the screen a listing of a program stored in its memory? _____
 3. How do you tell the computer to execute (do or obey) a program stored in its memory? _____
-

4. Which program would produce the RUN shown below?_____

<i>Program A</i>	<i>Program B</i>
10 CLS	10 CLS
20 PRINT "HA HA ";	20 PRINT "HA HA "
30 SOUND 176, 1	30 SOUND 176, 1
40 GOTO 20	40 GOTO 20

```

HA HA HA HA HA HA HA HA HA HA HA
HA HA HA HA HA HA HA HA HA HA H
HA HA HA HA HA HA HA HA HA HA HA
HA HA HA HA

```

and so on.

5. Teach the computer to cry. Write a program to fill the screen with BOO HOO.
6. Show how the screen will look after you enter and RUN the following program.

```

100 CLS
110 PRINT "   *"
120 PRINT "   *"
130 PRINT "   *"
140 PRINT "*****"
150 PRINT "*****"
160 PRINT "   *"
170 SOUND 89, 20

```

7. You complete the REMark statements in the following program.

```
100 REM _____  
110 CLS 4  
120 PRINT "ROSES ARE";  
130 SOUND 89, 20  
200 _____  
210 CLS 3  
220 PRINT "VIOLETS ARE";  
230 SOUND 108, 20  
300 _____  
310 CLS 1  
320 PRINT "FOR YOU, DEAR VALENTINE";  
330 SOUND 125, 20  
400 REM _____  
410 CLS 6  
420 PRINT "MY HEART IS";  
430 SOUND 176, 40
```

8. Write a program to play a short tune in the key of C.

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. This erases, removes, or deletes any old program that might be in the computer's memory. If you don't do this, statements of an old program might get mixed up with statements of your new program, thus causing mysterious and unpredictable results when you RUN the program. (frame 4).
2. Type LIST and press the **ENTER** key. (frame 7)
3. Type RUN and press the **ENTER** key. No, the computer will not run away. Instead, it will "run" (execute, carry out, do, obey) the program in its memory. Hmmm . . . what happens if there isn't any program in memory? Well, that's OK. Try it and find out. (frame 8)
4. Program A produces the computer hilarity shown. The difference in the programs is the semicolon at the end of the PRINT statement in line 20 of program A. Program B produces a more subdued laughing computer, as shown below.

```
HA HA  
HA HA  
HA HA  
HA HA  
HA HA
```

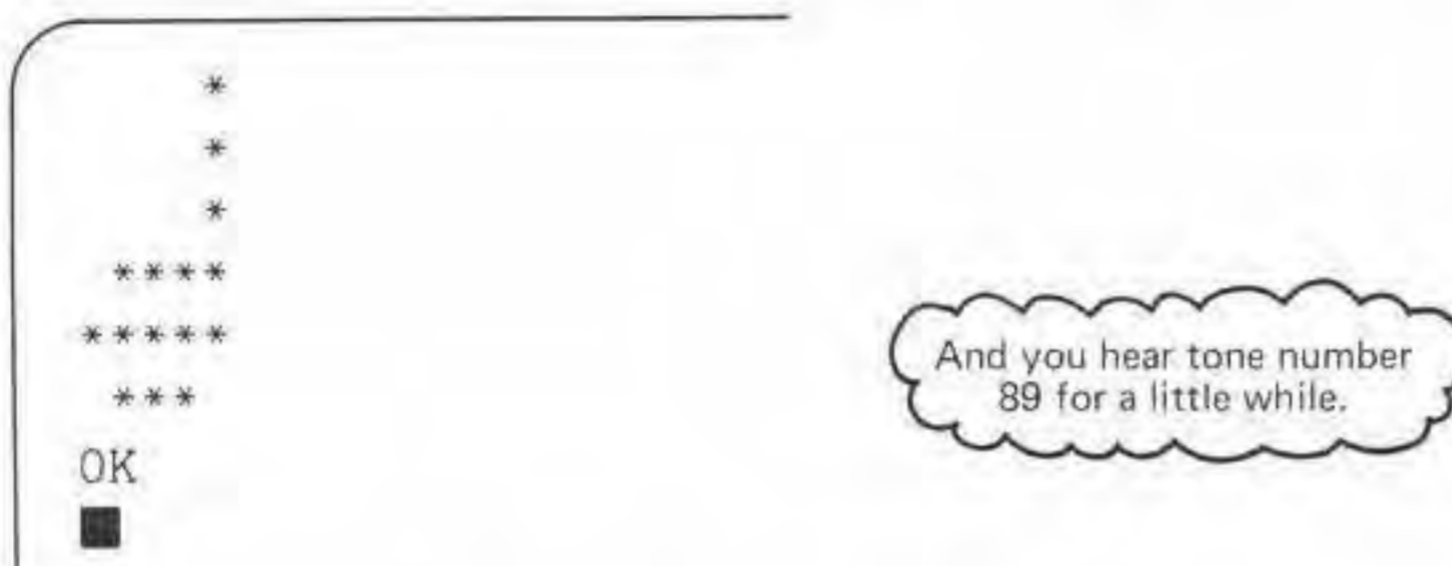
and so on. (frames 7, 9, 11)

5. It might look like this. Your program may be different. That's OK, as long as the computer does what you want it to do.

```
10 CLS 3
20 PRINT "BOO HOO ";
30 SOUND 153, 1
40 GOTO 20
```

(frame 11)

6. This notable (oops—sorry) program causes the screen to look like this:



7. We did it this way. Sorry about the atrocious pun in line 400.

```
100 REM**ROSES ARE RED (RED SCREEN)
200 REM**VIOLETS ARE BLUE (BLUE SCREEN)
300 REM**FOR YOU, DEAR VALENTINE (GREEN SCREEN)
400 REM**MY HEART IS SIGHIN' (CYAN SCREEN)
```

(frames 14, 20, 21)

8. No answer. See Appendix G for information on tone numbers and notes on the musical staff.



CHAPTER FOUR

Number Boxes

As you learn more and more about BASIC, you will find it easier and easier to get the Color Computer to do what you want it to do.

In this chapter, you will learn about places in the computer's memory which we call *number boxes*. Think of number boxes as places to store numbers or as *placeholders* where numbers can be stored while the computer uses them.

Number boxes are identified by labels called *numeric variables*. You will use numeric variables to represent numbers in BASIC programs.

You will learn three different ways to put numbers into number boxes, or *assign values to numerical variables*.

When you finish this chapter, you will be able to:

- Recognize and use numeric variables;
- Assign values to numeric variables;
- Use the INPUT statement to enter values of a numeric variable from the keyboard;
- Use READ and DATA statements to supply values of a numeric variable;
- Use numeric variables in PRINT, CLS, and SOUND statements.

Soon you will get the computer to do more while you do less!



1. Imagine that, deep down inside the computer, there are 26 little number boxes. Each number box can hold one number at any one time.

A	7	H		O		V	
B	5	I		P		W	
C	8	J		Q		X	
D	10	K	1	R		Y	
E		L		S		Z	
F		M		T	89		
G		N	0	U			

Some of the boxes have numbers in them. For example, 7 is in box A and 5 is in box B.

- What number is in box T? _____
- What number is in box D? _____
- What box contains the number 1? _____
- What box contains the number 8? _____
- What box contains zero (0)? _____
- You put the number 100 into box Z.

(a) 89; (b) 10; (c) K; (d) C; (e) N;
 (f) Box Z should now look like this:

Z

100

2. Boxes C and T are shown below. They are empty. Use a pencil to do the following.

C T

- Put 3 into box C.
- Put 89 into box T.
- Put 125 into box T. But wait! A box can hold only *one* number at a time. Before you can put 125 into box T, you must first erase the 89 that you put there previously.

(a) C ; (b) T ; (c) T

As you will soon learn, when the computer puts a number into a box, it first erases the previous number (if any) that was in the box.

3. The letters that identify boxes (A, B, C, and so on) are called *variables*. The number in box A is called the *value of A*; the number in box B is the *value of B*; the number in box C is the *value of C*; and so on.

Below are three number boxes, labelled C, T and D.

C T D

- C, T and D are called_____.
- The value of C is_____.
- The value of D is_____.
- The number 108 is the value of what variable?_____

(a) variables; (b) 7; (c) 20; (d) T

Number boxes can hold numbers only. The labels on number boxes (A, B, C, etc.) are sometimes called *numeric variables*.

- Flower boxes hold flowers.
- Candy boxes hold candy.
- Number boxes hold numbers.

Later you will also learn about *string boxes*, which can hold . . . (suspense) . . . strings! And you will also learn about *string variables*, whose values are strings.

4. How do you put numbers into number boxes? It's easy!

- Clear the screen
- Type: A = 7 and press **ENTER**

The screen looks like this:

```
A = 7
OK
■
```

The computer has put the number 7 into box A. Or, the computer has assigned a value 7 to the variable A. Note how you did this.

A = 7



In BASIC, $A = 7$ is a statement that tells the computer to put the number 7 into number box A. In other words, $A = 7$ tells the computer, "Assign the number 7 as the value of the variable A."

5. In frame 4, you told the computer to assign the number 7 as the value of the variable A. Now, find out what happened. Find out what is in box A. Find out what the value of A is.

- Clear the screen.
- Type: `PRINT A` and press **ENTER**.

The screen looks like this:

```
PRINT A
7
OK
■
```

If you forgot to clear the screen before typing `PRINT A`, the screen might now look like this:

```
A = 7
OK
PRINT A
7
OK
■
```

Remember: To tell the computer to print the number that is in number box A (or to print the value of variable A), you type:

PRINT A

6. How would you assign the number 89 as the value of the variable T?

(a) You type: _____

What would you then type to find out the value of T?

(b) You type: _____

(a) T = 89; (b) PRINT T

The screen might look like this:

```
T = 89
OK
PRINT T
  89
OK
■
```

Note the difference between the two PRINT statements below.

WITH" WITHOUT"

PRINT "T"

Tells the computer to print the string T which is enclosed in quotation marks.

PRINT T

Tells the computer to print the value of the variable T.

Try this:

First type: T = 89
 Then type: PRINT T
 Then type: PRINT "T"

The screen will look like this:

```
T = 89
OK
PRINT T
 89
OK
PRINT "T"
T
OK
■
```

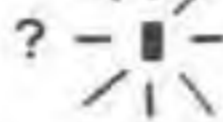
7. The INPUT statement is very useful for box stuffing. The following program introduces the INPUT statement.

```
10 CLS
20 INPUT T
30 PRINT T
40 GOTO 20
```



Enter the above program, type RUN, and press the **ENTER** key.

You get a question mark and the cursor.



The computer is doing the INPUT statement. The statement:

```
20 INPUT T
```

tells the computer:

- (1) type a question mark;
- (2) turn on the cursor;
- (3) wait for someone to type a number. When you type a number and press **ENTER**, the computer puts the number into box T and goes on to the next statement in line-number order.

Type 89 and press **ENTER**.

The screen will look like this:

```
? 89
  89
? █
```

The computer wants another number, a new value for T. Cooperate with your ever-patient computer. Type 125 and press **ENTER**.

```
? 89 ← You entered 89 as the value of T.
  89 ← It printed the value of T.
? 125 ← You entered 125 as the value of T.
  125 ← It printed the value of T.
? █
```

What? It wants still another number? How do you tell the computer to stop?

Press the **BREAK** key!

REMEMBER: To stop the computer, press the **BREAK** key.

8. Why does the computer keep asking for a value of T? Draw arrows to show how the program works (as in frames 10 and 16 in Chapter 3).

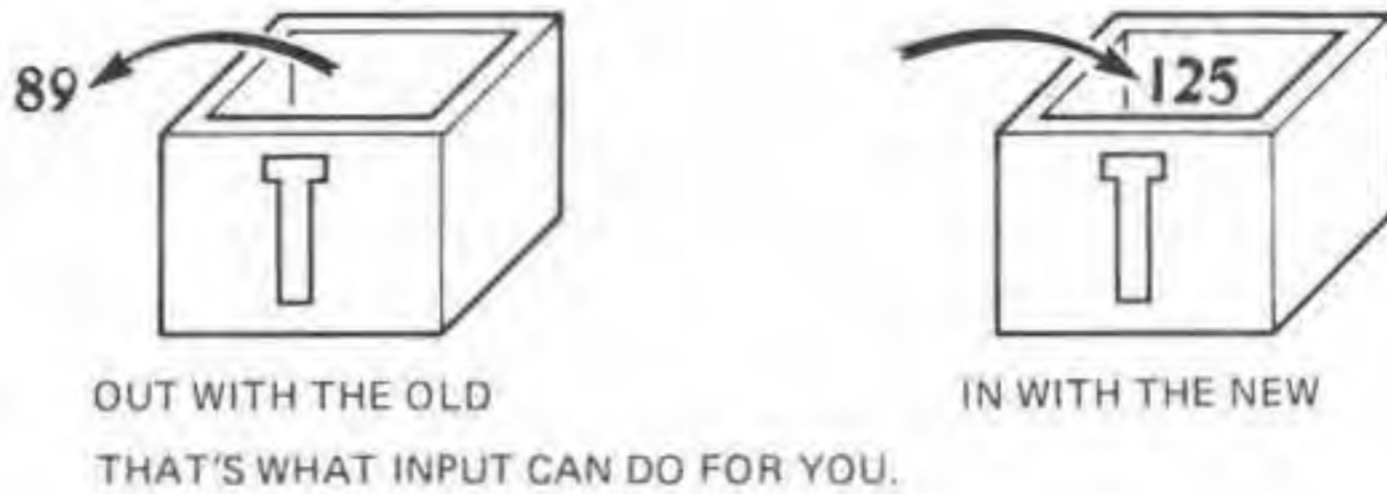
```
10 CLS
20 INPUT T
30 PRINT T
40 GOTO 20
```

```
10 CLS
  ↓
20 INPUT T ←
  ↓
30 PRINT T
  ↓
40 GOTO 20 ←
```

Aha! After printing the value of T, the GOTO 20 statement sends the computer back to the INPUT T statement in line 20.



The computer goes from line 20 to line 30 only after you type a value of T and press **ENTER**. This value *replaces* the previous value of T.

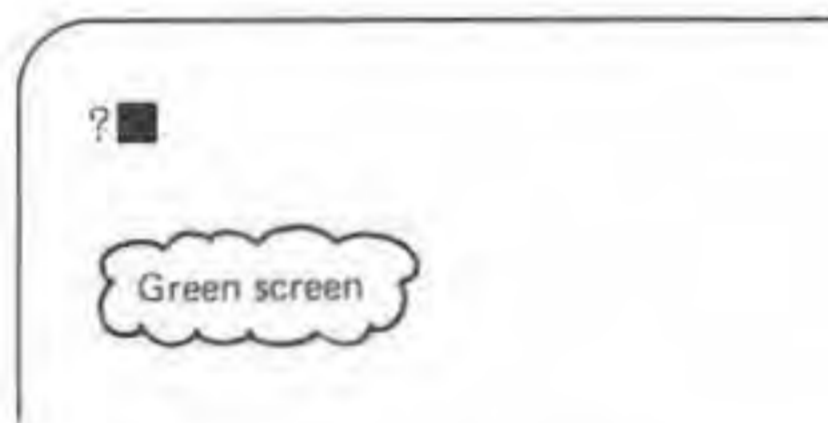


9. You can use a variable in a CLS statement.

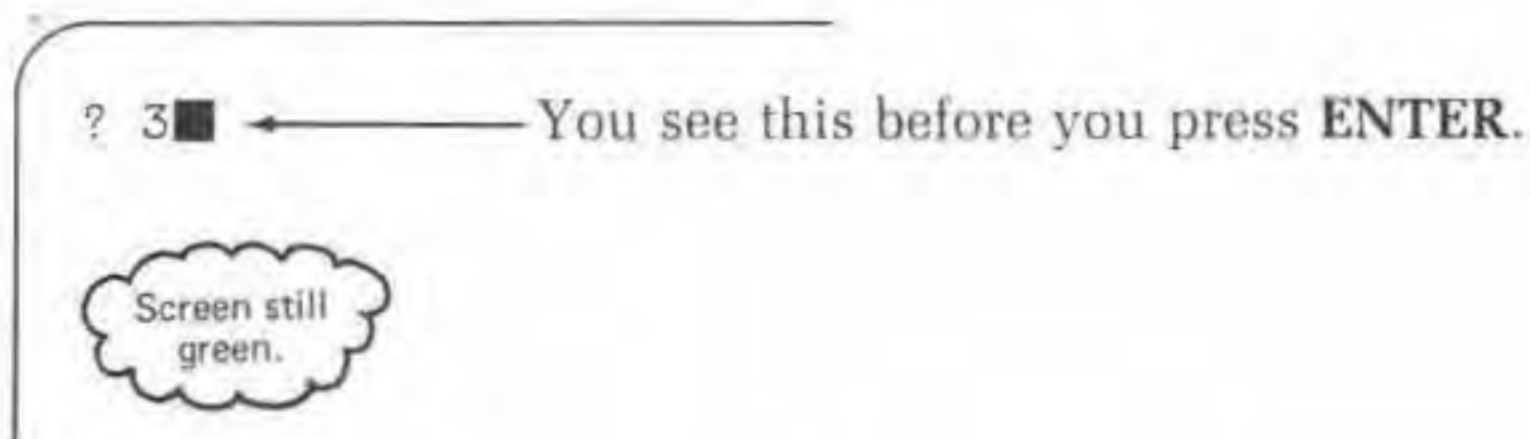
```

10 CLS
20 INPUT C    ↗ Let's use variable C for "Color."
30 CLS C
40 GOTO 20
    
```

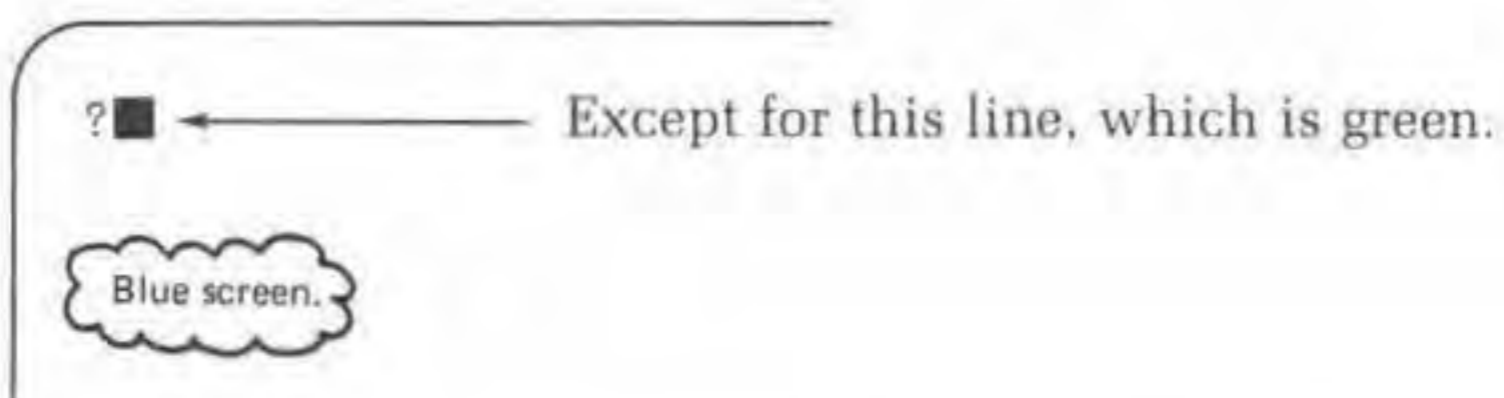
Try it. Enter the program and RUN it.



Now type 3 as the value of C.



And press **ENTER**.



Try another color. Type 4 for red, 8 for orange, or the color of your choice. Any number from 0 to 8 is OK. Try -1 or 9 just to see what happens.

10. You can use a variable in a SOUND statement.

```
10 CLS
20 INPUT T
30 SOUND T, 10
40 GOTO 20
```

A RUN might go like this.

```
? ■          Waiting for a value of T.
```

Enter 89 as the value of T and press **ENTER**.

```
? 89
? ■          The computer plays tone number 89 for a little
              while, then asks for a new value of T.
```

Next, supply 108 as the value of T.

```
? 89
? 108
? ■         The computer plays tone number 108 for a little
              while, then asks for a new value of T.
```

Suppose you enter 125 as the next tone. Show what the screen will look like.

```
? 89
? 108
? 125
? ■
```



Experiment! Try low tones ($T = 1$), high tones ($T = 218$), still higher tones ($T = 255$), and tones that don't exist ($T = 0$, $T = -1$, $T = 256$). Of course, expect to get ERROR messages when you use nonexistent tones.

11. Combine color and sound.

```

10 CLS
20 INPUT C
30 INPUT T
50 CLS C
60 SOUND T, 10
70 GOTO 20

```

Yes, line 40 is missing. Be patient. Enter the program as shown. Line 40 will make its dramatic entrance in the next frame.

A RUN begins like this:

```

? █

```

It wants the value of C.

Type 3 as the value of C and press **ENTER**.

```

? 3
? █

```

Now it wants the value of T.

Type 89 as the value of C. *Before* you press **ENTER**, this is what you see.

```

? 3
? 89 █

```

Remember, this is *before* you press **ENTER**.

Now press **ENTER**. The screen goes blue, all blue, while tone 89 plays. Then the computer goes back to the first INPUT statement (line 20), ready for a new value of C.

```

? █

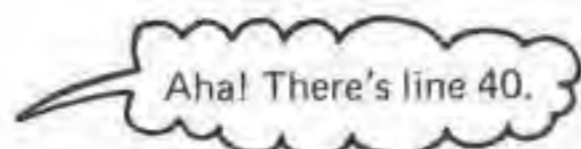
```

The screen is mostly blue, except for the top line, which is green.

Go ahead, try various values of C and T to find out what happens.

12. How about a variable duration in the SOUND statement? Of course, Use D for duration, and complete the following program.

```
10 CLS
20 INPUT C
30 INPUT T
40 INPUT _
50 CLS
60 SOUND T, _
70 GOTO 20
```



```
40 INPUT D
60 SOUND T, D
```

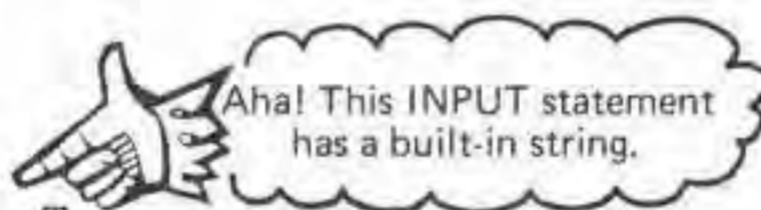
A RUN might go like this:

```
? 3      Color (C)
? 89     Tone (T)
? 20 ■   Duration (D)
```

Press **ENTER** and see/hear it happen.

13. The INPUT statement causes the computer to put a question mark on the screen, then wait for someone to enter something. Wouldn't it be nice if, instead of just a lonely question mark, the computer would also tell you what it wanted? Easy! The following program has an "enhanced" INPUT statement.

```
10 CLS
20 INPUT "TONE NUMBER"; T
30 SOUND T, 10
40 GOTO 20
```



The statement:

```
20 INPUT "TONE NUMBER"; T
```

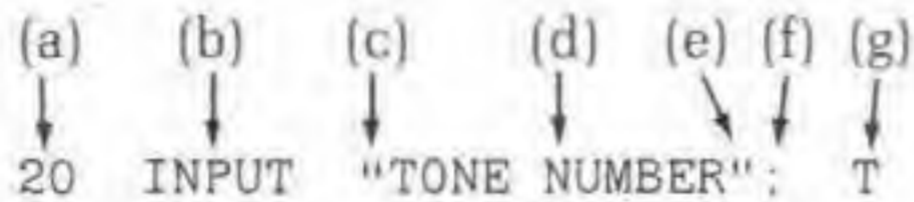
tells the computer to:

- (a) print the string TONE NUMBER on the screen;
- (b) print a question mark;
- (c) turn on the cursor; and
- (d) wait for someone to type a number and press **ENTER**.

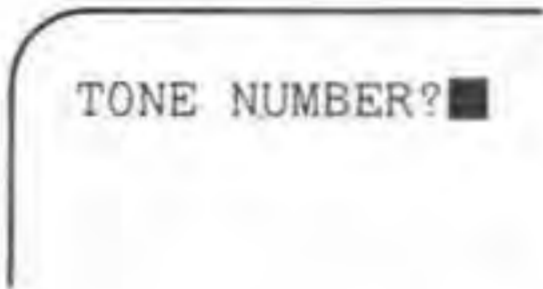
The statement consists of:

- (a) a line number,
- (b) the word INPUT,
- (c) quotation marks,
- (d) a string,
- (e) quotation marks,
- (f) a semicolon, and
- (g) a numeric variable.

Using the above list, put the appropriate letter [(a), (b), (c), etc.] above each arrow.



14. As usual, to find out what a program does, enter it into the computer and RUN it.



Now you know what the computer wants! Enter several tones, one after the other. For example:

```

TONE NUMBER? 89      Type tone, press ENTER
TONE NUMBER? 108     Type tone, press ENTER.
TONE NUMBER? 125     Type tone, press ENTER.
TONE NUMBER? 133     .
TONE NUMBER? 147     .
TONE NUMBER? ■      and so on. Press BREAK to quit.

```

Complete the following program so that the computer asks for TONE NUMBER and DURATION NUMBER.

```

10 CLS
20 INPUT "TONE NUMBER"; T
30 INPUT _____
40 SOUND T, D
50 GOTO 20

```

```

30 INPUT "DURATION NUMBER"; D

```

15. A RUN of the program in the preceding frame might look like this:

```

TONE NUMBER? 89
DURATION NUMBER? 10
TONE NUMBER? 176
DURATION NUMBER? 20
TONE NUMBER? 218
DURATION NUMBER? 30
TONE NUMBER? ■

```

Reduce eyestrain. Make stuff on the screen more readable. Put an "empty line" after each TONE NUMBER and DURATION NUMBER. Add the following line to the program:

```

45 PRINT

```

LIST the program. It should look like this:

```

LIST
10 CLS
20 INPUT "TONE NUMBER: T
30 INPUT "DURATION NUMBER"; D
40 SOUND T, D
45 PRINT ← You added this statement (we hope!).
50 GOTO 20
OK
■
    
```

Are you surprised that line 45, which you added, appeared between lines 40 and 50? _____

We hope you said NO. The computer always puts all lines in its memory in line-number order.

16. RUN the modified program.

```

TONE NUMBER? 89
DURATION NUMBER? 10
TONE NUMBER? 176
DURATION NUMBER? 20
TONE NUMBER? 218
DURATION NUMBER? 30
TONE NUMBER? ■
    
```

← Guess! Why did the computer print this empty line? _____

Because line 45 (45 PRINT) tells it to PRINT . . . well, nothing! Of course, this happens after every pair of TONE, DURATION numbers.

The statement:

```
45 PRINT
```

tells the computer to print an "empty line" on the screen. Use it when you want a line space to separate things vertically on the screen.

17. We confess. We didn't tell the whole truth. A number box, or numeric variable, can be labeled with more than just a single letter. For example:

T1 T2 T3

T1, T2, and T3 are three distinct numeric variables. T1 and T2 appear in a starring role in the following program.

```

100 REM ** KEYBOARD TONES
110 CLS

200 REM ** ENTER TONE NUMBERS
210 INPUT "1ST TONE"; T1
220 INPUT "2ND TONE"; T2

300 REM ** PLAY TONES
310 SOUND T1, 10
320 SOUND T2, 10

400 REM ** PLAY ANOTHER, SAM
410 GOTO 110

```

From now on, we will use lots of REMS to explain what is happening. We will also use vertical spacing to make the program easier for you to read. These spaces do not appear when you LIST the program.

Play around with this program. Then make it into a three-tone program by adding two statements.

```

230 _____
330 _____

```

```

230 INPUT "3RD TONE"; T3
330 SOUND T3, 10

```

18. Ha! Bet you are wondering whether you can add a fourth tone (T4), a fifth tone (T5), and so on. Yes, you can—up to T9.

BEWARE! The computer will confuse T10 with T1.

Alas, lazy computer looks only at the *first two* characters of a variable.

You can use KOLOR as a variable, but if you do, don't also use KOLA or KOOLAID.

Why? Because KOLOR, KOLA, and KOOLAID all have the same first two characters.

KOLOR KOLA KOOLAID

You can use any *one* of these as a variable.

You can't use TONE as a variable because it begins with TO, which is a reserved word.

You type: TONE = 89
It prints: ?SN ERROR

Other reserved words are PRINT, SOUND, CLS, GO, and INPUT. These may not be used as variables, nor can a variable *begin* with a reserved word.

You can't use: GOPHER PRINTER SOUND TTOTAL

Appendix E is a list of reserved words for the Color Computer.

19. You can put more than one variable in an INPUT statement.

```
100 REM ** KEYBOARD TONES
110 CLS

200 REM ** ENTER TONE NUMBERS
210 INPUT "TWO TONES, PLEASE": T1, T2

300 REM ** PLAY TONES
310 SOUND T1, 10
320 SOUND T2, 10

400 REM ** GO BACK FOR MORE TONES
410 GOTO 210
```

Line 210 is an INPUT statement with two variables.

When the computer asks TWO TONES, PLEASE?, you must enter two tones. You can press **ENTER** after *each* tone as follows.

If you supply one tone, it types a double question mark. It wants the second tone.

```
TWO TONES, PLEASE? 89
?? 108
```



Or, you can type both tones separated by a comma, then press **ENTER**.

```
TWO TONES, PLEASE? 89, 108
```



Remember, if you put two variables in an INPUT statement, you must supply two values.

```
210 INPUT "TWO TONES, PLEASE": T1, T2
```

```
TWO TONES, PLEASE? 89, 109
```

Why did the computer print a question mark after TWO TONES, PLEASE? _____

An INPUT statement *always* types a question mark, then turns on the cursor. Then, of course, the computer waits for you to do something.

20. Complete the following three-tone program.

```
100 REM ** KEYBOARD TONES
110 CLS
200 REM ** ENTER TONE NUMBERS
210 _____
300 REM ** PLAY TONES
310 _____
320 _____
330 _____
400 REM ** GO BACK FOR MORE TONES
410 GOTO 210
```

```
210 INPUT "THREE TONES, PLEASE": T1, T2, T3
310 SOUND T1, 10
320 SOUND T2, 10
330 SOUND T3, 10
```

Four tones? Five tones? More tones? You can do it!

21. Why not enter both tone number and duration number for each tone? Complete the following program to play three tones. Use D1, D2, and D3 for the duration numbers of the three tones.

```
100 REM ** KEYBOARD TONES
110 CLS

200 REM ** ENTER TONES AND DURATIONS
210 INPUT "TONE, DURATION": T1, D1
220 _____
230 _____

300 REM ** PLAY TONES
310 _____
320 _____
330 _____

400 REM ** GO BACK FOR MORE
410 GOTO 210
```

```
220 INPUT "TONE, DURATION": T2, D2
230 INPUT "TONE, DURATION": T3, D3
310 SOUND T1, D1
320 SOUND T2, D2
330 SOUND T3, D3
```

22. Write a program to play three (3) color tones. For each tone, ask: COLOR, TONE, DURATION? Follow the "outline" suggested by the following REM statements.

```
100 REM ** KEYBOARD COLOR TONES

200 REM ** ENTER COLOR, TONE DURATION

300 REM ** PLAY COLOR TONES

400 REM ** GO BACK FOR MORE
410 GOTO 210
```

If your program works, don't even look at our feeble attempt!

```

100 REM ** KEYBOARD COLOR TONES
100 CLS

200 REM ** ENTER COLOR, TONE, DURATION
210 INPUT "COLOR, TONE, DURATION"; C1, T1, D1
220 INPUT "COLOR, TONE, DURATION"; C2, T2, D2
230 INPUT "COLOR, TONE, DURATION"; C3, T3, D3

300 REM ** PLAY COLOR TONES
310 CLS C1: SOUND T1, D1
320 CLS C2: SOUND T2, D2
330 CLS C3: SOUND T3, D3

400 REM ** GO BACK FOR MORE
410 GOTO 210

```

VARIATION: Change line 410 to 410 GOTO 110

23. There is yet another way to stuff numbers into number boxes, or—in *computerese*—assign values to numerical variables. Use READ and DATA statements, as shown in the following program.

```

10 CLS
20 READ N           ← This is a READ statement.
30 PRINT N
40 GOTO 20
50 DATA 1, 22, 333 ← This is a DATA statement.

```

The statement:

```
20 READ N
```

tells the computer to read one number from a DATA list and put the number into box N (assign the number as the value of N). Every time line 20 is executed, the next number in the DATA list is read and assigned as the value of N.

This is how the screen looks when you RUN the program:

```

1
22
333
?OD ERROR IN 20
OK
■

```


The error message (?OD ERROR IN 20) means "out of data in line 20."

- (a) How many values are in the DATA statement (line 50)? _____
 (b) Were these values all printed on the screen? _____
 (c) Why do you suppose the computer printed an ?OD ERROR IN 20?

(a) 3; (b) yes; (c) It printed all the values, then tried to print a fourth value. Since there are only three values in the DATA statement, it printed an error message. That's OK, since it was finished doing what we wanted it to do.

24. Look at the DATA statement in line 50. Commas separate the numbers in the list.

```

50 DATA 1. 22. 333
  
```

No comma here. Commas between numbers. No comma here.

Write a DATA statement using the following numbers:

89 109 125 133 147 159 170 176

50 DATA _____

50 DATA DATA 89, 89, 109, 125, 133, 147, 159, 170, 176

25. READ and DATA statements make it easy to play music. Here is a program to play a DO, RE, MI scale in the key of C.

```

100 REM ** READ AND DATA MUSIC
110 CLS

200 REM ** READ A TONE NUMBER
210 READ T

300 REM ** PLAY A TONE
310 SOUND T, 10

400 REM ** GO BACK FOR ANOTHER
410 GOTO 210

900 REM ** HERE ARE THE TONE NUMBERS
910 DATA 89, 108, 125, 133
920 DATA 147, 159, 170, 176
  
```

We intentionally used two DATA statements to hold the tone numbers. We could have put all the tone numbers in *one* DATA statement, as follows:

```
910 DATA 89, 108, 125, 133, 147, 159, 170, 176
```

Or use 3 DATA statements, or 4, or more, if you wish. Want to have the computer play a tune? Put the tone numbers for the tune in DATA statements, as many as you wish! Try this one:

```
910 DATA 125, 108, 89, 108
920 DATA 125, 125, 125, 108
930 DATA 108, 108, 125, 147, 147
```

VARIATION: Change block 200 of the program, as follows:

```
200 REM ** READ AND PRINT TONE NUMBER
210 READ T
220 CLS
230 PRINT T
```

Try it and see what happens. Then delete line 220 and try it again. EXPERIMENT!

26. A READ statement can read more than one number. In the following program, the READ statement reads two numbers. It assigns the first number as the value of T and the second number as the value of D.

```
100 REM ** READ AND DATA MUSIC
110 CLS

200 REM ** READ TONE AND DURATION
210 READ T, D

300 REM ** PLAY A TONE
310 SOUND T, D

400 REM ** GO BACK FOR MORE
410 GOTO 210

900 REM ** TONE AND DURATION NUMBERS
910 DATA 89, 10 ← First tone and duration.
920 DATA 108, 20 ← Second tone and duration.
930 DATA 125, 30 ← Third tone and duration.
```

Show how to put all three tones and durations in one DATA statement

910 DATA _____

910 DATA 89, 10, 108, 20, 125, 30

NOT this way: 910 DATA 89, 108, 125, 10, 20, 30

27. We usually put DATA statements near the end of a program. Actually, they can be anywhere. All three of these short programs produce exactly the same results.

10 CLS	10 CLS	10 CLS
20 READ N	20 READ N	20 DATA 1, 22, 333
30 PRINT N	30 DATA 1, 22, 333	30 READ N
40 GOTO 20	40 PRINT N	40 PRINT N
50 DATA 1, 22, 333	50 GOTO 20	50 GOTO 30

28. Write a program to play color tones, using READ and DATA. Your program should read values of C (Color), T (Tone), and D (Duration), clear the screen to color C, play tone T for duration D, then do it all again. Review frames 22 and 25 if you have trouble doing this.

100 REM ** READ & DATA COLOR TONES

200 REM ** READ COLOR, TONE, DURATION

300 REM ** PLAY COLOR TONE

400 REM ** DO IT AGAIN

900 REM ** VALUES OF C, T, D

910 DATA 1, 89, 10, 2, 108, 10

920 DATA 3, 125, 10, 4, 133, 10

930 DATA 5, 147, 10, 6, 159, 10

940 DATA 7, 170, 10, 8, 176, 20

We put values for two color tones in each DATA statement.

We put values for two color tones in each DATA statement.

We did it like this:

```

210 READ C, T, D
310 CLS C: SOUND T, D or 310 CLS C
                        320 SOUND T, D

```

Self-Test

Take a break. Do something physical and relaxing. Jog, play tennis, stretch, ride a bike, go for a walk. Then browse through this Self-Test to assure yourself that you are learning more and more about BASIC.

1. What might you expect to find if you peeked into any of the following boxes?
 - (a) Flower box _____
 - (b) Candy box _____
 - (c) Batter's box _____
 - (d) Safe deposit box _____
 - (e) Post Office box _____
 - (f) Number box _____
2. In BASIC, a number box is identified by a label, or name. What is this label called? _____
3. Suppose that the number 7 is in number box A. Then 7 is called the _____ of numeric variable A.
4. Without actually using the computer, complete each of the following. (It's OK to guess!)
 - (a) You type: Z = 13
 It prints: OK
 You type: PRINT Z
 It prints: _____

 Cursor → ■
 - (b) You type: P = 7
 It prints: OK
 You type: P = 33
 It prints: _____
 You type: PRINT P

 ■

(c) You type: A = 7
It prints: _____
You type: B = A
It prints: _____
You type: PRINT B
It prints: _____



(d) You type: C = 12
It prints: _____
You type: PRINT "C"
It prints: _____



5. The statement: 20 INPUT "WHAT NOTE"; N tells the computer to:
- (a) _____
(b) _____
(c) _____
(d) _____
6. Describe what might happen if you RUN the following program.

```
10 CLS
20 INPUT "HOW OLD ARE YOU"; AGE
30 SOUND AGE, 20
40 GOTO 10
```

7. Describe what will happen if you RUN the following program.

```
10 CLS
20 READ T
30 SOUND T, 1
40 GOTO 20
50 DATA 10, 20, 30, 40, 50
60 DATA 60, 70, 80, 90, 100
70 DATA 110, 120, 130, 140, 150
80 DATA 160, 170, 180, 190, 200
90 DATA 210, 220, 230, 240, 250
```

8. Guess what will happen if you RUN this program and its variations:

```

100 REM ** MYSTERY PROGRAM
110 T = 1
120 D = 1

200 REM ** PRINT AND PLAY TONE T
210 CLS
220 PRINT T
230 SOUND T, D

300 REM ** CHANGE VALUE OF T
310 T = T + 1

400 REM ** PLAY IT AGAIN
410 GOTO 210

```

VARIATIONS

- (a) 110 T = 255
310 T = T - 1
- (b) 110 T = 1
310 T = 2*T
- (b) 110 T = 255
310 T = T/2

See Appendix D for information on Color Computer arithmetic, using +, -, *, and /.

Answers to Self-Test

The numbers in parentheses refer to the frames in the chapter where the topic is discussed.

1. (a) Flowers. Oh, you forgot to water the flowers? Well, maybe you found wilted flowers.
- (b) Candy, if you get there soon after the box is opened. Otherwise, you might just find crumpled paper wrappers.
- (c) Usually, nothing. However, during baseball season, you will occasionally find a batter.
- (d) Almost anything! Worthless stock, wills, jewelry, that insurance premium you spent days looking for at home, heirlooms, your first love letter, . . .
- (e) Mostly bills, unfortunately. Also lots of junk mail. Occasionally something worth reading such as the *Dymax Gazette* or the *ComputerTown, USA! News Bulletin*.
- (f) Numbers! Crunchy numbers to use in BASIC statements. (frames 1, 2)

2. A numeric variable. (frame 3)
3. Value. (frames 3–6)
4. (a) 13 (b) OK (c) OK (d) OK
OK OK OK C
OK 7

(frames 4–6)
5. (a) Print the string WHAT NOTE.
(b) Print a question mark.
(c) Turn on the cursor.
(d) Wait for someone to enter a value of N. (frame 13)
6. First, the screen looks like this:

```
HOW OLD ARE YOU?■
```

If you type your age and press **ENTER**, the computer uses your age as a tone number and plays a tone.

Young people will hear low tones.

Older people will hear higher tones.

Venerable dragons will hear nothing and see an error message.

```
HOW OLD ARE YOU? 1234
?FC ERROR IN 30
OK
■
```

7. You will hear a most unmusical scale! (frames 23–25)
 8. Try them and find out! To slow things down, change line 120 to $D = 10$ or $D = 20$ or whatever suits your eyespeed.
-

CHAPTER FIVE

String Boxes

Now you know about number boxes, handy tools for doing things with numbers. Next, put some *string boxes* in your Color Computer toolbox.

String boxes are places in the computer's memory that can store strings. String boxes are identified by names, or labels, called *string variables*.

In this chapter, you will learn how to put strings into string boxes and then use them in many ways.

When you finish this chapter, you will be able to:

- Recognize and use string variables;
- Assign values to string variables;
- Use the INPUT statement to enter values of a string variable from the keyboard;
- Use READ and DATA statements to supply values of a string variable;
- Write programs using both numeric and string variables;
- Use SOUND as a noisy time delay to slow things down so you can see what is happening;
- Use FOR and NEXT statements for silent time delays.

Your power increases! More tools appear in your computer toolbox. You become more able to get the computer to do what you want it to do!

1. Put your imagination to work again. This time, imagine that inside the computer there are a bunch of little *string boxes*.



String boxes can be very short or quite long or in-between.

A string box is identified by a label, or name, consisting of a letter followed by a dollar sign. For example, here are boxes called A\$, B\$, N\$, X\$, and Z\$.



Into a string box you can put . . . (suspense, trumpets, fanfare!) . . . a *string*.

- A string can be a name: GEORGE FIREDRAKE
- A string can be a telephone number: 415-323-6117
- A string can be a message: TRUST YOUR PSYCHIC TAILWIND
- A string can be gibberish: 12BZ%+A

A string can include almost any keyboard character. One important exception is the quotation mark ("). It can't be part of a string. Instead, quotation marks are sometimes used to enclose a string, as shown below.

"REALITY EXPANDS TO FIT THE AVAILABLE FANTASIES."

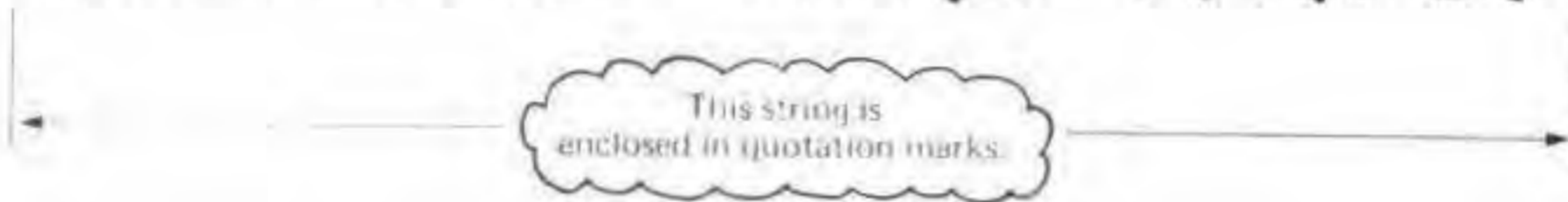
Enclose the following string in quotation marks.

RIDE THE THIRD WAVE!

“RIDE THE THIRD WAVE!”

REMEMBER: The quotation marks are not part of the string. They enclose the string.

“RIDE THE THIRD WAVE!”



2. Below are some string boxes.

A\$ B\$ C\$

D\$ ES

F\$ S\$

Z\$

- What is in box C\$? _____
- What is in box D\$? _____
- Which box contains seven “stars” (asterisks)? _____
- What is in box F\$? _____
- Which box has an important message for people who love their mothers? _____
- Which box has the name of the small city in which this book was written? _____

- (a) ABC
- (b) COMPUTERTOWN, USA!
- (c) S\$
- (d) Nothing. This box is "empty." Or perhaps it contains one or more invisible spaces. Later you will learn about both "empty strings" and strings consisting of one or more spaces.
- (e) Z\$
- (f) E\$ This is also the address of COMPUTERTOWN, USA!

3. The labels that identify string boxes (A\$, B\$, C\$, etc.) are called *string variables*. The string in a box is called a *string value*. The string in box A\$ is called the *value of A\$*; the string in box B\$ is called the *value of B\$*; the string in C\$ is called the *value of C\$*; and so on.

Below are three string boxes called A\$, N\$, and T\$.

A\$ MENLO PARK, CA 94025 N\$ GEORGE T\$ 415 323 6117

- (a) A\$, N\$, and T\$ are called _____
- (b) The value of A\$ is _____
- (c) The value of N\$ is _____
- (d) The string 415-323-6117 is the value of what string variable? _____

-
- (a) string variables
 - (b) MENLO PARK, CA 94025
 - (c) GEORGE
 - (d) T\$

4. In Chapter 4, you learned about number boxes and numeric variables.

These are numeric variables: A B C N Z
These are string variables: A\$ B\$ C\$ N\$ Z\$

What's the difference between numeric variables and string variables?

From the above evidence, you might say that a numeric variable is a letter of the alphabet, while a string variable is a letter followed by a dollar sign. OK. You might also have mentioned that the value of a numeric variable must be a number and that the value of a string variable must be a string.

Perhaps you remember that a numeric variable can be a letter followed by a digit or even a bunch of letters and digits. A string variable can also be a bunch of letters and/or digits, followed by a dollar sign. If so, the first character must be a letter.

These are numeric variables:	T1	ABC	KOLOR
These are string variables:	T1\$	ABC\$	KOLOR\$

Please recall, however, that you can't use variable names that begin with reserved words! (See frame 18, Chapter 4.) So you can't use TONE or TONE\$ (begins with TO), and you can't use GOAT or GOAT\$ (begins with GO). See Appendix E for a list of reserved words.

5. Here is how you tell the computer to stuff a string into a string box, or, more formally, how to assign a string to a string variable. To tell the computer to put the string GEORGE into string box N\$, do this:

You type: N\$ = "GEORGE"
It prints: OK



The Color Computer has put the string GEORGE into string box N\$. Remember how you did this.

N\$ = "GEORGE"

String variable

String

Now find out what is in string box N\$. Tell the computer to print the value of string variable N\$.

You type: PRINT N\$
 It prints: GEORGE
 OK

To tell the computer to print the value of string variable N\$, you type:

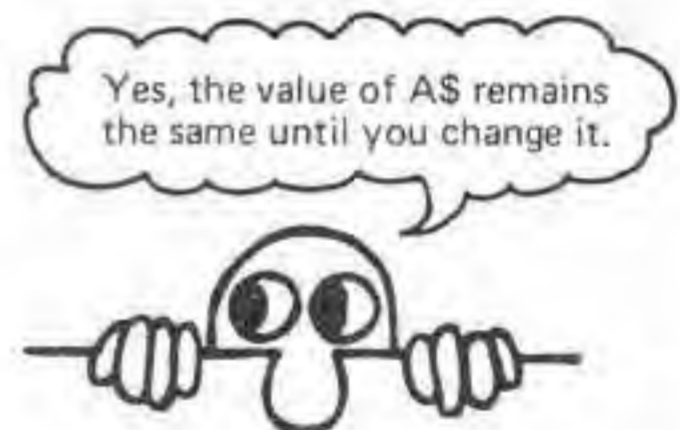
PRINT N\$

Your turn. Complete the following.

You type: A\$ = "THE FORCE IS WITH YOU"

- (a) It prints: _____
 You type: PRINT A\$
- (b) It prints: _____
- (c) _____
 You type: PRINT A\$
- (d) It prints: _____
- (e) _____

-
- (a) OK
 - (b) THE FORCE IS WITH YOU
 - (c) OK
 - (d) THE FORCE IS WITH YOU
 - (e) OK



IMPORTANT NOTICE: When you use this method to assign a value to a string variable, you must enclose the string in quotation marks.

A\$ = "PLEASE ENCLOSE ME IN QUOTATION MARKS"

6. How would you assign the string GREEN to the string variable C\$?

- (a) You type: _____

What would you then type to tell the computer to print the value of C\$?

- (b) You type: _____

-
- (a) C\$ = "GREEN"
 - (b) PRINT C\$
-

The screen might look like this:

```
C$ = "GREEN"
OK
PRINT C$
GREEN
OK
■
```

Note the difference between the two PRINT statements below.

WITH

PRINT "C\$"

Tells the computer to print the string C\$ which is enclosed in quotation marks.

WITHOUT

PRINT C\$

Tells the computer to print the value of the variable C\$.

EXPERIMENT! Try these, in top-to-bottom order.

```
A$ = "GREEN"
B$ = "SLEEVES"
PRINT A$, B$
PRINT A$; B$
PRINT A$ + B$
PRINT B$ " OF " A$
C$ = A$ + B$
PRINT C$
```

7. You can also use the INPUT statement to put strings into string boxes or, as some prefer to say, assign strings as values of string variables.

```

10 CLS
20 INPUT A$
30 PRINT A$
40 PRINT
50 GOTO 20
    
```

This program simply prints whatever you enter as the value of A\$.

Draw arrows to show the order in which statements are done by the computer.

```

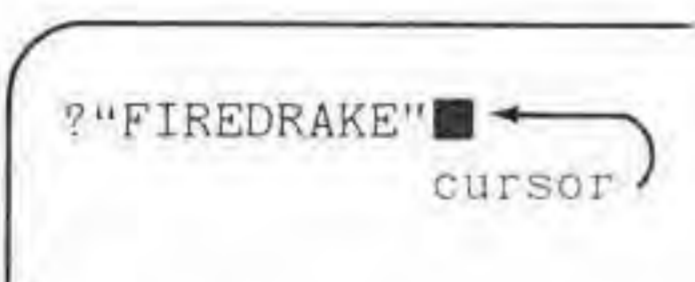
10 CLS
20 INPUT A$
30 PRINT A$
40 PRINT
50 GOTO 20
    
```

In following this program, the Color Computer always returns to the INPUT statement, prints a question mark, turns on the cursor, and waits for the next value of A\$.

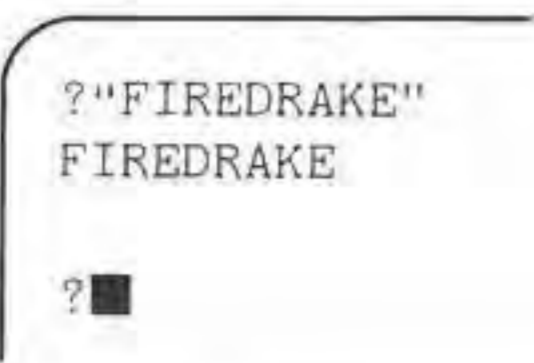
8. Store the program and RUN it. It begins like this:



Type the string FIREDRAKE, enclosed in quotation marks.



Press **ENTER**.



You typed a string (FIREDRAKE) enclosed in quotation marks. The computer put your string in string box A\$. It then printed the value of A\$.

After printing FIREDRAKE (the value of A\$), the computer printed a question mark and turned on the cursor. Why? _____

Look at the program in the preceding frame. After printing the value of A\$ (line 30), the computer prints a line space (line 40), then obeys the GOTO 20 statement in line 50 and goes to line 20, which is an INPUT statement.

9. Continue entering strings in response to question marks. Study these examples.

Huh? No quotation marks.
Seems to be OK.

Five spaces before FIREDRAKE.
It ignored the spaces.
Try quotation marks.
Aha! It printed the spaces.

Home, sweet home.
What happened?
Well, here's part of it.

Try quotation marks.
OK!

And so on—EXPERIMENT!

```
? "FIREDRAKE"
FIREDRAKE

? FIREDRAKE
FIREDRAKE

?   FIREDRAKE
FIREDRAKE

? "   FIREDRAKE"
   FIREDRAKE

? COMPUTERTOWN, USA!
EXTRA IGNORED
COMPUTERTOWN

? "COMPUTERTOWN, USA!"
COMPUTERTOWN, USA!

? ■
```

Our motto is:

When in doubt, use quotation marks when you enter a string as a value of a string variable in an INPUT statement.

Without quotation marks, the computer will:

ignore leading spaces but accept *trailing spaces*.

This string has trailing spaces: "FIRE Drake "

We couldn't show that on the screen because trailing spaces are . . . well . . . invisible.

Do funny things if a string has a comma.

Never mind. Plunge onward. Most, if not all, will be revealed. Or, *experiment* and find out for yourself what happens.

Oh, yes—when you are finished experimenting, press BREAK to stop the computer.

10. You can use the INPUT statement to put someone's name everywhere on the screen.

```
10 CLS
20 INPUT "YOUR NAME"; N$
30 PRINT N$:
40 GOTO 30
```

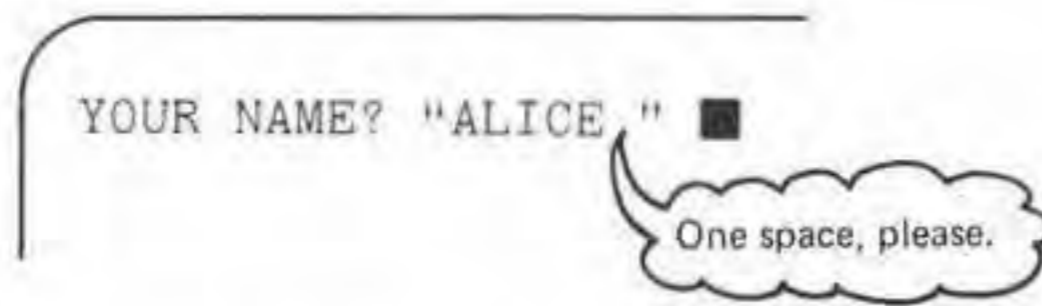
Draw arrows showing the order in which statements are obeyed by the computer.

```
10 CLS
  ↓
20 INPUT "YOUR NAME"; N$
  ↓
30 PRINT N$: ←
  ↓
40 GOTO 30 ←
```

11. As usual, to find out what a program does, enter it into memory, and RUN it. First, you see:

```
YOUR NAME?■
```

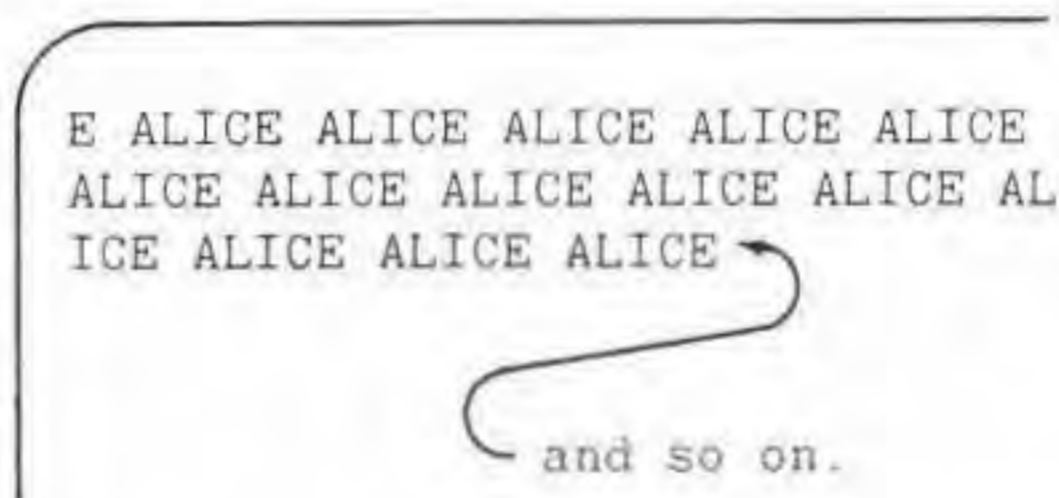
Alice, having recently returned from Wonderland, boldly tries it.



She types quotation marks, her name, one space, and quotation marks. She hasn't yet pressed ENTER.

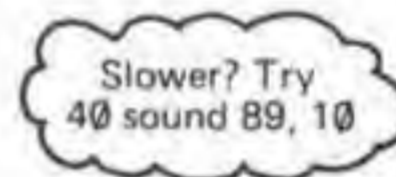
Then Alice presses ENTER. Oh! That boggles the eyes! In a blink, Alice's name fills the screen. Then—well, it's impossible to describe, because it is happening too quickly.

Press BREAK. The screen might look something like this:



Experiment! RUN the program again, or several times, if you wish. Then slow things down by adding a SOUND statement.

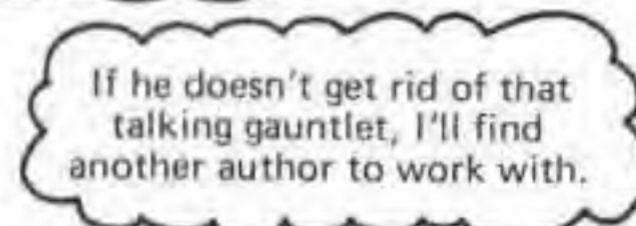
```
10 CLS
20 INPUT "YOUR NAME": N$
30 PRINT N$:
40 SOUND 89, 1
50 GOTO 30
```



Now RUN the program. The SOUND statement slows things down so that you can see what is happening. Try a bunch of things in response to YOUR NAME?

12. Instead of using a SOUND statement to slow down the computer, try our silent time delay, as shown below.

```
10 CLS
20 INPUT "YOUR NAME": N$
30 PRINT N$:
40 FOR K=1 TO 10
50 NEXT K
60 GOTO 30
```



Enter and RUN the program. Use whatever value of N\$ suits you. Try it a few times. Then change line 40 to one of the following.

```
40 FOR K = 1 TO 5
40 FOR K = 1 TO 20
40 FOR K = 1 TO 100
```

Experiment for a while, then read on.

13. FOR??? NEXT??? What do they do? OK, in response to popular demand, we will explain what is happening in lines 40 and 50.

<pre>40 FOR K=1 TO 10 50 NEXT K</pre>	←	This is called a <i>FOR-NEXT loop.</i>
---------------------------------------	---	---

The above FOR-NEXT loop simply tells the computer to count from 1 to 10.

Start here. Stop here.

```

      ↙           ↘
40 FOR K=1 TO 10
50 NEXT K

```

As the computer counts from 1 to 10, each counting number is put, momentarily, into number box K.

First the computer puts 1 into box K.
 Then it erases 1 and puts 2 into box K.
 Then it erases 2 and puts 3 into box K.
 Then it erases 3 and puts 4 into box K.

⋮

And so on. Eventually, it puts 10 into box K, then 11 . . . Oops! It was supposed to count to 10. So when 11 appears in box K, the computer quits counting and goes on to the statement following NEXT K.

This all happens very quickly, in a small fraction of a second!
 (a) How can you change line 40 so that the action on the screen occurs more slowly? _____
 (b) How can you change line 40 so that the action on the screen occurs more rapidly? _____

- (a) Use a number greater than 10. For example: 40 FOR K=1 TO 20
- (b) Use a number less than 10. For example: 40 FOR K=1 TO 5

14. A FOR-NEXT loop begins with a FOR statement and ends with a NEXT statement.

A numeric variable must follow the word FOR:

```
40 FOR K=1 TO 10
      ↑
  numeric variable
```

The same numeric variable follows the word NEXT:

```
50 NEXT K
      ↑
  numeric variable
```

Hmmm . . . one more time.

```
40 FOR K=1 TO 10
50 NEXT K
```

← same numeric variable

Don't overwork the numeric variable K. Any numeric variable may be used. These are OK FOR-NEXT loops.

```
40 FOR B=1 TO 100   40 FOR N=1 TO 73
50 NEXT B           50 NEXT N
```

But these are *not* OK:

- (a) 40 FOR X=1 TO 100
50 NEXT Y
- (b) 40 FOR K\$=1 TO 20
50 NEXT K\$

What is wrong with (a)? _____

What is wrong with (b)? _____

(a) The variable following NEXT is different from the variable following FOR. These must be the *same* variable.

(b) Oops! K\$ is a *string* variable. Use only numeric variables in FOR and NEXT statements.

15. You can, of course, put the entire FOR-NEXT loop on one line, as shown below in line 40.

```

10 CLS
20 INPUT "YOUR NAME": N$
30 PRINT N$:
40 FOR K=1 TO 10: NEXT K
50 GOTO 30
    
```

In this program, line 40 contains two statements. In a line that contains two statements, what character is used between the statements?_____

a colon (:)

```

40 FOR K=1 TO 10: NEXT K
    
```

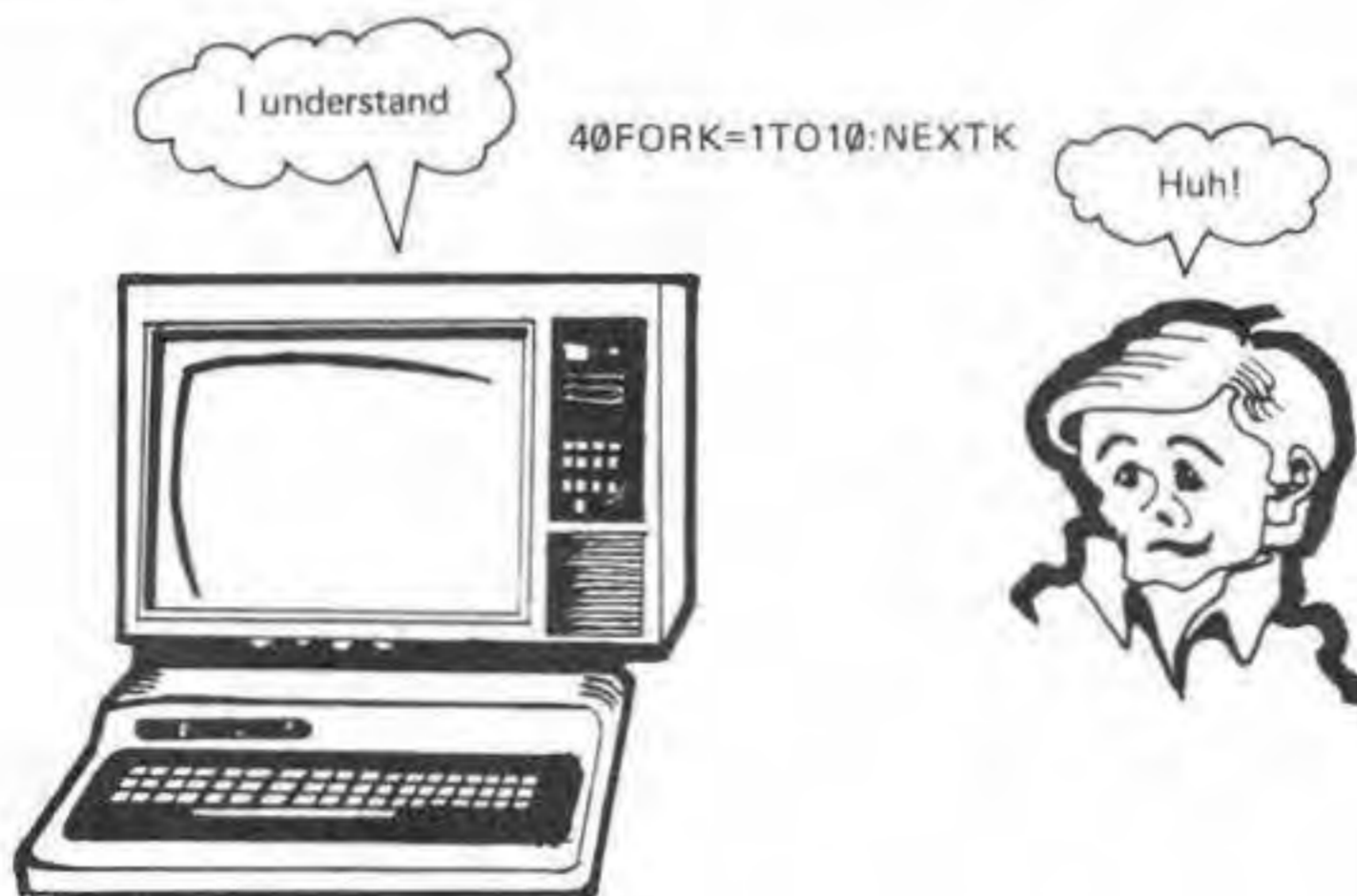
⏟
↑
⏟
 statement #1 colon statement #2

We usually put the colon immediately to the right of the first statement, then a space, then the second statement. You can do it differently, however, as shown by either of the following.

```

40 FOR K=1 TO 10 : NEXT K
40 FOR K=1 TO 10:NEXT K
    
```

The above are all the same to the computer. Use spaces to make lines readable by *people*.



16. OK, we confess: there is much more about FOR-NEXT loops that we haven't told you. So, here we go!

A FOR-NEXT loop *begins* with a FOR statement and *ends* with a NEXT statement. A FOR-NEXT loop may also have one or more statements *between* the FOR statement and the NEXT statement.

```

10 CLS
20 FOR T=1 TO 255
30 SOUND T, 1
40 NEXT T
    
```

This FOR-NEXT loop
has *three* statements.

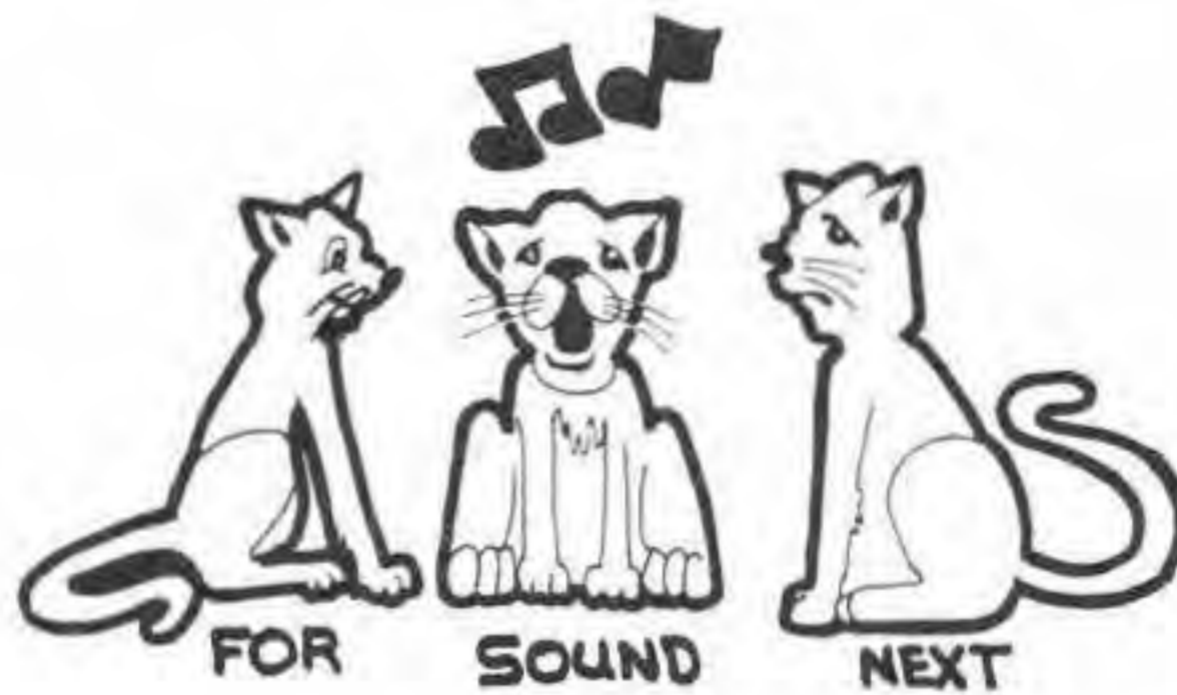
- (a) In the above FOR-NEXT loop (lines 20, 30, and 40), what statement is *between* the FOR statement and the NEXT statement? _____
- (b) We could have written the above program like this:

```

10 CLS
20 FOR T=1 TO 255: SOUND T, 1: NEXT T
    
```

In the above FOR-NEXT loop (line 20), what statement is *between* the FOR statement and the NEXT statement? _____

- (a) 30 SOUND T, 1
- (b) SOUND T, 1



17. Answer these questions about the program in the preceding frame.
- (a) What variable is used in the FOR statement and the NEXT statement? _____
 - (b) What values will this variable have, one after the other, when the computer runs the program? _____
 - (c) Does the FOR-NEXT variable appear elsewhere in the program? _____
If yes, where? _____
 - (d) What do you think will happen if you RUN the program? _____

- (a) T
- (b) 1,2,3, and so on, up to 255 . . . thank you, line 20.
- (c) Yes. In the SOUND statement: SOUND T, 1
- (d) The computer will make 255 very short sounds, starting with a low tone (T=1) and ending with a high, scratchy sound (T=255).

18. Your turn. You complete the following program, using a FOR-NEXT loop, to tell the computer to zip through the screen colors.

```
10 CLS
20 FOR C=__TO__
30   CLS ____
40 NEXT ____
```

You might have done it

<u>THIS WAY</u>		<u>THIS WAY</u>
10 CLS	or	10 CLS
20 FOR C=1 TO 8		20 FOR C=0 TO 8
30 CLS C		30 CLS C
40 NEXT C		40 NEXT C

Alas! When you ran the program, the colors flashed by so rapidly that you had no time to savor them. So—cleverly!—brilliantly!—you modified the program, as follows.

```
10 CLS
20 FOR C=0 TO 8
30   CLS C
40   FOR K=1 TO 200: NEXT K Aha! A time delay.
50 NEXT C
```

19. Note how we write our FOR-NEXT loops. We *indent* the statements on the inside of the loop.

```
10 CLS
20 FOR T=1 TO 255  beginning of loop
30   SOUND T, 1   inside of loop, (Indented)
40 NEXT T         end of loop
```

We indent statements inside the loop *two* spaces. We do this simply to make it easier for *people* to read the program. The Color Computer ignores the extra spaces, but people find programs easier to read and understand if we include the spaces.

Just for fun, try these programs:

```
10 CLS

20 FOR T=1 TO 255
30   SOUND T, 1
40 NEXT T

50 GOTO 20
```

```
10 CLS

20 FOR C=0 TO 8
30   CLS C
40   FOR K=1 TO 200: NEXT K
50 NEXT

60 GOTO 10
```

20. Why not get your name involved with some of this new stuff?

```
100 REM ** 1ST GRAND FINALE, CH. 5
110 CLS

200 REM ** FOR WHOM?
210 INPUT "YOUR NAME"; N$

300 REM ** CRESCENDO FOR N$
310 CLS
320 FOR T=1 TO 255
330   PRINT N$:
340   SOUND T, 1
350 NEXT T

400 REM ** LONG TIME DELAY
410 FOR K=1 TO 2500: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 110
```



VARIATION: change line 510 to: 510 GOTO 310
or change line 510 to: 510 GOTO 320

Note the programming style.

- Each “idea” begins with a REMark statement, numbered 100, 200, 300, and so on. We call these *block 100*, *block 200*, and so on.
- We indent statements on the inside of FOR-NEXT loops by two spaces. Why two spaces? Well, we decided, arbitrarily and capriciously, to indent by two spaces. The computer doesn’t care how many spaces you use. You prefer three spaces? Use three spaces.

To save time, you may omit the REMark statements when you enter a program into the computer. Why? Well, the computer ignores REMark statements. Beware! Don't GOTO a REMark statement. If you do this, you must enter the REMark statement as part of the program.

Oh, yes—one more thing: when you LIST a program, the computer does not respect the "indenting". It removes extra spaces between the line number and the first character of a statement.

21. You know how to do a crescendo; here is how to do a descendo. *Descendo*? That's probably not a word. Anyhow, try this program to see what we mean:

```
10 CLS
20 FOR T=255 TO 1 STEP -1
30  SOUND T, 1
40 NEXT T
```



If you RUN this program, you will hear the computer play 255 tones, beginning with a high, scratchy tone (Tone #255) and ending with a low, mellow tone (Tone #1).

255, 254, 253, 252, 251, . . . 5, 4, 3, 2, 1

Why? Look at the FOR statement.

Start here. End here. Aha! Count backwards.

```
20 FOR T=255 to 1 STEP -1
```

Something new has been added. STEP -1 tells the computer to change T by -1 (called "minus one"). This causes the computer to count backwards. The FOR statement begins at 255 and counts backwards (or down) to 1.

To make the computer count backwards, you must use STEP -1. If you don't believe it, RUN the following program.

```
10 CLS
20 FOR T=255 TO 1
30  SOUND T, 1
40 NEXT T
```

22. If you wish to leap ahead by twos, STEP 2 is what you choose.

```
10 CLS
20 FOR N=1 TO 11 STEP 2
30 PRINT N
40 NEXT N
```



But zero to eleven, in STEPs of twos
Gives even numbers only, and eleven eschews.

```
10 CLS
20 FOR N=0 TO 11 STEP 2
30 PRINT N
40 NEXT N
```

The following program with STEPs so trinity.
Gives only a hint of enumerable infinity.

```
10 CLS
20 FOR A=0 TO 3000 STEP 3
30 PRINT A
40 NEXT A
```

What? You want still more?
You do zero to twenty, in STEPs of 4,

```
10 CLS
20 FOR X=0 TO 40 STEP 4
30 PRINT X
40 NEXT X
```

We used X as our FOR-NEXT variable. You probably used something else.

23. Experiment! Find out more about FOR-NEXT by using the following program.

```
100 REM ** COUNTING PROGRAM
110 CLS

200 REM ** DIALOG WITH USER
210 PRINT "I WILL COUNT FOR YOU."
220 PRINT
230 INPUT "WHERE SHALL I START"; A
240 PRINT
250 INPUT "WHERE SHALL I STOP"; Z

300 REM ** COUNT FROM A TO Z
310 CLS
320 FOR K=A TO Z
330   PRINT K,
340 NEXT K

400 REM ** THAT'S ALL
410 END
```

Try these values of A and Z.

A	Z	YOU WILL SEE THESE NUMBERS			
1	3	1	2	3	
2	5	2	3	4	5
.5	3	.5	1.5	2.5	
7	7	7			
1	0	1	(only)	} Hmmm . . . seems as if it won't count backwards.	
2	1	2	(only)		
10	0	10	(only)		

Why won't the computer count backwards? _____

There is no STEP -1 in line 320.

24. Modify the program in the preceding frame so the computer asks for the STEP size, then uses the STEP size in counting from A to Z. Show only your changes and additions below.

Add these statements: 260 PRINT
 270 INPUT "STEP SIZE"; S

Change line 320: 320 FOR K=A TO Z STEP S

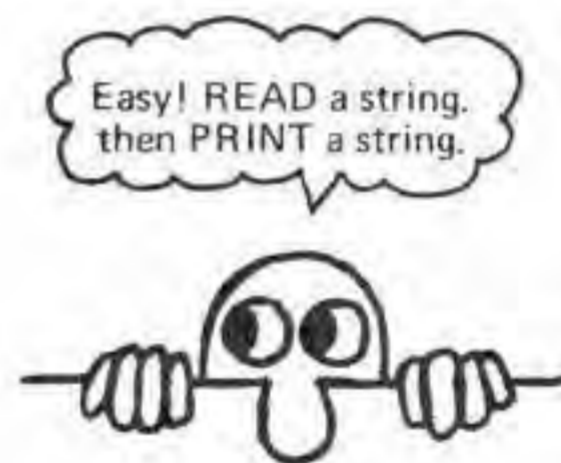
EXPERIMENT!

25. You can use READ and DATA statements to stuff strings into string boxes. First try this program.

```

10 CLS
20 READ N$
30 PRINT N$
40 GOTO 20
50 DATA DO, RE, MI, FA
60 DATA SOL, LA, TI, DO

```



The statement:

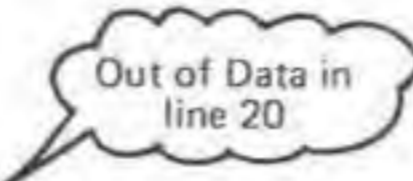
```
20 READ N$
```

tells the computer to read one string from a DATA list and put the string into box N\$ (assign the string as the value of N\$). Every time line 20 is executed, the next string in the DATA list is read and assigned as the value of N\$.

This is how the screen looks when you RUN the program.

```

DO
RE
MI
FA
SQL
LA
TI
DO
?OD ERROR IN 20
OK
■
    
```



- (a) How many strings are in the DATA statements (lines 50 and 60)? _____
- (b) Were these strings all printed on the screen? _____
- (c) Why do you suppose the computer printed an ?OD ERROR IN 20? _____

(a) 8; (b) yes; (c) It printed all the string, then tried to print yet another. Alas, there were no more strings to be printed, so the computer printed an error message. That's OK, since it was finished doing what we wanted it to do.

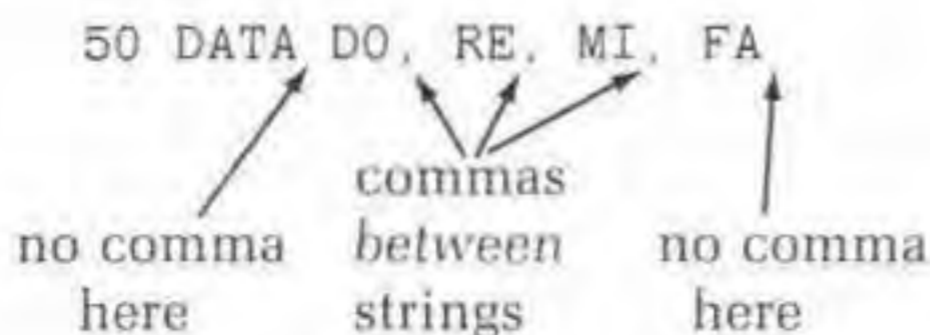
26. Look at the DATA statements in lines 50 and 60.

```

50 DATA DO, RE, MI, FA
60 DATA SOL, LA, TI, DO
    
```

The two DATA statements contain eight strings. What punctuation symbol appears *between* adjacent strings? _____

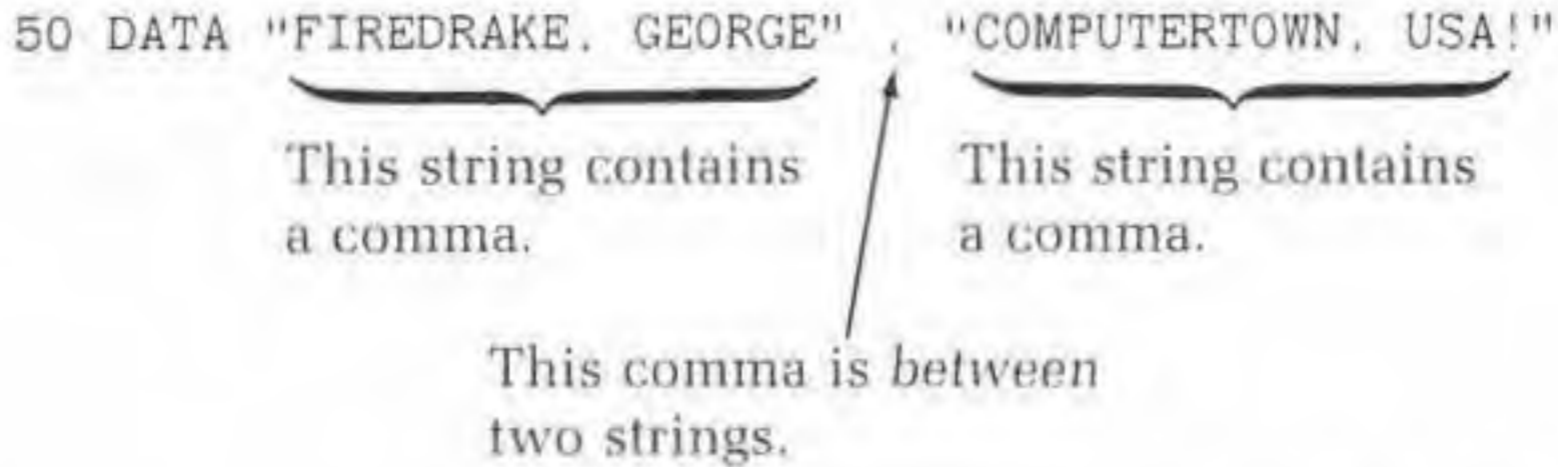
comma; for example,



IMPORTANT NOTICE! If a string contains a comma, the string must be enclosed in quotation marks.

LIKE THIS:

```
50 DATA "FIREDRAKE, GEORGE" , "COMPUTERTOWN, USA!"
```



27. Hmm . . . who said, "It all begins with do, re, mi?" Try this program—see what you hear and hear what you see.

```
100 REM ** DO RE MI PROGRAM
110 REM ** INSPIRED BY HERB MOORE

200 REM ** READ NOTE AND TONE NUMBER
210 READ N$, T

300 REM ** SHOW AND TELL
310 CLS
320 PRINT N$
330 SOUND T, 10

400 REM ** DO IT AGAIN
410 GOTO 210

900 REM ** NOTES AND TONE NUMBERS
910 DATA DO, 89, RE, 108, MI, 125, FA, 133
920 DATA SOL, 147, LA, 159, TI, 170, DO, 176
```

The READ statement (line 210) tells the computer to read a value for a string variable and a value for a numeric variable.

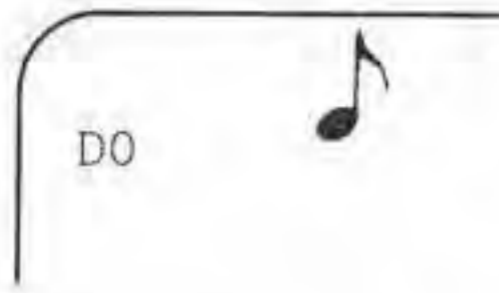
```
210 READ N$, T
```

string	numeric
variable	variable

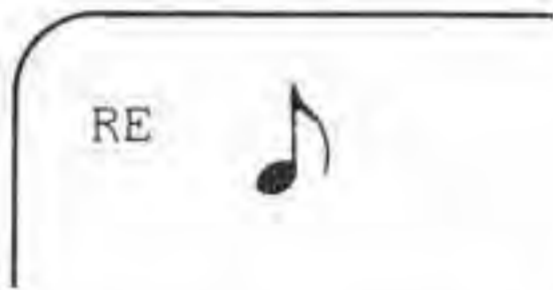
- (a) What is the first string value read into N\$? _____
- (b) What is the first numeric value read into T? _____

(a) DO; (b) 89

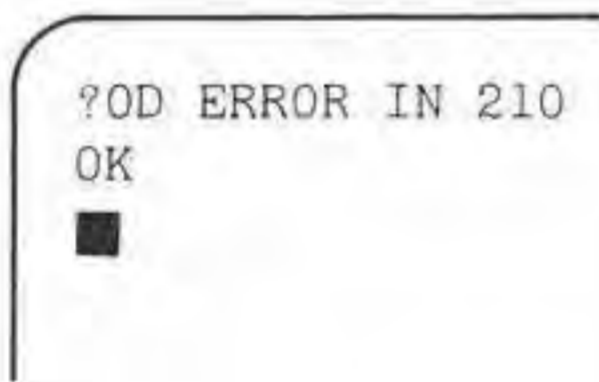
28. RUN the program. First you see DO on the screen and hear the musical tone for DO in the scale of C.



Then you see RE and the musical tone for RE.



And so on. Eventually, the computer runs out of music and you see:



Mix up the DO, RE, MIs. Write your own DATA statements—as many as you want. Change the duration in line 330. Make it longer. Then close your eyes. When you hear the sound, guess. Is it DO, RE, MI . . . ? Open your eyes and look at the screen. Are you correct?

If you want to learn your DO, RE, MIs (in the scale of C), try this variation:

```

100 REM ** DO RE MI PROGRAM.
200 REM ** READ NOTE AND TONE NUMBER
210 READ N$, T
300 REM ** SHOW AND TELL
310 CLS
320 SOUND T, 20      Or longer.
330 PRINT N$
400 REM ** TIME DELAY
410 Z = 460
420 FOR K=1 TO Z: NEXT K
500 REM ** DO IT AGAIN
510 GOTO 210
900 REM ** NOTES AND TONE NUMBERS
910 DATA MI, 125, TI, 170, TI, 170, DO, 89
920 DATA LA, 159, DO, 176, SOL, 147, FA, 133
930 DATA MI, 125, RE, 108, FA, 133, SOL, 147
940 DATA TI, 170, DO, 176, MI, 125, LA, 159
950 DATA FA, 133, DO, 89, TI, 170, TI, 170
and so on.

```

Now you can keep your eyes open and watch the screen. First you hear the note. Guess—is it DO, RE, MI . . . ? Then you see the answer.

Want more time to guess? Change the duration in line 320. The time delay (lines 410 and 420) is about one second. Change line 410 to make it longer or shorter.

Self-Test

Congratulations! You have completed another chapter. Undoubtedly, you have the stringth (Oops!) to twine your way through this Self-Test.

1. What is a string?_____
2. What is a "string box"?_____
3. What is a string variable?_____
4. What is the value of a string variable?_____
5. Complete the following.

You type: Z\$ = "ZZZZZ"
 It prints: OK
 You type: PRINT Z\$

It prints: _____

ZZZZZ



6. Complete the following. (Go ahead. Guess!)

You type: A\$ = "DO RE MI"
It prints: OK
You type: B\$ = A\$
It prints: OK
You type: PRINT A\$

It prints: _____

You type: PRINT B\$

It prints: _____

7. Complete the following:

You type: M\$ = "HAPPY BIRTHDAY, MOTHER"
It prints: OK
You type: PRINT "M\$"

It prints: _____

8. Examine the following program.

```
100 REM ** NAME, TONE, AND COLOR
200 REM ** GET INFORMATION
210 CLS
220 INPUT "YOUR NAME": N$
230 INPUT "TONE NUMBER (1 TO 255)": T
240 INPUT "SCREEN COLOR (1 TO 8)": C

300 REM ** PRINT N$ ON COLOR C WITH SOUND T
310 CLS C
320 PRINT N$:
330 SOUND T, 1
340 GOTO 320
```

Suppose you RUN the program and supply the following information.

```
YOUR NAME? "SPARROWHAWK"
TONE NUMBER (1 TO 255)? 89
SCREEN COLOR (1 TO 8)? 3 ■ cursor
```

Now you press ENTER. What will happen? _____

9. What will happen if you RUN the following program? Complete the REM statements and then describe what happens when you RUN the program.

```

100 REM ** MYSTERY PROGRAM
110 CLS

200 REM ** _____
210 SOUND 1.1

300 REM ** _____
310 FOR K = 1 TO 450: NEXT K

400 REM ** _____
410 GOTO 210

```

If you wish, change line 110 to CLS 2 or CLS 3 or _____ whatever.

10. What musical device is suggested by the above program and the following picture? _____



11. The following table shows musical notes and tone numbers for the scale of C. The scale is shown both as DO, RE, MI, and so on, and as letter notes C, D, E, and so on.

	DO	RE	MI	FA	SOL	LA	TI	DO
Notes:	C	D	E	F	G	A	B	C
Tone Numbers:	89	108	125	133	147	159	170	176

In frame 28, you saw the following DATA statements, which use DO, RE, MI, and so on.

```

910 DATA MI, 125, TI, 170, TI, 170, DO, 89
920 DATA LA, 159, DO, 176, SOL, 147, FA, 133
930 DATA MI, 125, RE, 108, FA, 133, SOL, 147
940 DATA TI, 170, DO, 176, MI, 125, LA, 159
950 DATA FA, 133, DO, 89, TI, 170, TI, 170
    
```

Rewrite the above DATA statements using letters for notes instead of DO, RE, MI . . .

```

910 _____
920 _____
930 _____
940 _____
950 _____
    
```

RUN the program in frame 28, using your DATA statements.

12. Write a program to tell the computer to “sound off” and display the seconds, 1 to 60, in the upper left corner of the screen.



As each number appears, you should hear a very short sound (use SOUND 1,1). After the number 60 appears, the computer should begin again at 1.

13. For each of the following FOR statements, write the sequence of values that the variable A will take on. We have already done the first one.

	FOR STATEMENT	SEQUENCE OF VALUES
(a)	FOR A = 1 TO 7	1,2,3,4,5,6,7
(b)	FOR A = 2 TO 6	_____
(c)	FOR A = 1 TO 7 STEP 2	_____
(d)	FOR A = 0 TO 7 STEP 2	_____
(e)	FOR A = 0 TO 4 STEP 5	_____
(f)	FOR E = 1 TO -1 STEP -1	_____

The next two are tricky! It's OK to guess.

- | | | |
|-----|-----------------------|-------|
| (g) | FOR A = 5 TO 3 | _____ |
| (h) | FOR A = 1 TO 2 STEP 0 | _____ |

14. For each of the following programs, pretend that you are the computer. Show what will be on your screen when the program is run.

```
(a) 100 CLS
     110 FOR B = 0 TO 10
     120   PRINT 10 - B
     130 NEXT B
     140 PRINT "BLASTOFF!!!"
```

```
(b) 100 CLS
     110 FOR L = 0 TO 15 STEP 3
     120   PRINT L
     130 NEXT L
```

```
(c) 100 CLS
     110 FOR C = 10 TO 0 STEP -1
     120   PRINT C
     130 NEXT C
     140 PRINT "BLASTOFF!!!"
```

```
(d) 100 FOR C = 10 TO 0 STEP -1
     110 CLS
     120   PRINT C
     130 NEXT C
     140 PRINT "BLASTOFF!!!"
```

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. A bunch of keyboard characters, typed one after the other. Although sometimes you don't have to, when in doubt enclose a string in quotation marks. (frame 1)
2. A place in the computer's memory (RAM) that can hold, or store, a string. (frames 1,2)
3. The name of a string box. A string variable can be a letter followed by a dollar sign (A\$), a letter and a digit followed by a dollar sign (N3\$), or a letter followed by letters and/or digits and a dollar sign (AB\$, S123\$, CITY\$). BEWARE! The computer looks at only the first two characters. It will be confused if you use ABC\$ and ABD\$ in the same program. And, if your string variable causes an SN ERROR, perhaps you used a reserved word. (frames 3,4)

4. The value of a string variable can be a string. (frames 3.5)
 5. It prints: ZZZZZ
OK (frames 5.6)
 6. It prints: DO RE MI
OK
The statement B\$ = A\$ tells the computer to copy the value of A\$ into B\$.
It prints: DO RE MI
OK (frames 5.6)
 7. It prints: M\$
OK
Did we fool you? Note the quotation marks surrounding M\$ in the PRINT statement. If you want the computer to print the value of string variable M\$, type PRINT M\$. (frame 6)
 8. The screen will become blue. Then the string SPARROWHAWK will be printed, again and again. Each copy of SPARROWHAWK will be accompanied by a short tone. As SPARROWHAWK take over the screen, the scene will become green. (frames 10, 11, 20)
 9. 200 REM ** PLAY A SHORT, LOW TONE
300 REM ** TIME DELAY
400 REM ** DO IT AGAIN
You will hear a short, low tone about once every second (frames 12, 13, 20)
 10. Here are some hints. You see a gnome who lives in a very large city. A very large city is also called a metropolis and sometimes abbreviated "metro." Got it?
 11. 910 DATA E, 125, B, 170, B, 170, C, 89
920 DATA A, 159, C, 176, G, 147, F, 133
930 DATA E, 125, D, 108, F, 133, G, 147
940 DATA B, 170, C, 176, E, 125, A, 159
950 DATA F, 133, C, 89, B, 170, B, 170
RUN the program to see what happens. (frames 26, 28)
 - 12, 13, 14. No solutions. We leave these for you to puzzle over.
-

CHAPTER SIX

Skipping Round the Screen

In this chapter, you will learn more about how to control and use the screen. You will learn how to tell the Color Computer to print information anywhere on the screen, blink it on and off, move it around.

When you finish this chapter, you will be able to:

- Position information on the screen, using the PRINT @ statement;
- Blink information on and off anywhere on the screen;
- Write programs to do simple animation—that is, cause things to move from place to place on the screen;
- Understand and use screen printing positions, numbered 0 to 511;
- Interpret screen printing positions as rows numbered 0 to 15, and columns numbered 0 to 31.

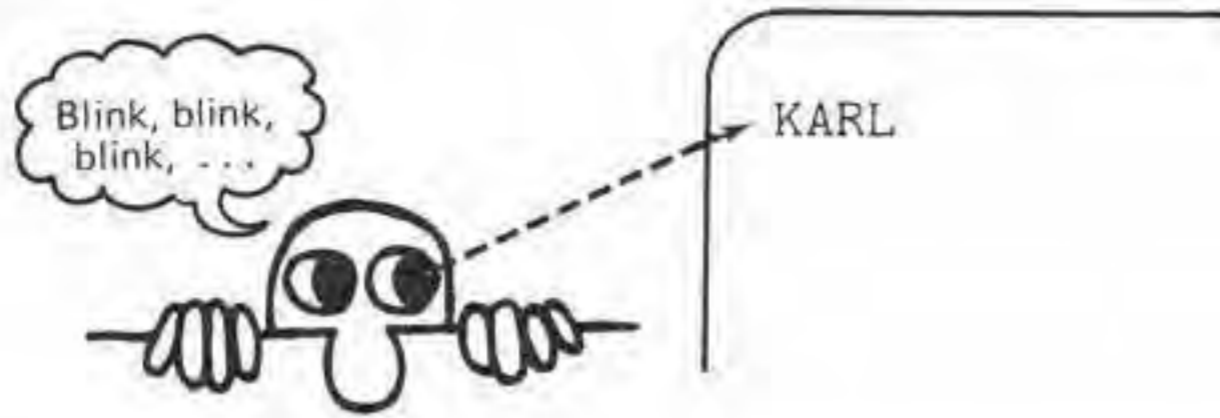
1. The following program causes Karl's name to blink on, blink off, on, off, on, off, on . . . and so on, until someone presses **BREAK**.

```
100 REM ** NAME BLINKER
200 REM ** NAME IS ON-SCREEN
210 CLS
220 PRINT "KARL"
230 FOR K=1 TO 500: NEXT K

300 REM ** SCREEN IS BLANK
310 CLS
320 FOR K=1 TO 500: NEXT K

400 REM ** DO IT AGAIN
410 GOTO 210
```

RUN the program.



- (a) When Karl's name is on the screen, what color is the screen? _____
- (b) When Karl's name is *not* on the screen, what color is the screen? _____
- (c) How would you change the program so that, when Karl's name is *not* on the screen, the screen is orange? _____
- (d) How would you change the program so that, when Karl's name is on the screen, most of the screen is blue?
- (e) How would you change the program so the computer will blink Judy's name on and off? _____

- (a) green; (b) green; (c) 310 CLS 8;
 (d) 210 CLS 3; (e) 220 PRINT "JUDY"

2. Make it easier to change the name.

```

100 REM ** NAME BLINKER
200 REM ** GET NAME
210 CLS
220 INPUT "WHAT IS YOUR NAME"; N$

300 REM ** NAME IS ON-SCREEN
310 CLS
320 PRINT N$
330 FOR K=1 TO 500: NEXT K

400 REM ** SCREEN IS BLANK
410 CLS
420 FOR K=1 TO 500: NEXT K

500 REM ** BLINK IT AGAIN
510 GOTO 310
  
```


- (a) Suppose you change line 330 to: 330 FOR K=1 TO 1000: NEXT K
Will the name be on-screen for a shorter time or a longer time? _____
- (b) Suppose you change line 330 to: 330 FOR K=1 TO 200: NEXT K
Will the name not be on-screen (screen clear) for a shorter or longer time? _____

- (a) Longer. It takes the computer longer to count to 1000 than to 500.
(b) Shorter. It takes the computer less time to count to 200 than to 500.

3. EXPERIMENT!

- Replace either or both time delays (lines 330 and 420) with SOUND statements.
- Change lines 210, 310, and 410. Use colors of your choice.
- Try other blinking names or messages, perhaps one of the following.

TOMORROW IS YOUR MOTHER'S BIRTHDAY,
 MAN THE PUMPS - THE BOAT IS SINKING!
 MAY THE FORCE BE WITH YOU.
 LIVE LONG AND PROSPER.
 IF DRAGONS CAN FLY, SO CAN I.
 _____ IS THE GREATEST.

↑
Put your name here!



Try different combinations of messages, time delays, sounds and colors to get effects that *you* like.

- Change line 320 as follows:

```
320 PRINT N$;  
      ↑  
    semicolon
```

Try line 320 with and without the semicolon for colors other than green.

4. Imagine a Color Computer with a giant TV screen. It's the big game of the season and the Color Computer is firing up the Rooter Club—building energy to help your team win the game.

```
100 REM**GO TEAM GO!  
  
200 REM**'GO' ON A BLUE SCREEN  
210 CLS 3: PRINT "GO":  
220 FOR K=1 TO 500: NEXT K  
  
300 REM**'TEAM' ON ORANGE SCREEN  
310 CLS 8: PRINT "TEAM":  
320 FOR K=1 TO 500: NEXT K  
  
400 REM**'GO!' ON MAGENTA SCREEN  
410 CLS 7: PRINT "GO!":  
420 FOR K = 1 TO 1000: NEXT K  
  
500 REM**KEEP IT GOING  
510 GOTO 110
```

Check the time delays. Are they all the same length of time? _____

No. The delay in line 420 is twice as long as the delays in lines 220 and 320. The exhortation to your loyal fans goes:
GO TEAM GO . . . OH!

5. Three times delays in that last program. Aha! A splendid opportunity to introduce . . . (fanfare!) . . . *subroutines*. Write the time delay *once* as a subroutine, then use it as often as needed.

The following program has a *time delay subroutine* beginning at line 900. This subroutine is used, or *called*, in lines 220, 320, 420, and 430.

```

100 REM**GO TEAM GO!
200 REM**'GO' ON A BLUE SCREEN
210 CLS 3: PRINT "GO";
220 GOSUB 910 ←———— Use time delay subroutine.

300 REM**'TEAM' ON ORANGE SCREEN
310 CLS 8: PRINT "TEAM";
320 GOSUB 910 ←———— Use time delay subroutine.

400 REM**'GO!' ON MAGENTA SCREEN
410 CLS 7: PRINT "GO!";
420 GOSUB 910 }
430 GOSUB 910 } ←———— Use time delay subroutine
                          twice for longer delay.

500 REM**KEEP IT GOING
510 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 FOR K=1 TO 500: NEXT K
920 RETURN

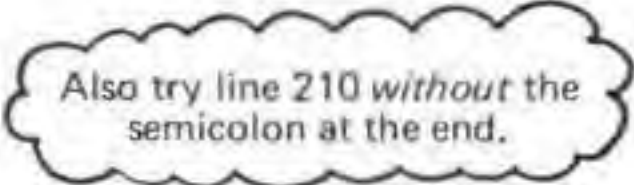
```

The time delay subroutine is called (used) by the GOSUB 910 statement in lines 220, 320, 420, and 430. Obedient as always, the Color Computer goes to line 910, does it, then moves on to line 920. Aha! Line 920 says RETURN. So, the computer RETURNS to the statement following the GOSUB 910, and continues from there. Clever!

A subroutine is a helper. You *call* it when you need it. It does what it is designed to do, then RETURNS to the statement following the GOSUB that called it.

6. Your turn. Complete the following program to blink George's name on, off, on, off, on . . . and so on.

```
100 REM**NAME BLINKER
200 REM**BLINK NAME ON
210 CLS 4: PRINT "GEORGE";
220 _____
300 REM**BLINK NAME OFF
310 CLS 5
320 _____
400 REM**DO IT AGAIN
410 GOTO 210
700 REM**TIME DELAY SUBROUTINE
710 FOR K=1 TO 500: NEXT K
720 RETURN
```

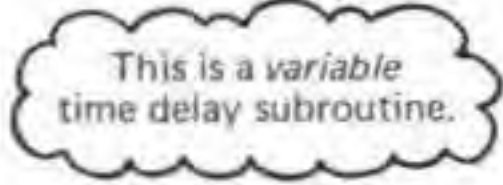


Also try line 210 *without* the semicolon at the end.

```
-----
220 GOSUB 710
320 GOSUB 710
```

7. How about a variable delay? Try it with Ann's name.

```
100 REM**NAME BLINKER
200 REM**BLINK NAME ON
210 CLS 1: PRINT "ANN";
220 GOSUB 810
300 REM**BLINK NAME OFF
310 CLS 2
320 GOSUB 810
400 REM**DO IT AGAIN
410 GOTO 210
800 REM**TIME DELAY SUBROUTINE
810 Z = 500
820 FOR K=1 TO Z: NEXT K
830 RETURN
```



This is a *variable* time delay subroutine.

The amount of delay (how far the computer counts) is controlled by the value of Z in line 810. To speed up or slow down the blinking, change the value of Z.

```
Faster: 810 Z = 200
Slower: 810 Z = 1000
```

The subroutine is called twice, once in line _____
and once in line _____.

220; 320

8. Here is still another way to do time delay subroutines. Set the value of Z, then call the subroutine.

```

100 REM**NAME BLINKER
200 REM**NAME ON
210 CLS 1: PRINT "URSULA";
220 Z = 1000 ←———— This much delay.
230 GOSUB 910

300 REM**NAME OFF
310 CLS 6
320 Z = 500 ←———— This much delay.
330 GOSUB 910

400 REM**GO AROUND AGAIN
410 GOTO 220

900 REM**TIME DELAY SUBROUTINE
910 FOR K=1 TO Z: NEXT K
920 RETURN

```

URSULA will be on-screen about twice as long as off-screen (Z=1000 in line 220; Z=500 in line 320). Change the program so things will happen more rapidly with URSULA on-screen about three times as long as off-screen. Which two lines will you change?

(a) Lines _____ and _____.

How might you change them?

(b) Change line _____ to _____.

(c) Change line _____ to _____.

(a) 220; 320

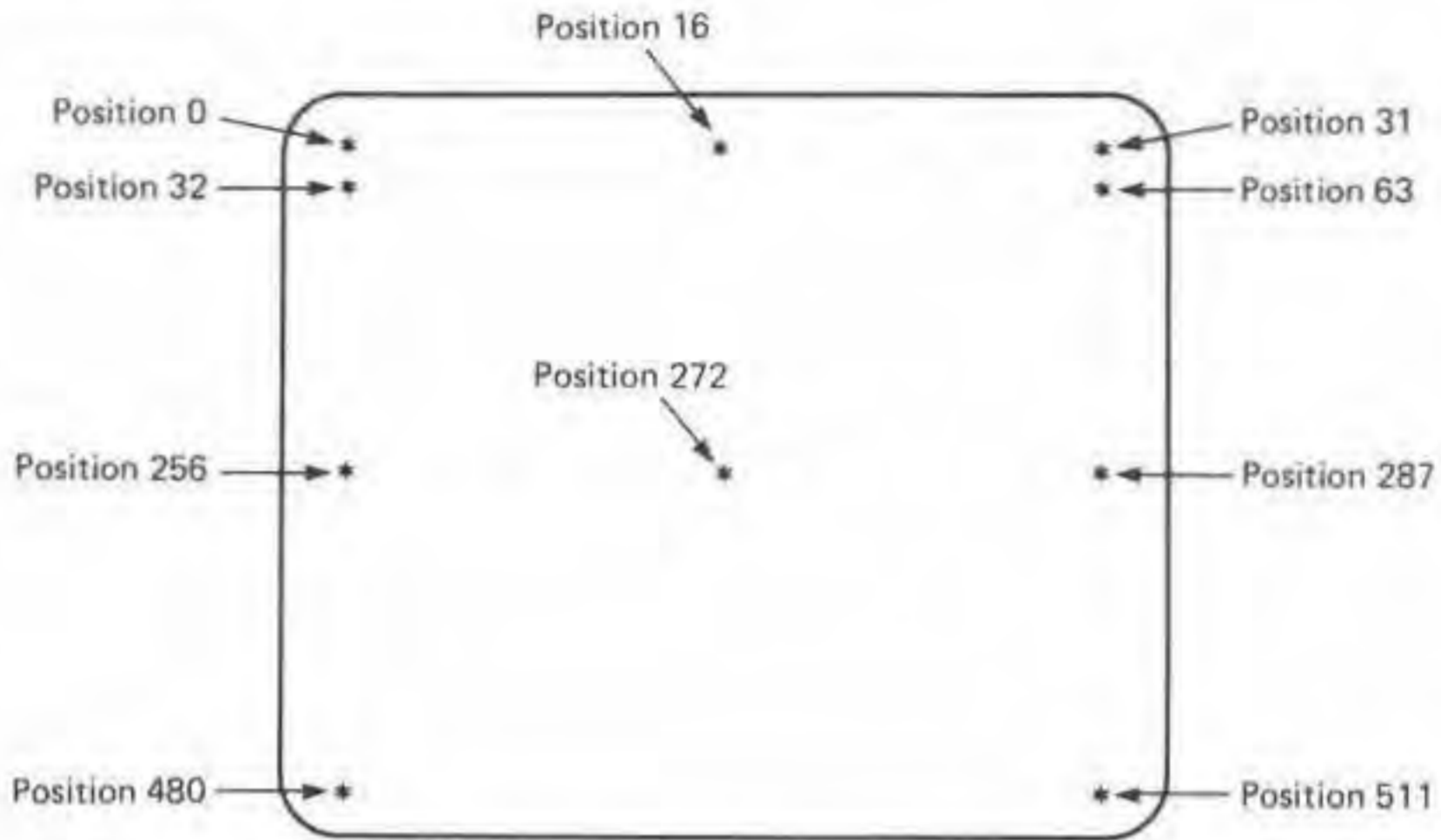
(b) 220; 220 Z = 600

(c) 320; 320 Z = 200

You may have used different values of Z in lines 220 and 320. That's OK, as long as the value of Z in line 220 is about three times as big as the value of Z in line 320.

9. Everything seems to happen in the upper left corner of the screen. Well, let's not wear out that corner. Instead, make things happen elsewhere on the screen.

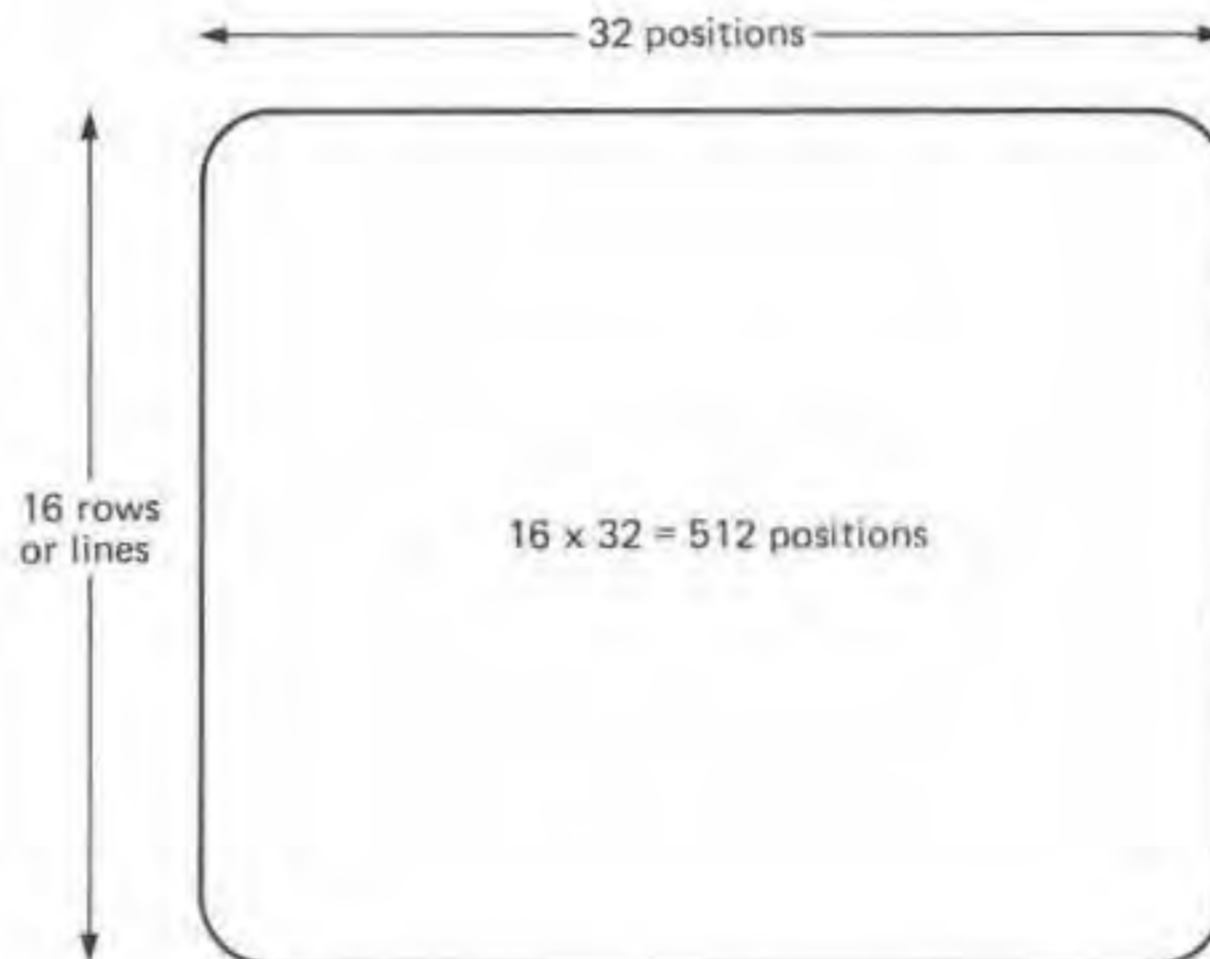
The Color Computer can print in any one of 512 *print positions* on the screen, from print position zero (0) in the upper left corner to print position 511 in the bottom right corner. Each print position can hold one character of information.



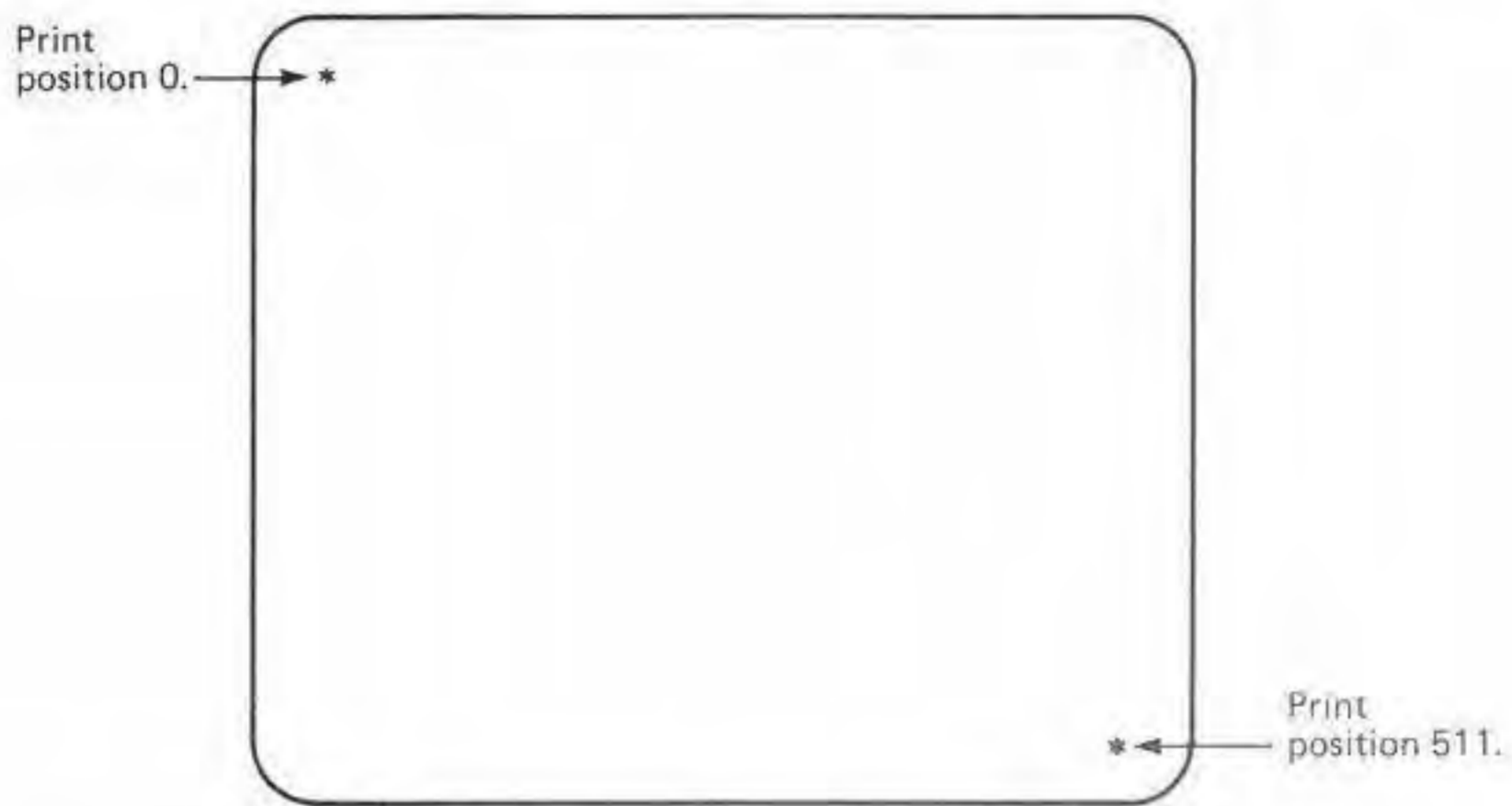
Print positions are numbered from _____ (top left) to _____ (bottom right).

0; 511

Think of the screen having 16 rows with 32 print positions in each row.



10. The screen has exactly 512 *print positions*. Each print position can display one character. Print positions are numbered from 0 to 511.

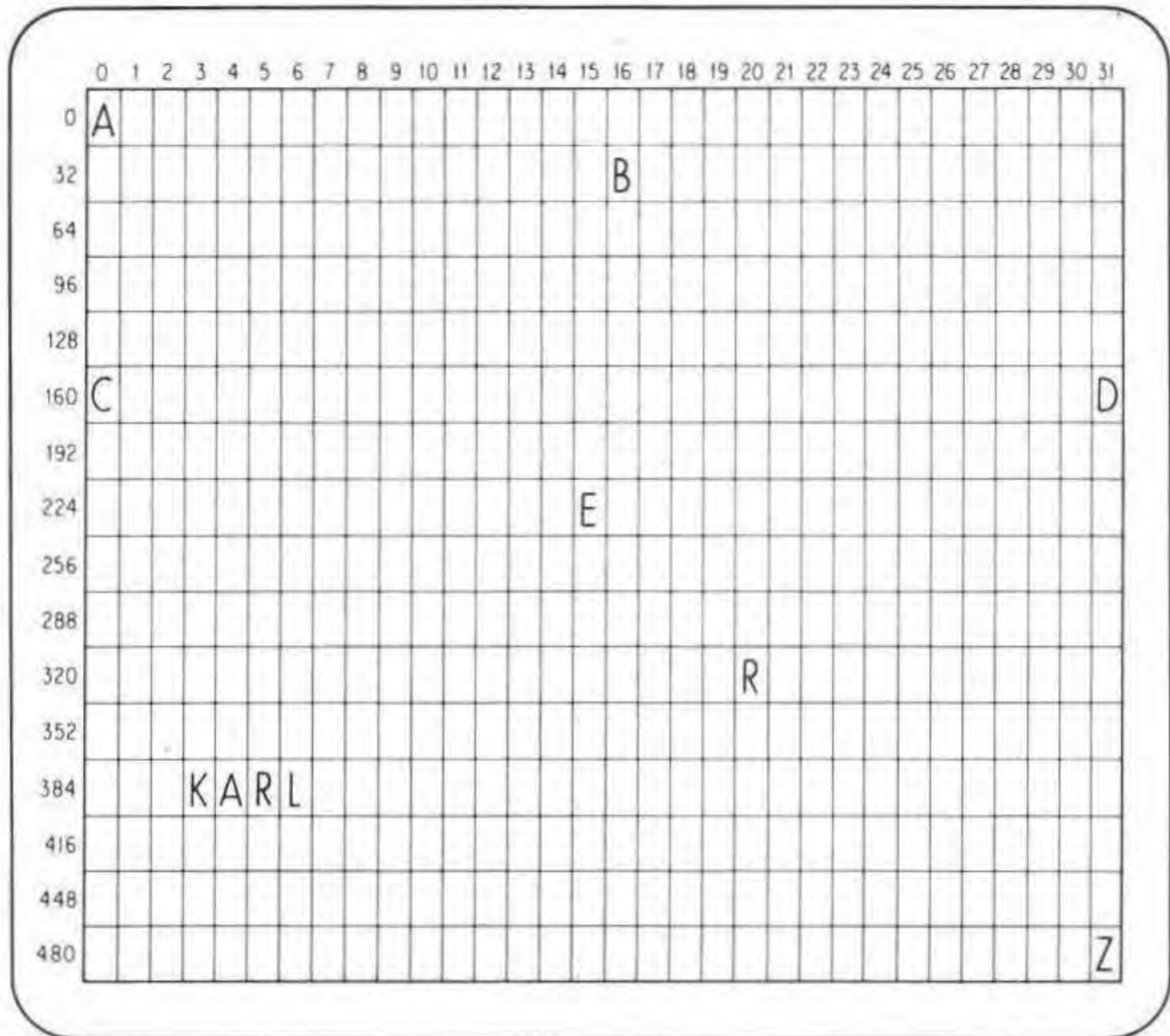


Print position 0 is at the upper left corner of the screen. Where is print position 511? _____

at the bottom right corner of the screen

11. Here is a more detailed map of the screen.

See Appendix F
for screen maps
like this.



The letter A is at screen position 0. The letter B is at print position 48 (32 + 16).

- (a) Where is the letter C? _____
- (b) Where is the letter D? _____
- (c) What is at print position 200? _____
- (d) What is at print position 239? _____
- (e) What print position is R at? _____
- (f) The word KARL uses 4 print positions. What are they? _____
- (g) What is at print position 512? _____

- (a) At print position 160.
- (b) At print position 191 (160 + 31).
- (c) Nothing. Or, perhaps an invisible space.
- (d) E
- (e) 340
- (f) 387, 388, 389 and 390
- (g) Oops! There is *no* print position 512. Z is at print position 511. Remember, print positions are numbered 0 to 511.

12. One of the nicest things about the Color Computer is how easy it is to tell the computer to print something *anywhere* on the screen. Easy! Just use the PRINT @ statement.

PRINT @

When you talk about the PRINT @ statement, call it the "PRINT AT" statement. Here are some examples:

The statement: PRINT @0, "A"
tells the computer to print the letter A at print position 0.

The statement: PRINT @388, "KARL"
tells the computer to print KARL at print position 388. KARL will then occupy positions 388, 389, 390, and 391.

PRINT @ 388, "KARL"

Print at this position

Put a comma here.

Your turn.

- (a) The statement: PRINT @128, "LIZZIE" tells the computer to print the string _____ at print position _____.
- (b) Write a statement to tell the computer to print JIM at print position 144. _____

- (a) LIZZIE; 128
- (b) PRINT @144, "JIM"

Did you remember the comma?



13. Experiment! Here is a program to help you print whatever you want, wherever you want it:

```

100 REM**PRINT WHAT WHERE
200 REM**ASK FOR INFORMATION
210 CLS
220 INPUT "WHAT SHALL I PRINT"; W$
230 INPUT "WHERE SHALL I PRINT IT"; W

300 REM**PRINT W$ AT POSITION W
310 CLS
320 PRINT @W, W$;

400 REM**VARIABLE TIME DELAY
410 Z = 2000
420 FOR K = 1 TO Z: NEXT K

500 REM**DO IT AGAIN
510 GOTO 210

```

In the above program, W\$ is a _____ variable and W is a _____ variable.

 string; numeric

Experiment. Change lines 210 and 310 as follows:

210 CLS 2 310 CLS 2

or

210 CLS 8 310 CLS 8

or

You pick the colors.

14. Enter the program from the preceding frame and RUN it. Try this:

```

WHAT SHALL I PRINT? CHEWBACCA
WHERE SHALL I PRINT IT? 270■

```

Before pressing ENTER.

Now press ENTER.

```

CHEWBACCA

```

This stays on-screen
for about four seconds.

Hmmm, what might happen if we tell the computer to print something in print position 512? Try it.

```
WHAT SHALL I PRINT? DARTH VADER
WHERE SHALL I PRINT IT? 512 ■
```

↖ Before pressing ENTER.

Now press ENTER.

```
?FC ERROR IN 320
OK
■
```

Well, there isn't a print position 512, so the computer printed an error message. FC means "Function Call," which doesn't tell much. In this case, it means that in line 320, we tried to print something at a nonexistent print position.

Print positions are numbered from _____ to _____

0: 511

This is also illegal: PRINT @-1, "*"
 But this is OK: PRINT @23.67, "*"

The computer will ignore the .67 and print at position 23.

15. Your turn. Write a program to
- (a) clear the screen;
 - (b) ask for a message (string);
 - (c) ask for a place to blink it;
 - (d) blink the message entered in (b) at the place entered in (c).

Use the REMark statements as an outline of the program.

```
100 REM**MESSAGE BLINKER  
200 REM**GET MESSAGE AND PLACE  
300 REM*BLINK MESSAGE ON  
400 REM**BLINK MESSAGE OFF  
500 REM**DO IT AGAIN  
900 REM**TIME DELAY SUBROUTINE
```

We did it like this. Your program might be different.

```
100 REM**MESSAGE BLINKER  
200 REM**GET MESSAGE AND PLACE  
210 CLS  
220 INPUT "YOUR MESSAGE"; M$  
230 INPUT "WHERE SHALL I BLINK IT"; P  
300 REM**BLINK MESSAGE ON  
310 CLS: PRINT (@P, M$;  
320 Z = 500  
330 GOSUB 910  
400 REM**BLINK MESSAGE OFF  
410 CLS 2  
420 Z = 300  
430 GOSUB 910  
500 REM**DO IT AGAIN  
510 GOTO 310  
900 REM**TIME DELAY SUBROUTINE  
910 FOR K=1 TO Z: NEXT K  
920 RETURN
```

16. Here are some programs for you to try. *Before* you enter and run a program, try to puzzle out what will happen.

```

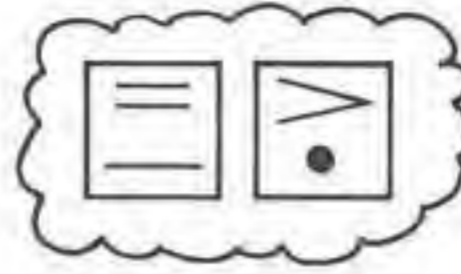
100 REM**ARROWS
110 CLS 2

200 REM**LET FLY AN ARROW
210 FOR P=0 TO 509
220   PRINT @ P, "->";
230   Z = 20
240   GOSUB 910
250 NEXT P

300 REM**DO IT AGAIN
310 GOTO 110

900 REM**TIME DELAY SUBROUTINE
910 FOR K=1 TO Z: NEXT K
920 RETURN

```



Hmmm . . . now try this variation:

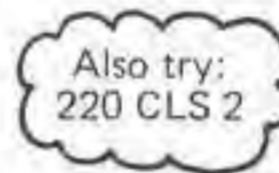
```

100 REM**INDEFATIGABLE ARROW
200 REM**LET FLY AN ARROW
210 FOR P = 0 to 509
220   CLS
230   PRINT @ P, "->";
240   Z = 20
250   GOSUB 910
260 NEXT P

300 REM**DO IT AGAIN
310 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 FOR K=1 TO Z: NEXT K
920 RETURN

```





Now make two changes and run the program.

```

100 REM**TIRED ARROW
240   Z = P

```

When you walk, run, fly, or teleport—are you like INDEFATIGABLE ARROW or TIRED ARROW?

17. Write a program to make a single left-pointing arrow (<-) travel from printing position 509 to position 0, then repeat over and over and over. For the left-pointing arrow, use these keys:  

```
100 REM**UNTIRING LEFT ARROW
```

```
200 REM**FLY, OH ARROW!
```

```
210 FOR P = 509 TO 0 STEP -1
```

```
220   CLS 2
```

```
230   PRINT @ P, "<-";
```

```
240   Z = 20
```

```
250   GOSUB 910
```

```
260 NEXT P
```

```
300 REM**DO IT AGAIN
```

```
310 GOTO 210
```

```
900 REM**TIME DELAY SUBROUTINE
```

```
910 FOR K=1 TO Z: NEXT K
```

```
920 RETURN
```



STEP -1 causes P to go "backwards" from 509 to 0.

Upstart! Who would believe a talking gauntlet?



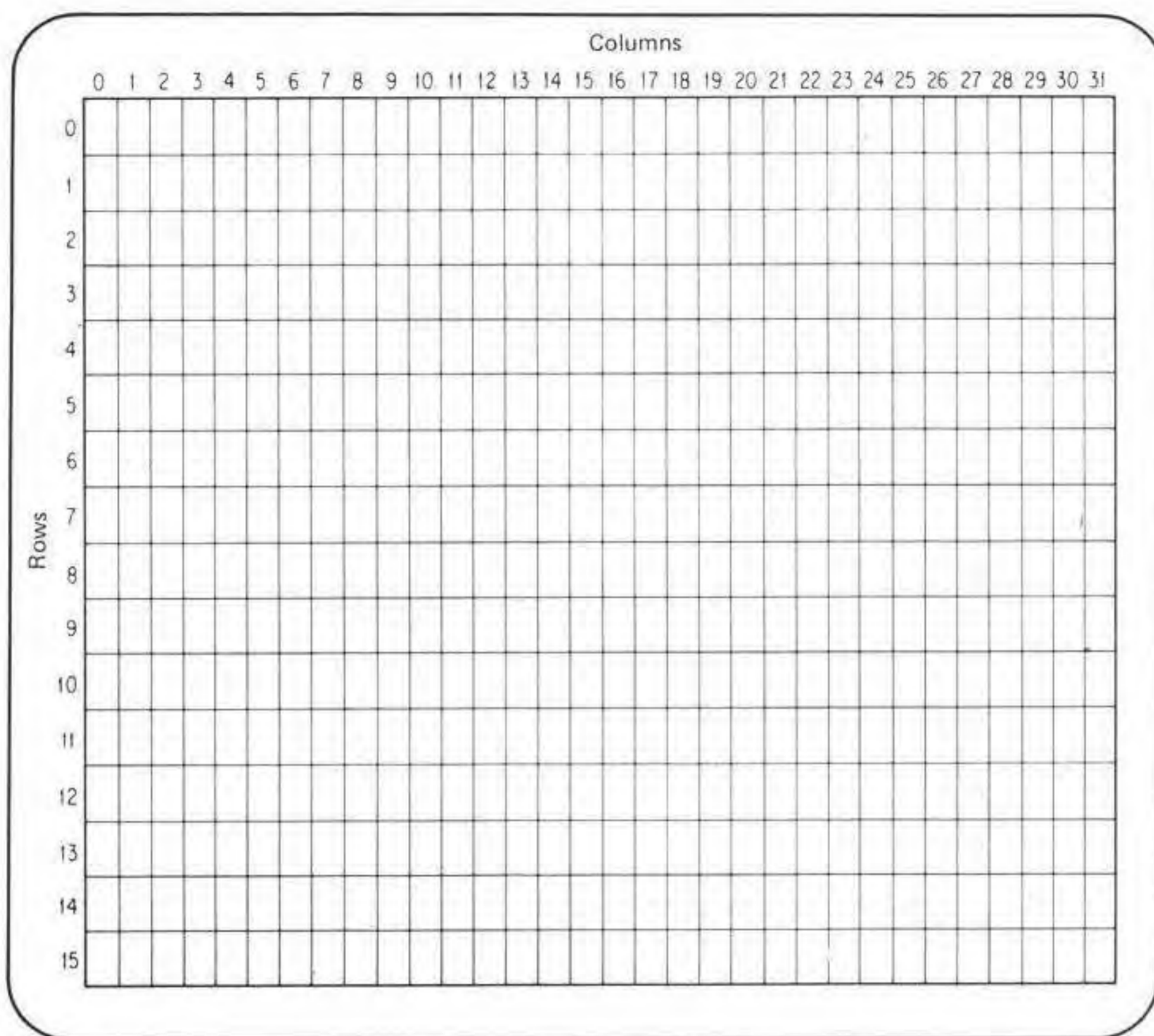
Why 509 instead of 511? We are trying to avoid printing anything in position 511, because that causes the screen to scroll—everything would move up one line. The backward arrow will first appear in positions 509 and 510, thus avoiding position 511.

18. Here again is a screen map:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0																																
32																																
64																																
96																																
128																																
160																																
192																																
224																																
256																																
288																																
320																																
352																																
384																																
416																																
448																																
480																																

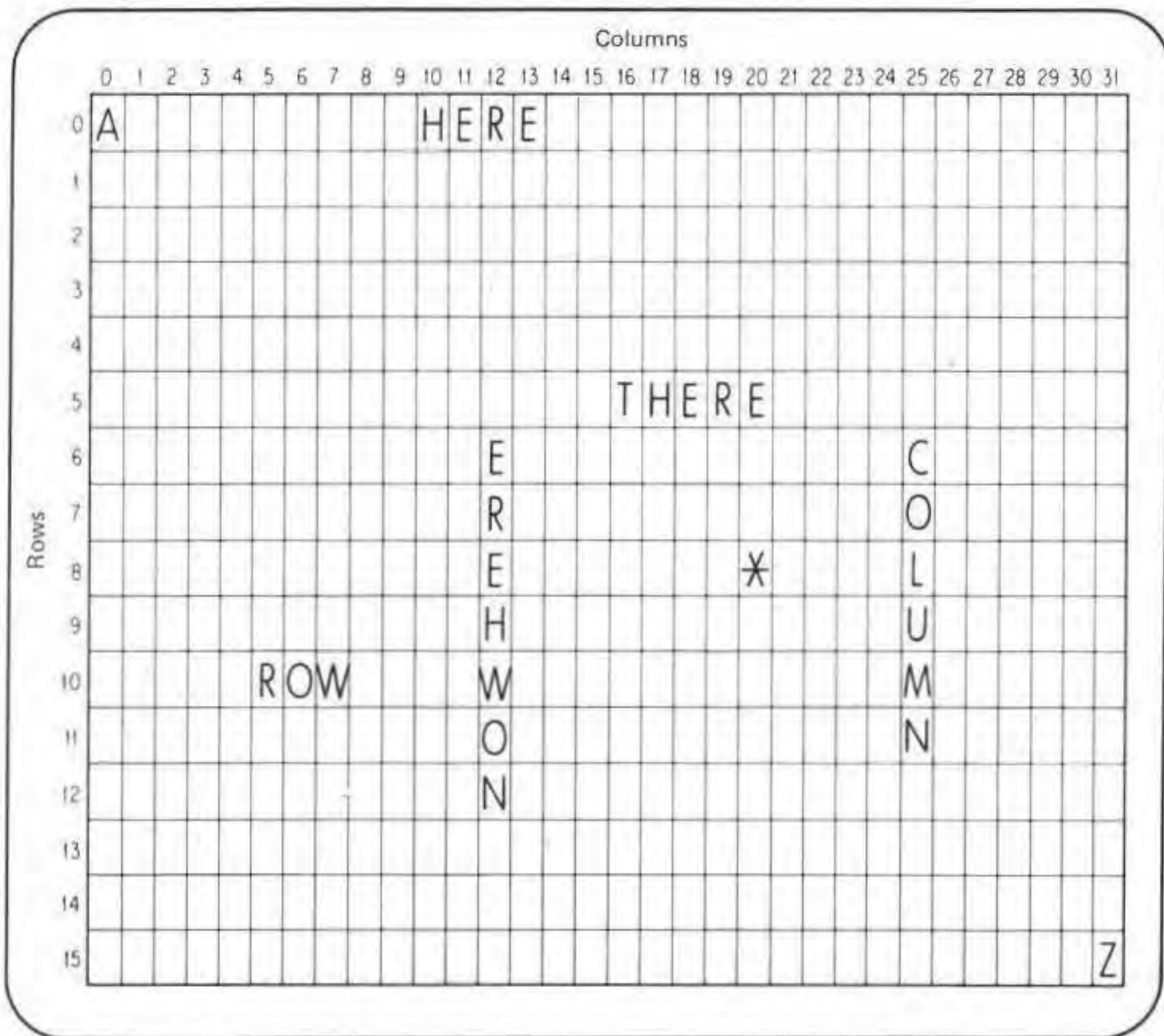
Look at the left edge of the screen. There are 16 lines with 32 print positions in each line. Going across, the positions are numbered 0 to 31. Going down, the positions are 0, 32, 64, 96, and so on.

Here is a slightly different screen map.



This map shows the screen with 16 ROWS, numbered 0 to 15, and 32 COLUMNS, numbered 0 to 31.

19. Explore the following screen map.



- (a) In what row is the word ROW? _____
- (b) In what column is the word COLUMN? _____
- (c) What is in row 0, column 0? _____
- (d) Where is Z? At row _____, column _____.
- (e) Where is a lonely star (*)? _____
- (f) What is at row 0, columns 10, 11, 12 and 13? _____
- _____
- (g) Where is THERE? _____
- (h) Where is NOWHERE? _____

(a) 10; (b) 25; (c) A; (d) 15, 31; (e) row 8, column 20; (f) HERE; (g) row 5, columns 16, 17, 18, 19 and 20; (h) column 12, rows 12, 11, 10, 9, 8, 7, and 6. Hmmm . . . we suspect that, at first, you thought NOWHERE was, well, nowhere.

20. EXPERIMENT! Play with this program:

```

100 REM**EXPERIMENT: ROWS, COLUMNS
200 REM**DIALOG WITH HUMAN
210 CLS
220 INPUT "YOUR MESSAGE": M$
230 INPUT "WHAT ROW    "; ROW
240 INPUT "WHAT COLUMN "; COL

300 REM**PRINT MESSAGE AT ROW, COL
310 SP = 32*ROW + COL
320 PRINT @ SP, M$:

400 REM**TIME DELAY
410 Z = 2000
420 FOR K = 1 TO Z: NEXT K

500 REM**DO IT AGAIN
510 GOTO 210

```



In line 230, ROW is a numeric variable. OK values are 0 to 15. COL is a numeric variable. OK values are 0 to 31. These values are used in line 310 to compute Screen Position (SP).

$$310 \text{ SP} = \underbrace{32 * \text{ROW}}_{\substack{\text{32 times} \\ \text{ROW number} + \text{COL number}}} + \text{COL}$$

To tell the computer to multiply, use the asterisk (*).

Experiment with this program. Remember, ROW numbers should be 0 to 15. Try 16 or -1 and see what happens. COLUMN numbers should be 0 to 31. Try 32 or -1 to see what happens.

If things happen too quickly, change the time delay in line 410.

Self-Test

What, another Self-Test? Of course—another chance for you to increase your self-esteem.

1. In the following Mystery Program, complete the REMark statements in lines 200, 300, and 400.

```

100 REM**MYSTERY PROGRAM
110 A$ = "*"

200 REM**_____
210 CLS 3
220 PRINT A$:
230 FOR K=1 TO 100: NEXT K

300 REM**_____
310 CLS 3
320 FOR K=1 TO 100: NEXT K

400 REM**_____
410 GOTO 210

```

Also try it without
the semicolon, end
of line 220.



Without actually running the program, explain what the computer will do if you run the program. _____

Now, if you wish, RUN the program.

2. What will happen if you run the following program?

```

100 REM**MYSTERY PROGRAM

200 REM**???.
210 CLS 1: GOSUB 910
220 CLS 2: GOSUB 910
230 CLS 3: GOSUB 910
240 CLS 4: GOSUB 910
250 CLS 5: GOSUB 910
260 CLS 6: GOSUB 910
270 CLS 7: GOSUB 910
280 CLS 8: GOSUB 910

300 REM**DO IT AGAIN
310 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 Z = 200
920 FOR K=1 TO Z: NEXT K
930 RETURN

```

3. Rewrite the program in question 1 so the computer uses a variable time delay. The value of the time delay is set in line 120, as shown below.

```
100 REM ** BLINK VALUE OF A$
110 A$= "***"
120 Z = 100

200 REM ** BLINK A$ ON
300 REM ** BLANK SCREEN
400 REM ** DO IT AGAIN
900 REM ** TIME DELAY SUBROUTINE
```

4. What will happen when you run this program?

```
100 REM ** MYSTERY PROGRAM
200 REM ** BLINKING NUMBERS
210 FOR N=1 TO 9
220   CLS
230   PRINT N;
240   GOSUB 910
250   CLS
260   GOSUB 910
270 NEXT N

300 REM ** DO IT AGAIN
310 GOTO 210

900 REM ** TIME DELAY SUBROUTINE
910 Z = 200
920 FOR K=1 TO Z: NEXT K
930 RETURN
```

5. Rewrite the program in Question 4 so lines 220, 230, and 240 are on the same line; lines 250 and 260 are on the same line.

```
100 REM ** MYSTERY PROGRAM
200 REM ** BLINKING NUMBERS
210 FOR N=1 TO 9
220 _____
230 _____
240 NEXT N

300 REM ** DO IT AGAIN
310 GOTO 210
```

etc.

6. (a) The Color Computer has (how many?) _____ screen print positions numbered from _____ to _____.
- (b) Print positions are arranged in (how many?) _____ rows with (how many?) _____ positions in each row?
- (c) Complete the following table showing the first (leftmost) and last (rightmost) print position in each row.

Row	First Position	Last Position
0	0	31
1	32	63
2	64	95
3	96	127
4	_____	_____
5	_____	_____
6	_____	_____
7	_____	_____
8	_____	_____
9	_____	_____
10	_____	_____
11	_____	_____
12	_____	_____
13	_____	_____
14	_____	_____
15	_____	_____

7. Rewrite the program in question 1 so the value of A\$ is blinked near the center of the screen.
8. What will happen if you RUN the following program? Complete the REMark statements.

```

100 REM**MYSTERY PROGRAM
110 READ A$, SP

200 REM**_____
210 CLS 3
220 PRINT A$:
230 FOR K=1 TO 100: NEXT K

300 REM**_____
310 CLS 3
320 FOR K=1 TO 100:NEXT K

400 REM**_____
410 GOTO 210

500 REM**_____
510 DATA "*", 271
    
```


9. Examine the following program.

```
100 REM**BLINK VALUE OF A$
110 A$ = "*"
120 ROW = 7
130 COL = 15

200 REM**BLINK A$ ON
210 CLS 3
220 PRINT @ 7*ROW + COL, A$:
230 FOR K=1 TO 100: NEXT K

300 REM**BLANK SCREEN
310 CLS 3
320 FOR K=1 TO 100: NEXT K

400 REM**DO IT AGAIN
410 GOTO 210
```

- (a) What character will blink on and off? _____
(b) At what print position (0 to 511)? _____
(c) How would you change lines 120 and 130 so the blinking occurs at screen position 72? _____
-

10. Write a program to blink something somewhere on the screen. The placement and the actual message are to be entered in response to INPUT statements. When you RUN the program, it might go like this:

```
WHAT SHALL I BLINK? HAPPY BIRTHDAY
WHAT ROW NUMBER ? 7
WHAT COLUMN NUMBER? 8 ■
```

Cursor. ENTER not yet pressed.

Now, when someone presses **ENTER**, the screen goes blank except for the message **HAPPY BIRTHDAY** blinking in row 7, columns 8 through 21.

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. 200 REM**BLINK VALUE OF A\$ ON
300 REM**BLANK SCREEN
400 REM**DO IT AGAIN

The value of A\$ will blink on and off in the upper left corner of a blue screen. Experiment. Since line 110 assigns a star (*) as the value of A\$, a star (*) will blink on and off in the upper left corner of a blue screen.

If you want something else to blink, change line 110. For example, try this:

```
110 A$ = "GLENDA"
```

With the above change, Glenda's name will blink. (frames 1, 2)

2. The eight screen colors will blink on, one after the other. First green, then yellow, then blue, and so on. After orange appears, it will start all over again. (frame 7)

3. 100 REM**BLINK VALUE OF A\$
110 A\$ = "*"
 - 120 Z = 100

```
200 REM**BLINK A$ ON
210 CLS 3
220 PRINT A$:
230 GOSUB 910
```

```
300 REM**BLANK SCREEN
310 CLS 3
320 GOSUB 910
```

```
400 REM**DO IT AGAIN
410 GOTO 210
```

```
900 REM**TIME DELAY SUBROUTINE
910 FOR K=1 TO Z: NEXT K
920 RETURN
```

(frames 7,8)

4. The number 1 will blink once; the number 2 will blink twice; the number 3 will blink 3 times . . . and so on. The number 9 will blink 9 times; then it will start again from 1. (no particular frame)
5. 220 CLS: PRINT N;: GOSUB 910
230 CLS: GOSUB 910 (review from previous chapters)

6. (a) 512; 0; 511
(b) 16; 32

Row	First Position	Last Position	
0	0	31	
1	32	63	
2	64	95	
3	96	127	
4	128	159	
5	160	191	
6	192	223	
7	224	255	
8	256	287	
9	288	319	
10	320	351	
11	352	383	
12	384	415	
13	416	447	
14	448	479	
15	480	511	(frames 9, 10, 11, 18)

7. Easy! Change line 220, as follows:

```
220 PRINT @239, A$;
```

Print position 239 is near the center of the screen. You may have chosen print position 240 or 271 or 272 or ???

Here is another way. Add line 120 and change line 220, as follows:

```
120 SP = 239           We use SP to mean  
220 PRINT @SP, A$;    "Screen Position."   (frames 9-13)
```

8. 200 REM**BLINK A\$ ON
300 REM**BLANK SCREEN
400 REM**DO IT AGAIN
500 REM**DATA: VALUES OF A\$, SP

The value of A\$ will blink on and off at screen position SP. The values of A\$ and SP are read by line 110 from the DATA statement in line 510. (Chapters 4 & 5)

9. (a) A star (*) which is assigned as the value of A\$ in line 110.
 (b) Position $7 * 32 + 15 = 239$.
 (c) Let's see now . . . each row has 32 positions. Row 0 begins with position 0; row 1 begins with position 32; row 2 begins with position 64; aha! $64 + 8 = 72$. So, change lines 120 and 130, as follows:

```
120 ROW = 2
130 COL = 8
```

Actually, we did it this way:

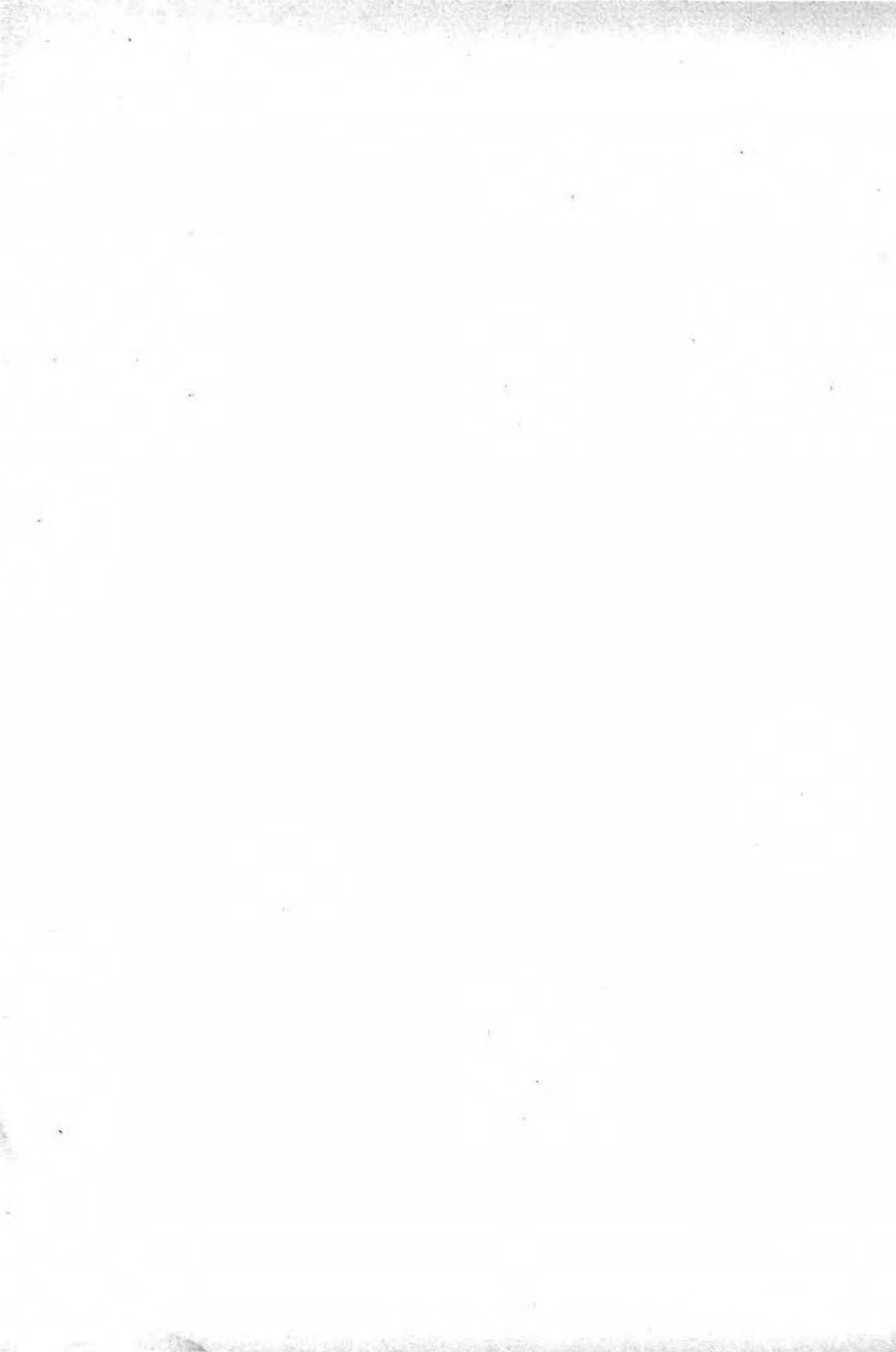
```

      2 ←ROW
32 ) 72
    64
    ---
     8 ←COL
```



Of course, you could also look at the screen map in Appendix F, find position 72, then look up the row and column. (frames 18, 20)

10. No answer. We are sure that you can do this one. Good luck!



CHAPTER SEVEN

Graphics Galore

In this chapter, you will begin to learn how to put patterns and pictures of bright colors on the Color Computer screen.

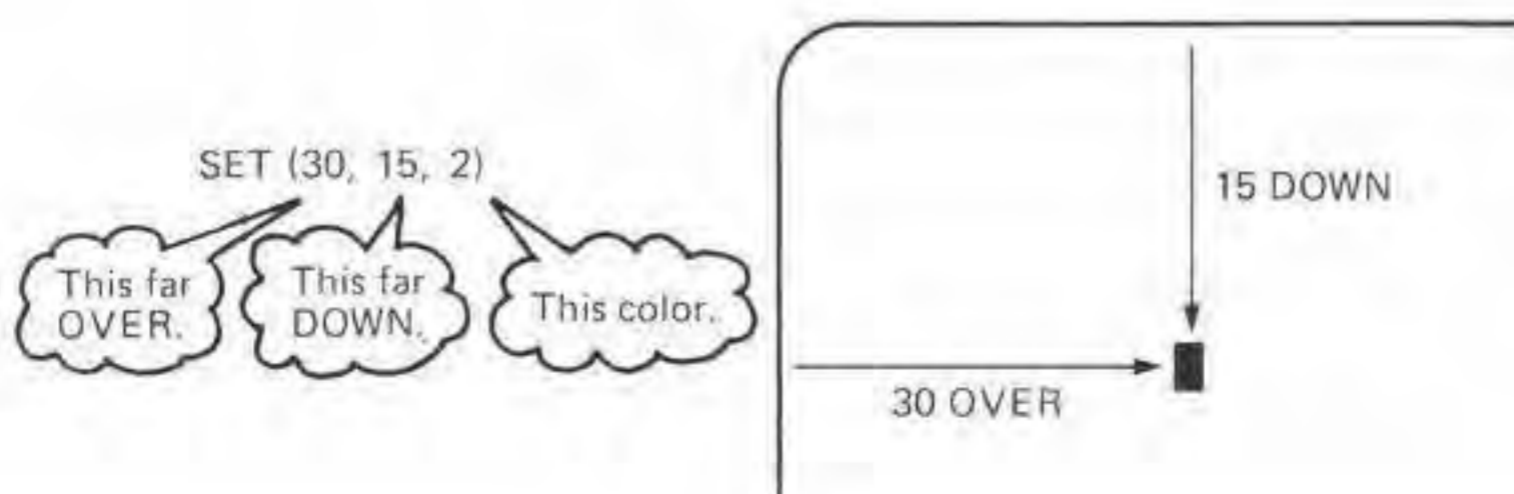
When you finish this chapter, you will be able to:

- Use the SET statement to turn on tiny rectangles of light in 8 different colors;
- Control the placement of tiny rectangles of light on the screen—the key to making patterns and pictures on the screen;
- Understand, use, and write programs to “paint” stripes of color on the screen;
- Put “constellations” of brightly colored “stars” on the screen;
- Recognize and use 16 graphic shapes in 8 different colors.

1. Patterns! That’s what you will do in this chapter—you will put patterns on the screen, using little rectangles of colored light. To do this, you use the SET statement.

SET turns on a tiny rectangle of colored light somewhere on the screen.

Somewhere on the screen? Well, of course, you must tell the computer *where* to turn on the tiny rectangle of light. You must also tell it what color you want.

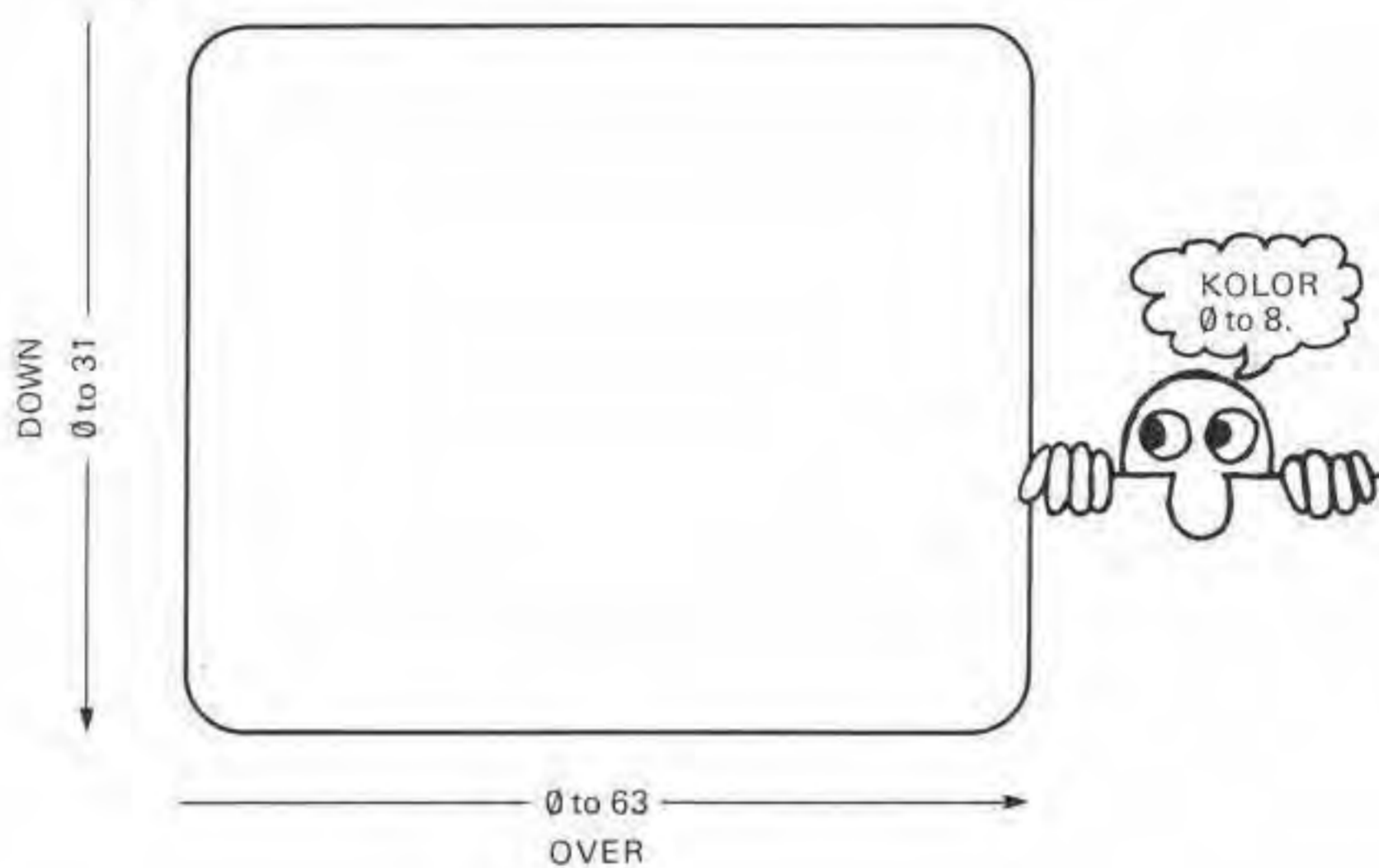


- (a) SET(10, 20, 3) tells the computer to turn on a tiny rectangle of colored light.
How far OVER?____How far DOWN?____What color?_____
- (b) SET(0, 0, 4) tells the computer to turn on a tiny rectangle of colored light.
How far OVER?____How far DOWN?____What color?_____

- (a) 10; 20; blue
- (b) 0;0; red (This is the upper left corner of the screen.)

2. Think of it like this: SET(OVER, DOWN, KOLOR)

OVER can be a whole number, 0 to 63.
DOWN can be a whole number, 0 to 31.
KOLOR can be a whole number, 0 to 8.



- (a) What is the smallest value allowed for OVER?____
- (b) What is the largest value allowed for OVER?____
- (c) What is the smallest value allowed for DOWN?____
- (d) What is the largest value allowed for DOWN?____
- (e) What is the smallest value allowed for KOLOR?____
- (f) What is the largest value allowed for KOLOR?____

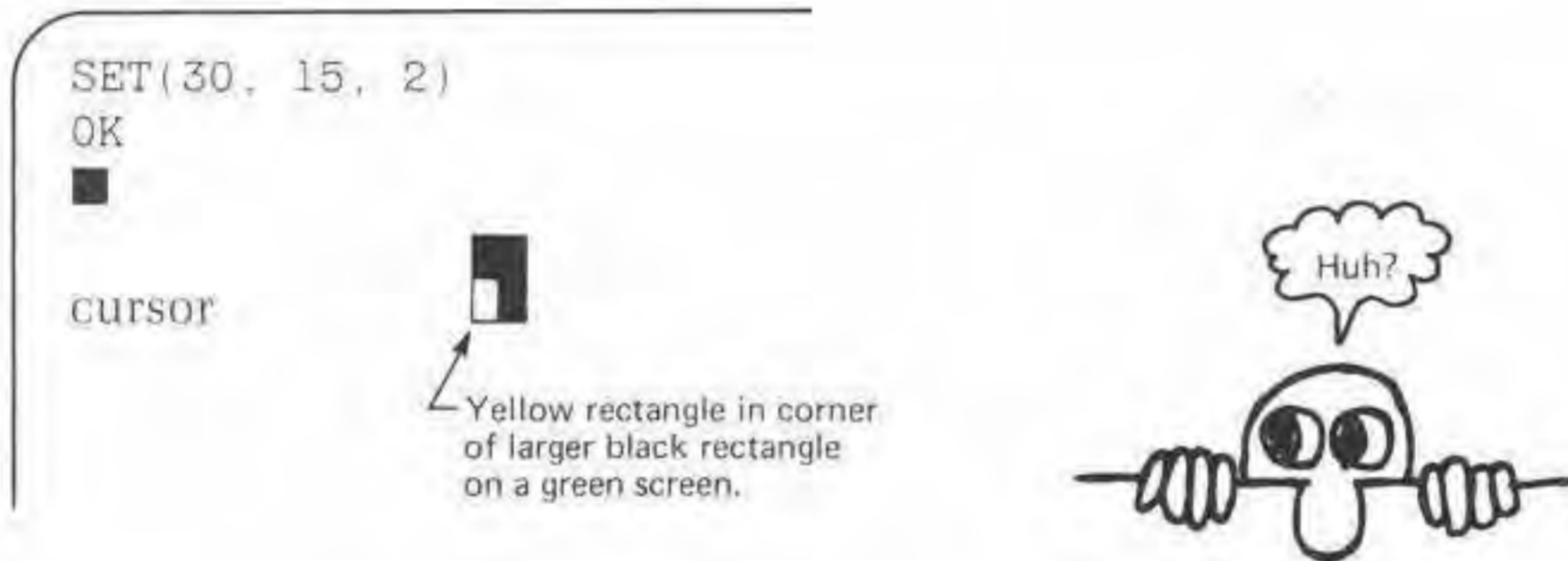
- (a) 0; (b) 63; (c) 0; (d) 31; (e) 0; (f) 8

You have probably guessed that numbers outside these ranges produce error messages.

BEWARE! COLOR may be used as a numeric variable on the minimum Color Computer. However, it is a reserved word in *Extended Color BASIC*. If you are using *Extended Color BASIC*, use COLR or COULOUR or KOLOR or HUE or TINT or PIGMENT or perhaps just the letter C.

We will frequently use KOLOR.

3. Try some. Press **CLEAR**, then type SET(30, 10, 2) and press **ENTER**. This is what you will see:

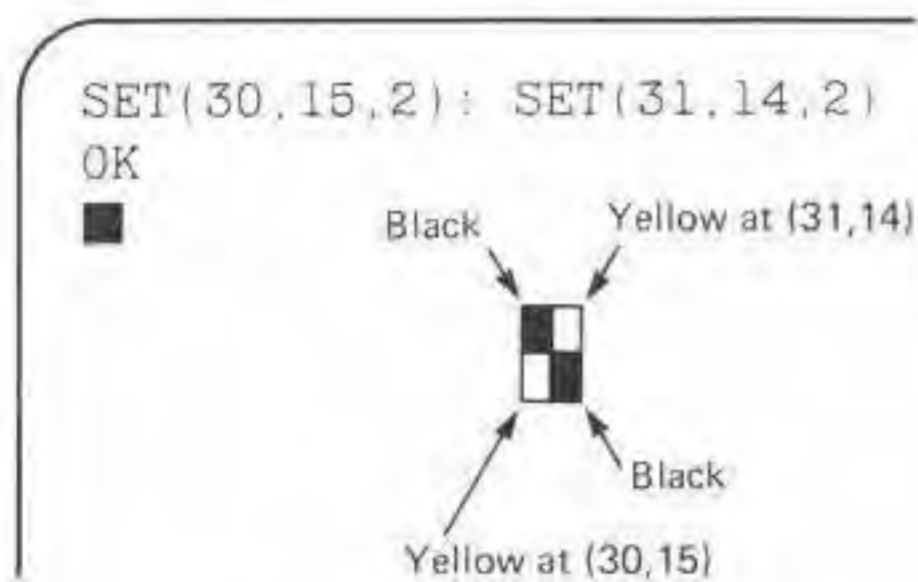


Hmmm . . . that's not quite what we said would happen, is it? You see a yellow rectangle in one corner of a larger black rectangle on a green screen.

The larger black rectangle is a screen print position. The smaller yellow rectangle is SET in one of four corners of a print position. Try this:

Press **CLEAR**.

Type SET(30, 15, 2); SET(31, 14, 2) and press **ENTER**.



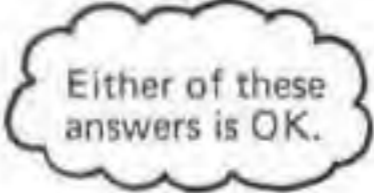
DON'T press **CLEAR**. Instead, type two SET statements on the same line to set the two remaining black areas to yellow.

You type: _____

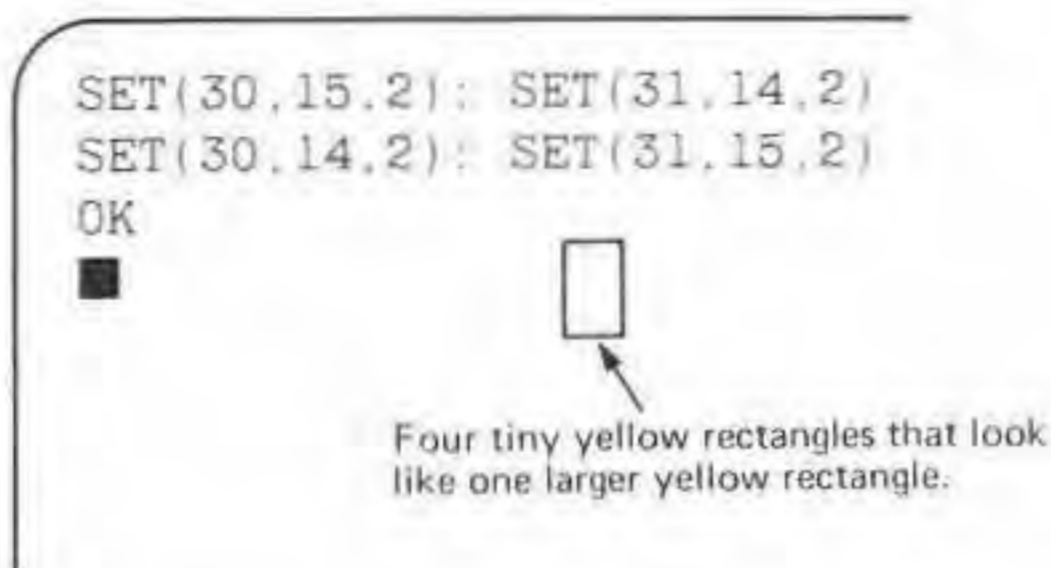
SET(30,14,2): SET(31,15,2)

OR

SET(31,15,2): SET(30,15,2)



The screen might now look like this:

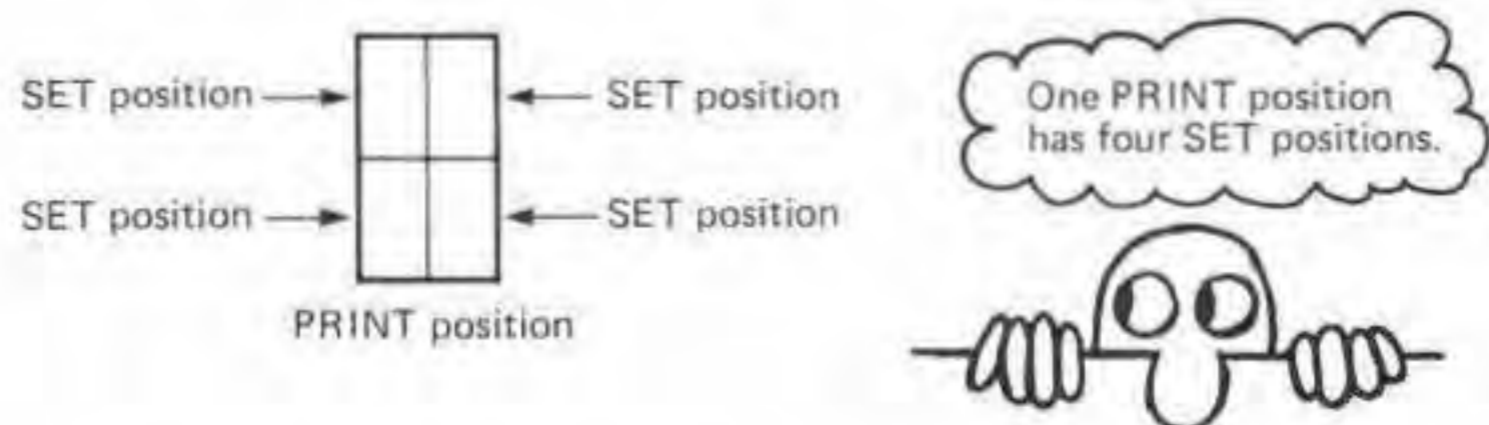


4. Press **CLEAR**, then try this:

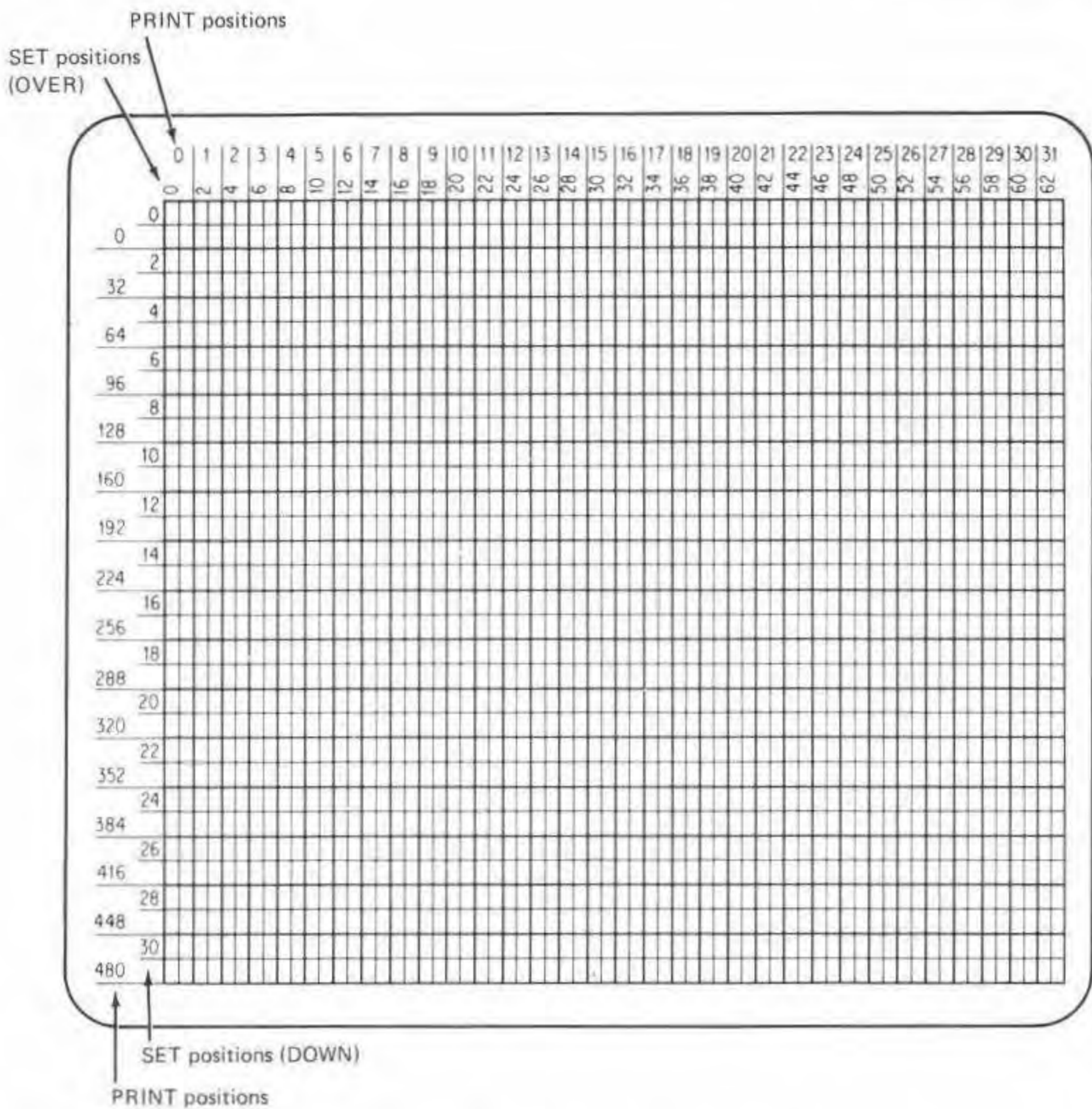
- (1) Type SET(30,14,2) and press **ENTER**. This turns on a tiny yellow blip at position (30,14).
- (2) Type SET(31,14,3) and press **ENTER**. This turns on a tiny blue blip at position (31,14). Oops! The tiny yellow blip also went blue! You now have *two blue blips*.
- (3) Type SET(30,15,4) and press **ENTER**. Wait a minute! There are now *three red blips*. What's going on here?
- (4) Guess what will happen if you type SET(31,15,8) and press **ENTER**.

All four blips are now orange. This happens because SET positions (30,14), (30,15), (31,14), and (31,15) are all in the *same screen print position*. What does that mean? Read on.

5. Each screen PRINT position contains four SET positions.



The following screen map shows both PRINT and SET positions.



Complete the following table showing the correspondence between PRINT positions and SET positions.

	PRINT POSITIONS	SET POSITIONS (OVER, DOWN)
	0	(0,0), (1,0), (0,1), and (1,1)
(a)	511	_____
(b)	73	_____
(c)	_____	(12,16), (13,16), (12,17), (13,17)
(d)	_____	(62,10), (63,10), (62,11) (63,11)

(a) (62,30), (63,30), (62,31), (63,31); (b) (18, 4), (19, 4), (18,5), (19,5);
 (c) 262; (d) 191

6. EXPERIMENT! Try this program:

```
100 REM ** EXPERIMENT WITH SET
200 REM ** DIALOG WITH EXPERIMENTER
210 CLS
220 INPUT "HOW FAR OVER"; OVER
230 INPUT "HOW FAR DOWN"; DOWN
240 INPUT "WHAT COLOR "; KOLOR

300 REM ** TINY COLOR ON BLACK SCREEN
310 CLS 0
320 SET(OVER, DOWN, KOLOR)

400 REM ** VARIABLE TIME DELAY
410 Z = 2000
420 FOR K=1 TO Z: NEXT K

500 REM ** GO DO IT AGAIN
510 GOTO 210
```

It might go like this:

```
HOW FAR OVER? 30
HOW FAR DOWN? 15
WHAT COLOR ? 2■ ← cursor
```

Before you press **ENTER**, guess where the tiny rectangle will appear on the screen. Then press **ENTER**. The screen will become *all black* (see line 310), except for a tiny yellow rectangle near the center of the screen.

VARIATIONS: Change line 310 so that the screen is cleared to a color other than black. Try CLS 2 or CLS 3 or whatever you want. Then RUN the program to find out what happens.

7. Here is a different program to experiment with. In this program, our variables are OV for OVER, DO for DOWN, and CO for color in lines 220 and 310.

```
100 REM ** EXPERIMENT WITH SET
110 CLS 0

200 REM ** DIALOG WITH A PERSON
210 PRINT @0,
220 INPUT "OVER, DOWN, COLOR"; OV, DO, CO

300 REM ** TURN ON TINY COLOR
310 SET(OV, DO, CO)

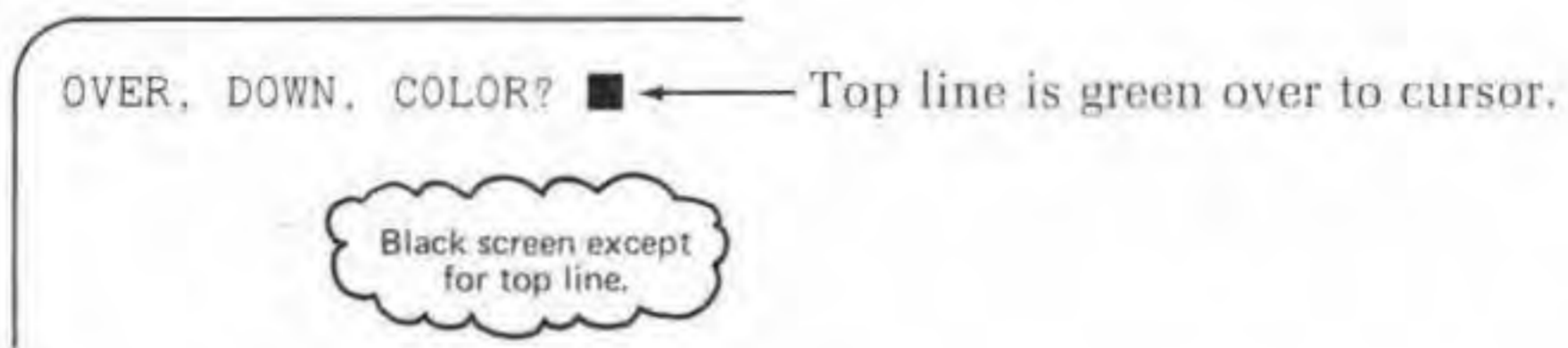
400 REM ** GO DO IT AGAIN
410 GOTO 210
```

Before you RUN the program, answer these questions. (It's OK to guess.)

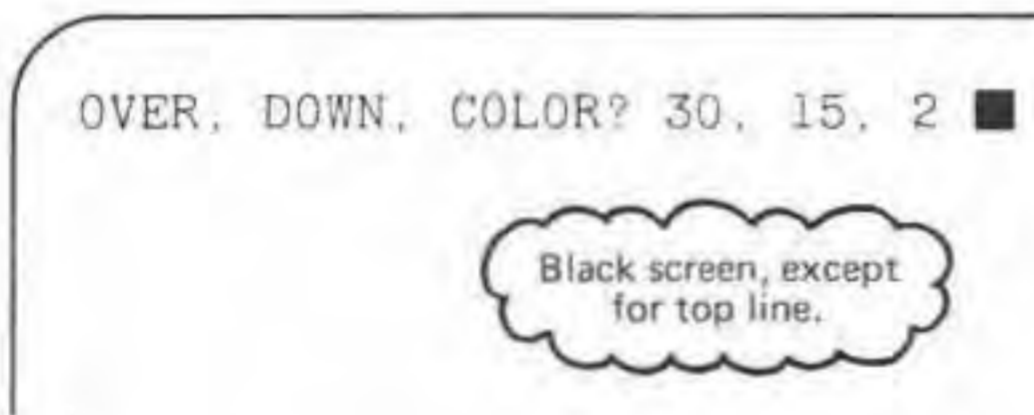
- (a) What color will the screen be (line 110)? _____
- (b) What will happen in lines 210 and 220? _____

-
- (a) Black, except for the top line, which will be mostly green, and for tiny color rectangles which you put on the screen as you use this program.
 - (b) Line 210 positions the cursor at print position zero. That's all it does. Then the INPUT statement (line 220) prints OVER, DOWN, COLOR? and waits for you to respond. The message OVER, DOWN, COLOR? will begin at print position zero.

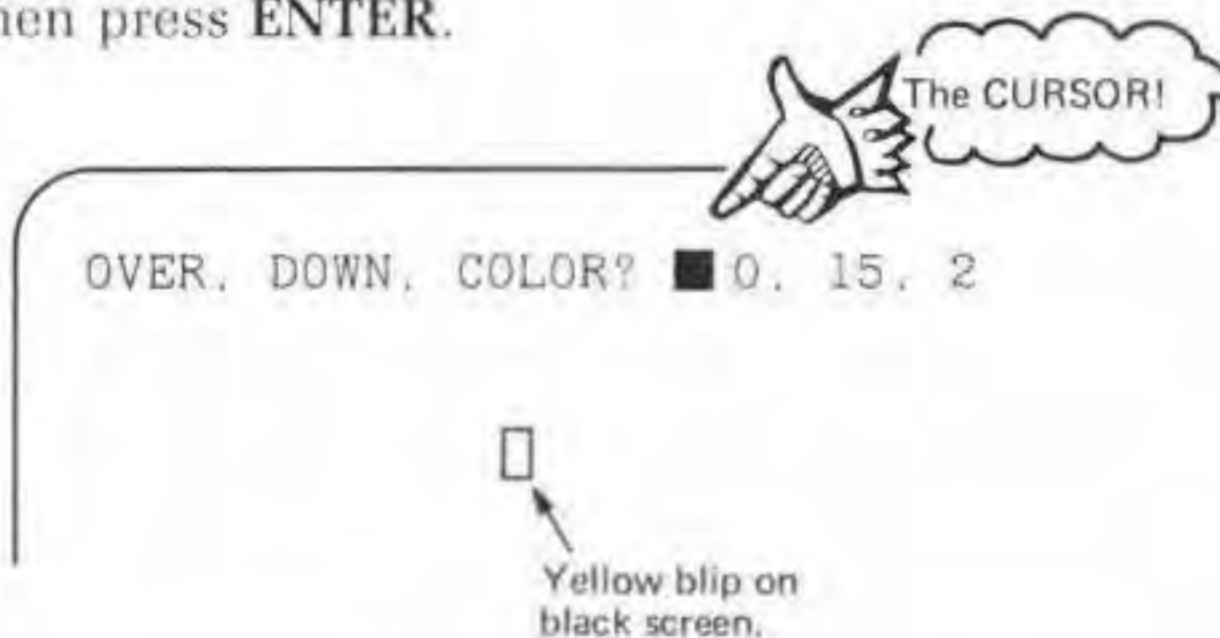
8. RUN it. It begins like this.



Now type values for OV, DO, and CO.



Then press ENTER.



Your turn. Guess why the cursor is where it is. _____

After putting the yellow blip on the screen, the computer executed the GOTO 210 in line 410. Again, look at lines 210 and 220. Together, they cause the message OVER, DOWN, COLOR? to be printed at screen print position zero. The cursor wiped out the 3 in 30. The numbers are simply left over from before. Go ahead, type new values and press **ENTER**. Then do it again, and again. Continue as long as you wish.

9. In the program in frame 7, replace block 200 with the following.

```
200 REM ** DIALOG WITH A PERSON
210 PRINT @0, CHR$(30)
220 PRINT @0,
230 INPUT "OVER, DOWN, COLOR": OV, DO, CO
```

Something new—the CHR\$ function. CHR is short for CHaRacter. You will get more acquainted with this character in Chapter 10. In line 210 above, CHR\$(30) tells the computer to erase everything to the end of the line being printed.

```
PRINT @0, CHR$(30)
```

↑
Beginning here, erase everything to the right on this line.

Try it. With this change, the computer will erase everything on the top line of the screen before doing lines 220 and 230. Presto! No leftover numbers.

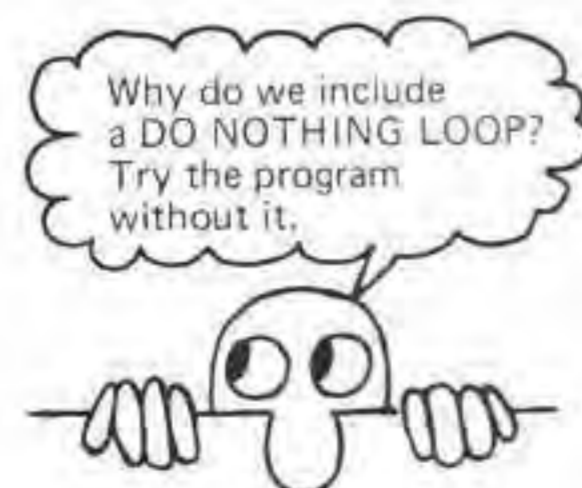
Variations: Change line 110 to CLS or CLS 8 or experiment.

10. Imagine that the screen is the night sky. You are looking north when you see . . . well, try this program to see what you might see.

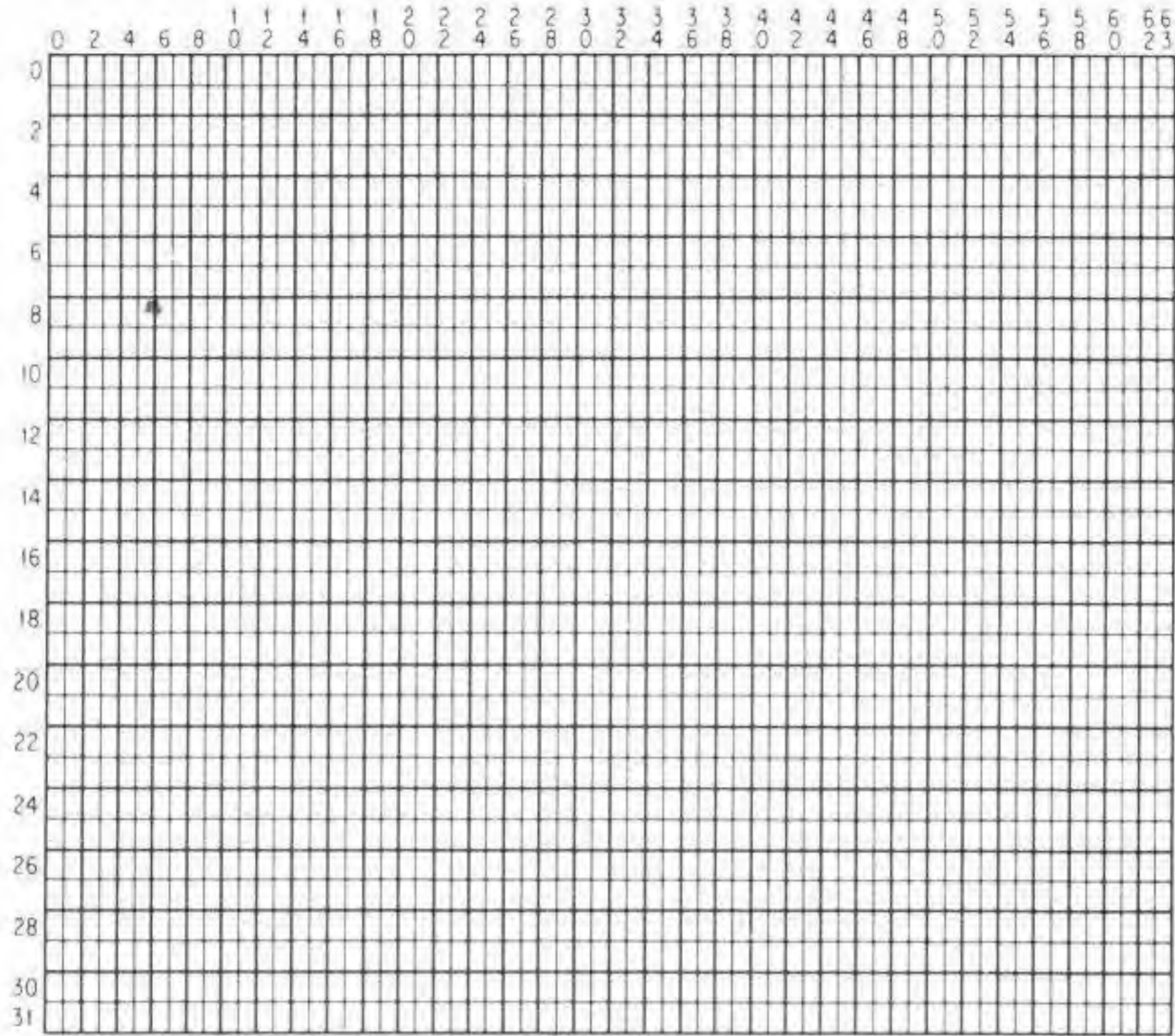
```
100 REM ** SOMETHING IN THE SKY
110 CLS 0

200 REM ** WINK THEM ON
210 SET( 6, 12, 2)
220 SET(18, 10, 2)
230 SET(26, 12, 2)
240 SET(34, 14, 2)
250 SET(38, 20, 2)
260 SET(54, 20, 2)
270 SET(56, 14, 2)

300 REM ** DO NOTHING LOOP
310 GOTO 310
```



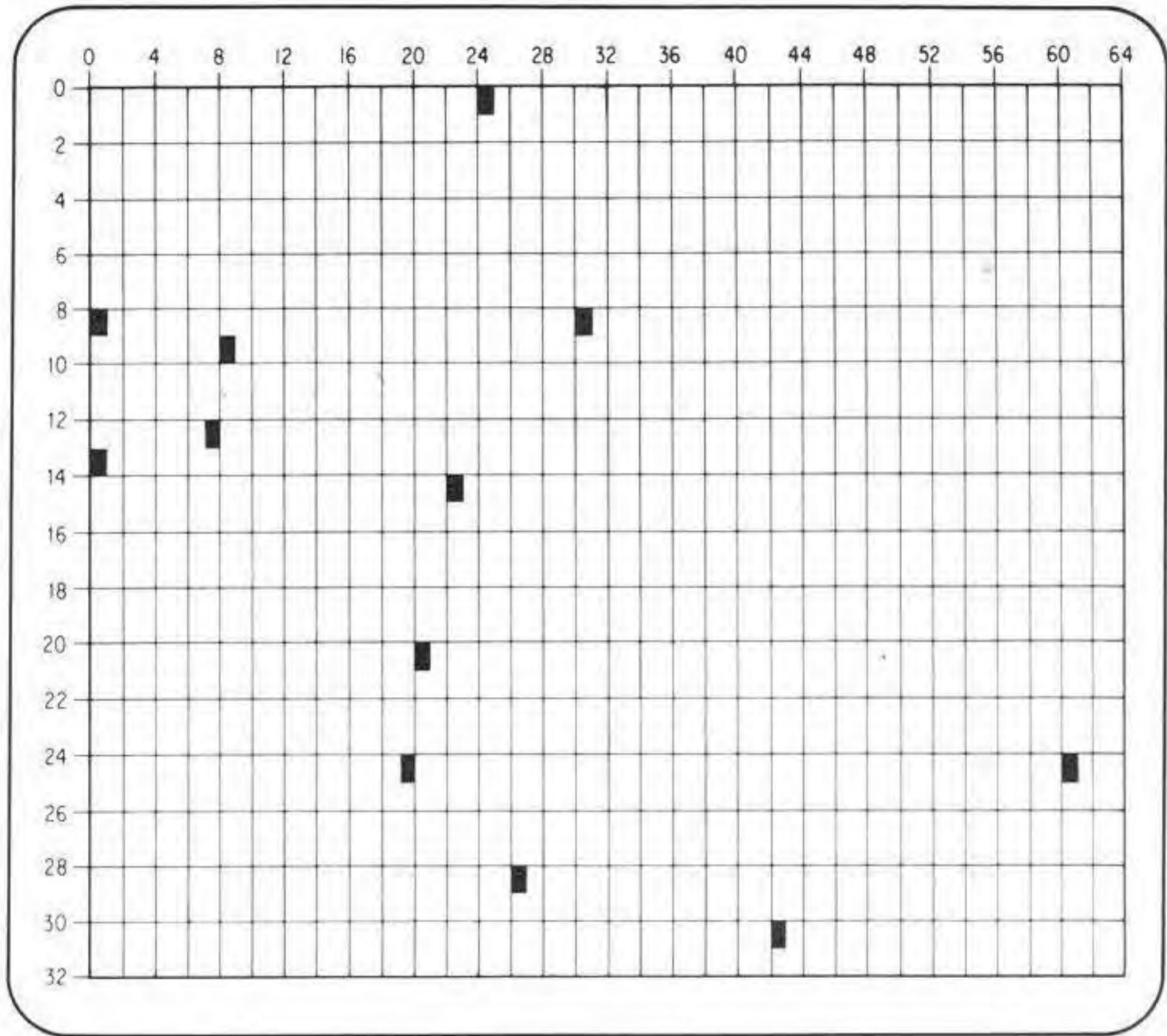
Plot the SET points on the following screen map.



What do you see? _____

We hope you said, "The Big Dipper."

11. Here is a screen map showing the constellation Draco.



Write a program to put Draco on the screen. Use orange stars.

```

100 REM ** DRACO ON A BLACK SCREEN
110 CLS 0

200 REM ** HERE'S DRACO!
210 SET( 0, 8, 8)
220 SET( 0, 13, 8)
230 SET( 7, 12, 8)
240 SET( 8, 9, 8)
250 SET(19, 24, 8)
260 SET(20, 20, 8)
270 SET(22, 14, 8)
280 SET(24, 0, 8)
290 SET(26, 28, 8)
300 SET(30, 8, 8)
310 SET(42, 30, 8)
320 SET(60, 24, 8)

400 REM ** DO NOTHING LOOP
410 GOTO 410

```

Of course, your program might be different.

12. Try using black (0) as the color in a SET statement. For example:

Do This	See This
Type CLS 0	Mostly black screen.
Type SET(32, 16, 2)	Yellow blip near center of screen.
Type SET(32, 16, 0)	Yellow blip is still there.

In a SET statement, color code 0 (black) has *no effect*. It does nothing at all.

So how do you turn off (to black) a single blip? Use the RESET statement.

RESET(OVER, DOWN) turns off (to black) the tiny rectangle of light at screen position OVER, DOWN.

Do This	See This
Type CLS 0	Mostly black screen.
Type SET(32, 16, 2)	Yellow blip near center of screen.
Type RESET(32, 16)	Yellow blip is gone.

Complete the following program to blink a red blip at screen position (31, 15).

```
100 REM ** BLIP BLINKER
110 CLS 0

200 REM ** TURN BLIP ON
210 SET(31, 15, 4)
220 GOSUB 910

300 REM ** TURN BLIP OFF
310 _____
320 _____

400 REM ** GO BLINK AGAIN
410 GOTO 210

900 REM ** TIME DELAY SUBROUTINE
910 Z = 100
920 FOR K=1 TO Z: NEXT K
930 RETURN
```

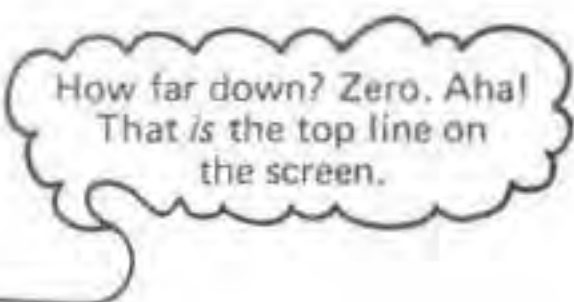
```
310 RESET(31, 15)          320 GOSUB 910
```

13. Blips are tedious. It takes lots of blips to put a little color on the screen. So let's paint stripes. First, paint a green stripe across the top of a black screen.

```
100 REM ** PAINT STRIPES
110 CLS 0

200 REM ** GREEN STRIPE
210 FOR OVER=0 TO 63
220   SET(OVER, 0, 1)
230 NEXT OVER

300 REM ** DO NOTHING LOOP
310 GOTO 310
```



How far down? Zero. Aha!
That is the top line on
the screen.

RUN it. The screen will go all black (line 110), then the computer will draw a narrow green stripe from left to right across the top of the screen. That's all . . . nothing more happens.

What is the purpose of line 310?_____

This causes the computer to seem to "do nothing" while you look at the green stripe. When you are tired of looking at a green stripe on a black screen, press **BREAK** and go on to the next frame.

14. Now we would like the computer to paint a yellow stripe below the green stripe so that *both* stripes remain on-screen. Which program segment below should we add, A or B?_____

A	B
300 REM ** YELLOW STRIPE	300 REM ** YELLOW STRIPE
310 FOR OVER=0 TO 63	310 FOR OVER=0 TO 63
320 SET(OVER, 1, 2)	320 SET(OVER, 2, 2)
330 NEXT OVER	330 NEXT OVER
400 REM ** DO NOTHING LOOP	400 REM ** DO NOTHING LOOP
410 GOTO 410	410 GOTO 410

Please explain your choice._____

B; alas, program A will paint a yellow stripe in the same PRINT positions as the green stripe. The green will thus change its stripe to yellow. Hmmm . . . can a green leopard change its stripes?

Try both program segments with the original program in the preceding frame.

15. Your turn. Write an additional program segment so the computer draws a green stripe, then a yellow stripe, then a blue stripe, and all three stripes remain on-screen.

```
400 REM ** PAINT A BLUE STRIPE
410 _____
420 _____
430 _____
500 REM ** DO NOTHING LOOP
510 _____
```

```
410 FOR OVER=0 TO 63
420 SET(OVER, 4, 3)
430 NEXT OVER
510 GOTO 510
```



NOT 3!

16. You could add a red stripe, buff stripe, and so on. But, there is an easier way! Save wear and tear on your fingers—use this short program.

```
100 REM ** COLOR STRIPES, THE EASY WAY
110 CLS 0

200 REM ** 8 STRIPES IN 8 COLORS
210 FOR KOLOR=1 TO 8
220   DOWN = 2*KOLOR - 2

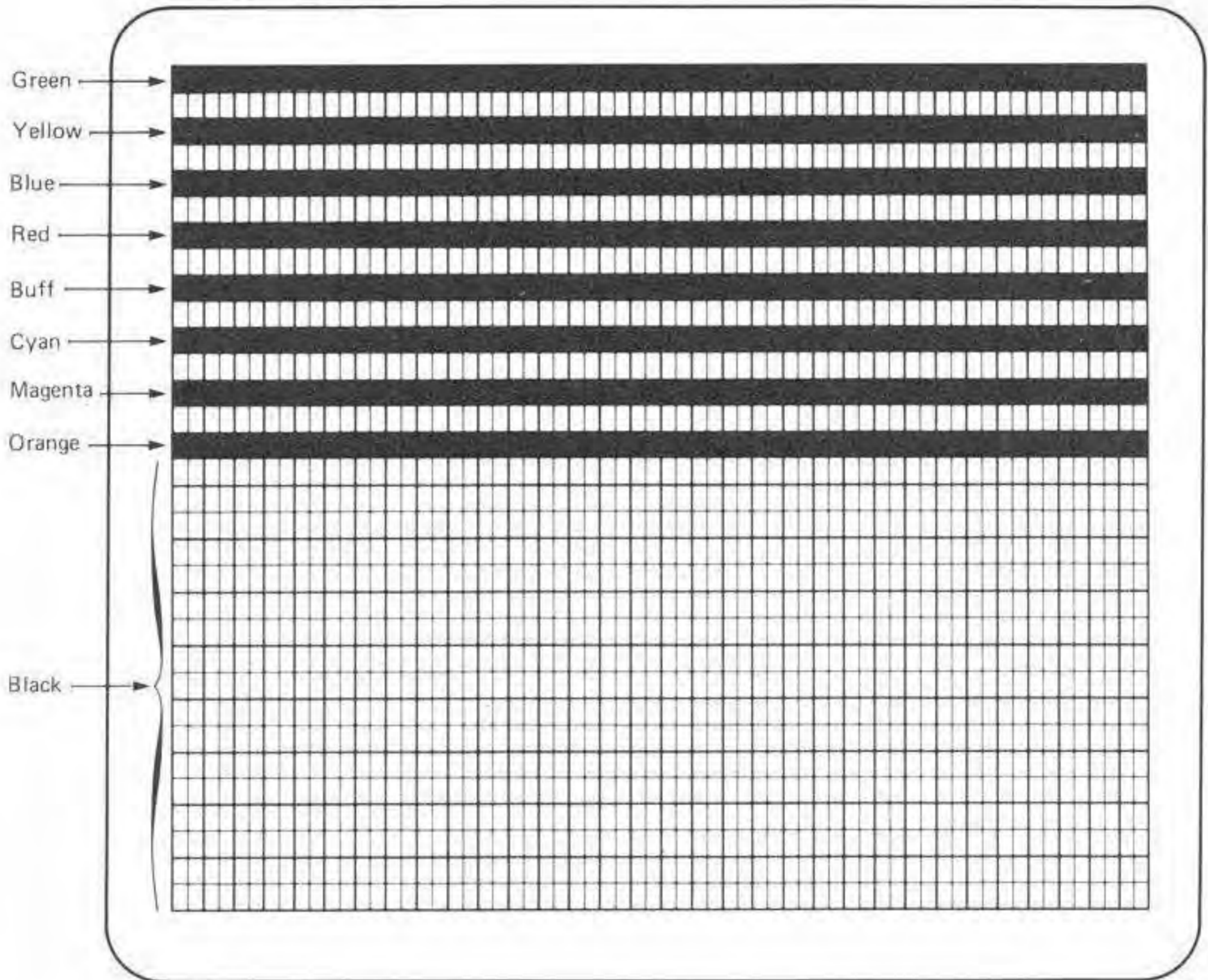
230   FOR OVER=0 TO 63
240     SET(OVER, DOWN, KOLOR) This paints a stripe
250   NEXT OVER                 whose color is KOLOR.

260 NEXT KOLOR

300 REM ** DO NOTHING LOOP
310 GOTO 310
```

RUN the program. The computer will paint 8 stripes in the top half of the screen, as indicated on the following screen map.

Between color stripes, the color is black.



If you get different colors, fiddle with the color controls on your TV until you get the above colors or the colors of your choice.

VARIATION: Change line 310 to 310 GOTO 110

17. Yes, perhaps that last program is a bit boggling. After all, it has a FOR-NEXT loop *inside* another FOR-NEXT loop.

```

200 REM ** 8 STRIPES IN 8 COLORS
210 FOR KOLOR=1 TO 8
220   DOWN = 2*KOLOR - 2
230   FOR OVER=0 TO 63
240     SET(OVER, DOWN, KOLOR)
250   NEXT OVER
260 NEXT KOLOR

```

} inside loop } outside loop

- (a) What values will KOLOR have (line 210)? _____
- (b) In line 220, the value of DOWN is computed. The computer multiplies (*) the value of KOLOR by 2, subtracts (-) 2, then assigns the result to DOWN.

When KOLOR is 1, DOWN is 0 (2*1 - 2 is 0)

When KOLOR is 2, DOWN is 2 (2*2 - 2 is 2)

When KOLOR is 3, DOWN is _____

When KOLOR is 4, DOWN is _____

and so on, until

when KOLOR IS 8, DOWN is _____

- (c) Got it? _____

-
- (a) 1, 2, 3, 4, 5, 6, 7, and 8
- (b) 4 (DOWN = 2*3 - 2 = 4)
- 6 (DOWN = 2*4 - 2 = 6)
- 14 (DOWN = 2*8 - 2 = 14)

Here is a handy table showing values of KOLOR and DOWN:

KOLOR	1	2	3	4	5	6	7	8
DOWN	0	2	4	6	8	10	12	14

- (c) We hope you said YES! or EUREKA! The computer will paint a green stripe (KOLOR = 1) at DOWN = 0, a yellow stripe (KOLOR = 2) at DOWN = 2, a blue stripe (KOLOR = 3) at DOWN = 4, and so on.

18. For each value of KOLOR and DOWN, the computer executes the inside loop (lines 230, 240, and 250), thus painting one stripe across the screen.

What variable changes as the computer goes around and around and around doing the inside loop? _____

OVER. It takes on values 0, 1, 2, 3, and so on, up to 63.

19. Your turn. Write a program to paint 8 vertical stripes on a black screen. Paint a green stripe at OVER = 0, a yellow stripe at OVER = 2, a blue stripe at OVER = 4, and so on until an orange stripe appears at OVER = 14. Here is a handy table for you to use:

KOLOR	1	2	3	4	5	6	7	8
DOWN	0	2	4	6	8	10	12	14

```

100 REM ** VERTICAL COLOR STRIPES
200 REM ** 8 STRIPES IN 8 COLORS
300 REM ** DO NOTHING LOOP

```

```

100 REM ** VERTICAL COLOR STRIPES
110 CLS 0

200 REM ** 8 STRIPES IN 8 COLORS
210 FOR KOLOR=1 TO 8
220   OVER = 2* KOLOR - 2
230   FOR DOWN =0 TO 31
240     SET(OVER, DOWN, KOLOR)
250   NEXT DOWN
260 NEXT KOLOR

300 REM ** DO NOTHING LOOP
310 GOTO 310

```

20. Here are some variations of the horizontal stripe program. You, of course, can write similar variations of your vertical stripe program. Our variations show only changes in block 200. Blocks 100 and 300 are as before.

VARIATION 1

```
200 REM ** 8 WIDE STRIPES ON 1/2 SCREEN
210 FOR KOLOR=1 TO 8
220   DOWN = 2*KOLOR - 2
230   FOR OVER=0 TO 63
240     SET(OVER, DOWN, KOLOR)
250     SET(OVER, DOWN+1, KOLOR)
260   NEXT OVER
270 NEXT KOLOR
```

VARIATION 2

```
200 REM ** 8 NARROW STRIPES FULL SCREEN
210 FOR KOLOR=1 TO 8
220   DOWN = 4*KOLOR - 4
230   FOR OVER=0 TO 63
240     SET(OVER, DOWN, KOLOR)
250   NEXT OVER
260 NEXT KOLOR
```

VARIATION 3

```
200 REM ** 8 WIDE STRIPES FULL SCREEN
210 FOR KOLOR=1 TO 8
220   DOWN = 4*KOLOR - 4
230   FOR OVER=0 TO 63
240     SET(OVER, DOWN, KOLOR)
250     SET(OVER, DOWN+1, KOLOR)
260     SET(OVER, DOWN+2, KOLOR)
270     SET(OVER, DOWN+3, KOLOR)
280   NEXT OVER
290 NEXT KOLOR
```

21. Hmm . . . look at lines 240 through 270 in VARIATION 3, preceding frame. Could that be written as a FOR-NEXT loop? Of course. Complete VARIATION 4, below.

```

200 REM ** 8 WIDE STRIPES FULL SCREEN
210 FOR KOLOR=1 TO 8
220   DOWN = 4*KOLOR - 4
230   FOR OVER=0 TO 63

240     FOR X=____TO____

250       SET(OVER,____, KOLOR)

260     NEXT____

270   NEXT OVER
280 NEXT KOLOR

```

```

240   FOR X=0 TO 3
250     SET(OVER,DOWN+X, KOLOR)
260   NEXT X

```

22. Prepare your fingers to paint some stripes. The following program is your "paintbrush."

```

100 REM ** STRIPE 'PAINTBRUSH'
110 CLS 0

200 REM ** DIALOG WITH PAINTER
210 PRINT "@0, CHR$(30): PRINT "@0, ;
220 INPUT "DOWN,L,R,CLR"; DOWN L, R, CLR
*
300 REM ** PAINT HORIZONTAL STRIPE
310 FOR OVER=L TO R
320   SET(OVER, DOWN, CLR)
330 NEXT OVER

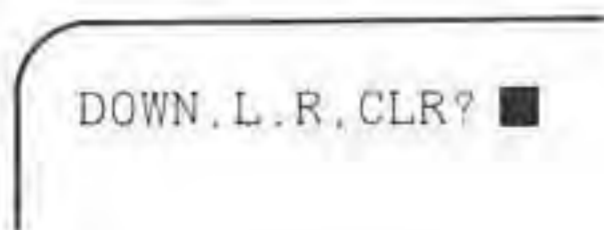
400 REM ** DONE. SOUND OFF.
410 SOUND 89, 10

500 REM ** GO BACK FOR MORE
510 GOTO 210

```

Aha! L is the Left end of the stripe. R is the Right end.

RUN the program. It begins with a black pallette . . . er. screen, that is . . . with a narrow "text window" at the top. Like this:



Yes, we admit it is a little cryptic. Try 12, 10, 23, 8.

```
DOWN,L,R,CLR? 12,10,23,8 ■
```

Before pressing ENTER.

Now press **ENTER**. This is what you see. Well, not quite—we have added some comments to the computer's handiwork.

```
DOWN,L,R,CLR?■
```

Just what we wanted! An orange stripe (CLR = 8) from OVER=L (where L = 10) to OVER=R (where R=23) at DOWN = 12.

The computer is waiting patiently for *your* values of DOWN,L,R, and CLR. Go ahead! Paint on the screen.

23. Here again is block 200 of the paintbrush program.

```
200 REM ** DIALOG WITH PAINTER
210 PRINT @0, CHR$(30); PRINT @0, ;
220 INPUT "DOWN,L,R,CLR": DOWN, L, R, CLR
```

What does line 210 tell the computer to do? _____

Line 210 contains two statements.

```
210 PRINT @0, CHR$(30); PRINT @0, ;
```

statement #1
statement #2

Statement #1 tells the computer to position the cursor at screen PRINT position 0, then erase everything to the right on that line. This clears the entire line to a nice mellow green.

Statement #2 positions the cursor at screen PRINT position 0. The computer then goes on to line 220 and executes the INPUT statement.

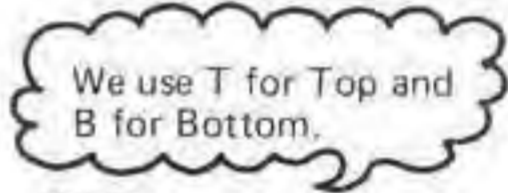
24. The program in frame 21 lets you paint horizontal stripes. Change it so you can paint vertical stripes. Rewrite blocks 200 and 300.

```
200 REM ** DIALOG WITH A PAINTER
```

```
300 REM ** PAINT VERTICAL STRIPE
```

```
200 REM ** DIALOG WITH A PAINTER
210 PRINT @0, CHR$(30); PRINT @0, ;
220 INPUT "OVER, T, B, CLR": OVER, T, B, CLR

300 REM ** PAINT A VERTICAL STRIPE
310 FOR DOWN=T TO B
320 SET(OVER, DOWN, CLR)
330 NEXT DOWN
```



Here is a more informative DIALOG WITH PAINTER that uses the "text window" four times, each time asking for *one* thing:

```
200 REM ** DIALOG WITH PAINTER

210 PRINT @0, CHR$(30); PRINT @0, ;
220 INPUT "HOW FAR DOWN": DOWN

230 PRINT @0, CHR$(30); PRINT @0, ;
240 INPUT "LEFT END OF STRIPE": L

250 PRINT @0, CHR$(30); PRINT @0, ;
260 INPUT "RIGHT END OF STRIPE": R

270 PRINT @0, CHR$(30); PRINT @0, ;
280 INPUT "WHAT COLOR": CLR
```

25. In frame 10, we showed you a program to put "The Big Dipper" on the screen. The Big Dipper has seven stars. Use FOR-NEXT along with READ-DATA to put a constellation on the screen.

```

100 REM ** A CONSTELLATION
110 CLS 0

200 REM ** NS IS NUMBER OF STARS
210 READ NS

300 REM ** TURN ON NS STARS
310 FOR STAR=1 TO NS
320   READ OVER, DOWN
330   SET(OVER, DOWN, 8)
340 NEXT STAR

400 REM ** DO NOTHING LOOP
410 GOTO 410

900 REM ** STAR DATA
910 DATA 7
920 DATA 6, 12, 18, 10 } Values of OVER and DOWN
930 DATA 26, 12, 34, 14 } for 7 stars.
940 DATA 38, 20, 54, 20 }
950 DATA 56, 14

```



How would you rewrite the DATA statements to put Draco on the screen? (See frame 11.)

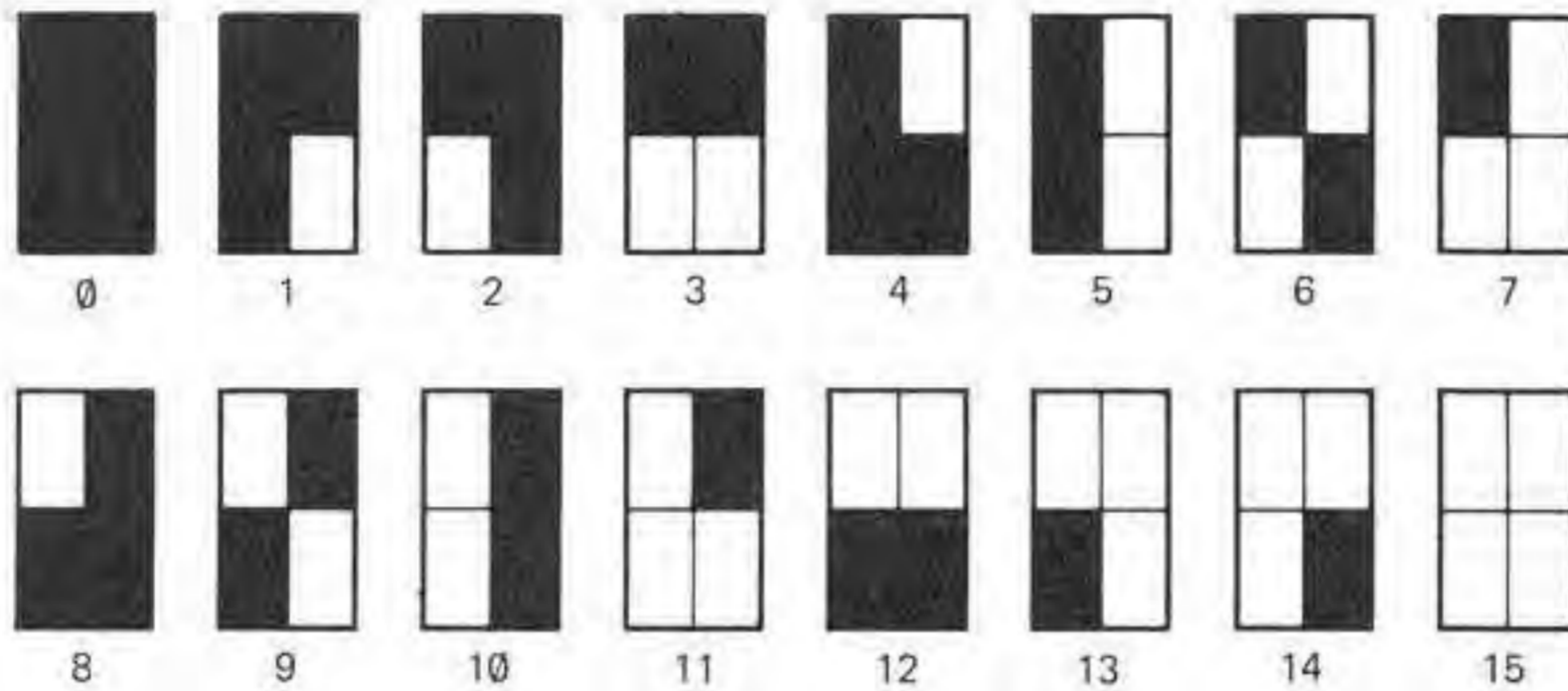
```

900 REM ** STAR DATA
-----
920 DATA 0, 8, 0, 14 } ← Value of NS 910 DATA 12
930 DATA 7, 13, 8, 9 } Values of OVER,DOWN
940 DATA 19, 24, 20, 20 } for 12 stars
950 DATA 22, 14, 24, 0 }
960 DATA 26, 28, 30, 8 }
970 DATA 42, 30, 60, 24 }

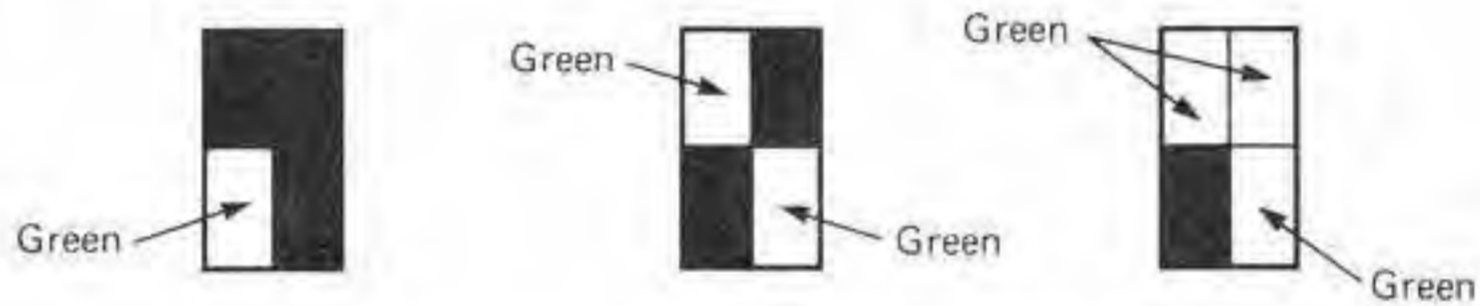
```

We put information for two stars in each DATA statement. If you wish, use fewer DATA statements and put more than two stars per DATA statement.

26. Remember, each PRINT position has four SET positions (see frame 5). Using one color plus black, you can make 16 distinct graphics shapes.



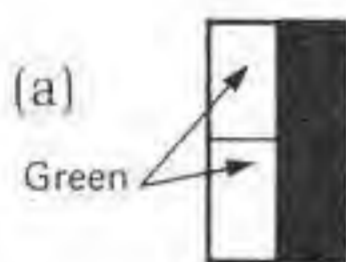
Choose a color. Can't decide? Choose green. Color the above shapes green. For example:



Using green, you get graphics characters 128 through 143.

$$\text{Graphics Character} = 128 + \text{Shape Number.}$$

What are the graphics character numbers for the following?



Draw the shape for each green graphics character.

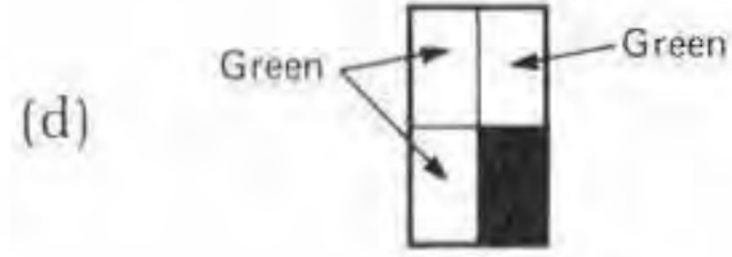
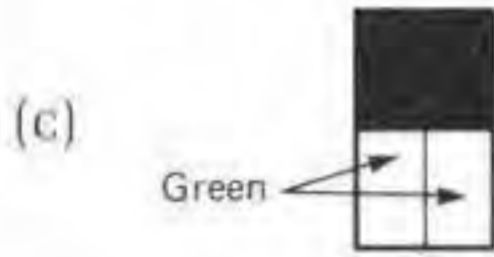
(c) 131



(d) 142

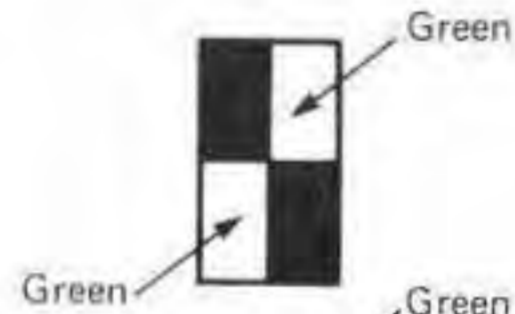


- (a) 138 (128 + 10)
- (b) 134 (128 + 6)

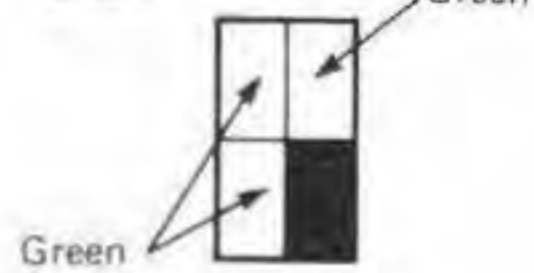


27. To put graphics character on the screen, use the CHR\$ function.

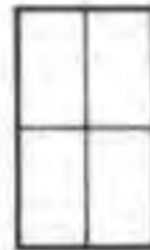
- CHR\$(134) is this character:



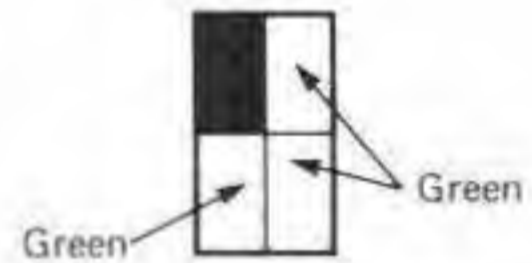
- CHR\$(142) is this character:



(1) What character is CHR\$(137)?



(2) Write a CHR\$ function for the character



(2) CHR\$(135)

28. Make your constellations more colorful. Rewrite the program called A CONSTELLATION (frame 25) so the computer reads the KOLOR of each star along with its galactic coordinates, OVER and DOWN. You need change only blocks 200 and 300.

```
300 REM ** TURN ON STARS
```

```
900 REM ** STAR DATA
```

```
-----

300 REM ** TURN ON STARS
310 FOR STAR=1 TO NS
320   READ OVER, DOWN, KOLOR
330   SET(OVER, DOWN, KOLOR)
340 NEXT STAR

900 REM ** STAR DATA
910 DATA 7
920 DATA 6, 12, 1, 18, 10, 2, green star, yellow star
930 DATA 26, 12, 3, 34, 14, 4 blue star, red star
940 DATA 38, 20, 5, 54, 20, 6 buff star, cyan star
950 DATA 56, 14, 7 magenta star
```

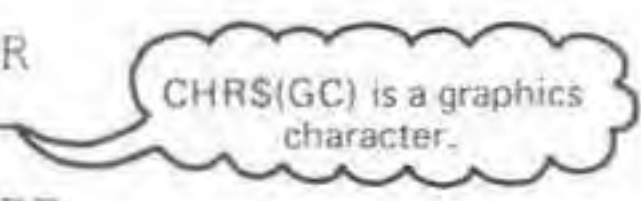
29. Use the following program to put some green graphics characters on the screen. Since it is somewhat difficult to see green on green or black on black, we use a cyan screen (line 110).

```
100 REM ** GRAPHICS CHARACTERS
110 CLS 6

200 REM ** TALK TO GRAPHICS WATCHER
210 PRINT @0, CHR$(30); PRINT @0, :
220 INPUT "GC, SP": GC, SP

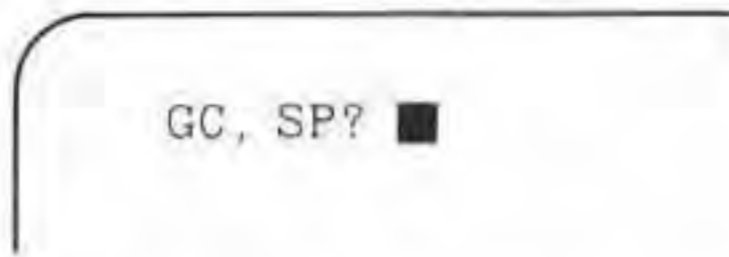
300 REM ** PRINT GRAPHICS CHARACTER
310 PRINT @SP, CHR$(GC);

400 REM ** SOUND OFF AND GO FOR MORE
410 SOUND 89, 1; SOUND 89, 1
420 GOTO 210
```

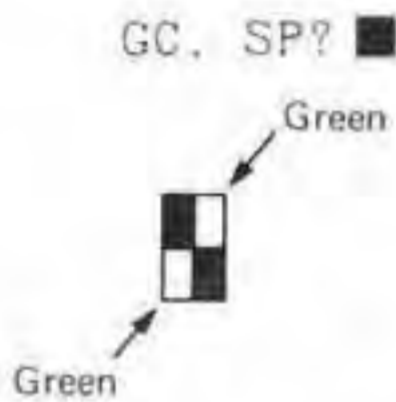


CHR\$(GC) is a graphics character.

RUN the program. It begins like this:



For GC, enter a number from 128 to 143. These are the codes for green graphics characters. For SP, we suggest any number from 32 to 510. SP means "Screen Position." We entered 134, 96, and pressed **ENTER**:



On a cyan screen.

EXPERIMENT! Try any number from 128 to 255 as the value of GC.

30. Yes, there are yellow graphics characters, blue graphics characters, and so on. Each has a graphics character code. Use `PRINT@` and `CHRS` to tell the computer to print a graphics character and where to print it.

Graphics Code (GC)	CHRS(GC)
128 to 143	Green character
144 to 159	Yellow character
160 to 175	Blue character
176 to 191	Red character
192 to 207	Buff character
208 to 223	Cyan character
224 to 239	Magenta character
240 to 255	Orange character

Hmmm. . . Here is a sneaky way to get the color of your choice. First, see frame 28 for the SHAPE number (0 to 15). Then pick a KOLOR (1 to 8). The graphics code (GC) is then:

$$GC = 128 + SHAPE + 16*(KOLOR - 1)$$

For example, suppose you want shape number 9 in red.

$$GC = 128 + 9 + 16*(4 - 1) = 185$$

Your turn. What is GC for shape number 12 in magenta?

$$GC = 128 + \underline{\quad} + \underline{\quad} = \underline{\quad}$$

$$GC = 128 + 12 + 16*(7 - 1) = 236$$

31. Use this program to put graphics characters on the screen. Try to build some pictures. Use a screen map to help you decide where to put each graphics character. This time, you are working on a black screen.

```

100 REM ** GRAPHICS CHARACTERS
110 CLS 0

200 REM ** TALK TO SHAPE MAKER
210 PRINT @0, CHR$(30): PRINT @0, :
220 INPUT "SHAPE (0 to 15)": SHAPE
230 PRINT @0, CHR$(30): PRINT @0, :
240 INPUT "COLOR (1 to 8)": KOLOR
250 PRINT @0, CHR$(30): PRINT @0, :
260 INPUT "WHERE (32 TO 510)": SP

300 REM ** PRINT GRAPHICS CHARACTER
310 GC = 128 + SHAPE + 16*(KOLOR - 1)
320 PRINT @SP, CHR$(GC):

400 REM ** SOUND OFF AND GO FOR MORE
410 SOUND 89, 1: SOUND 89, 1
420 GOTO 210
    
```

After you enter the shape number (line 220), color (line 240), and screen position (line 260), the computer prints your graphics character on the screen. This is done by line 320.

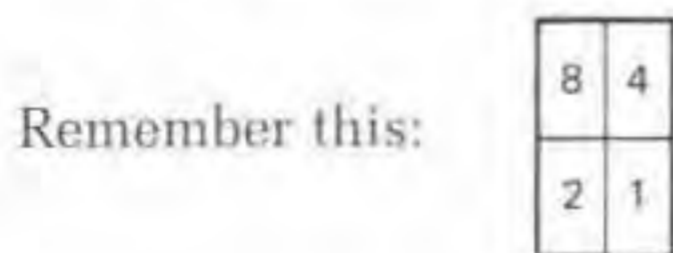
```

320 PRINT @SP, CHR$(GC);
                                        
                    one graphics character
    
```

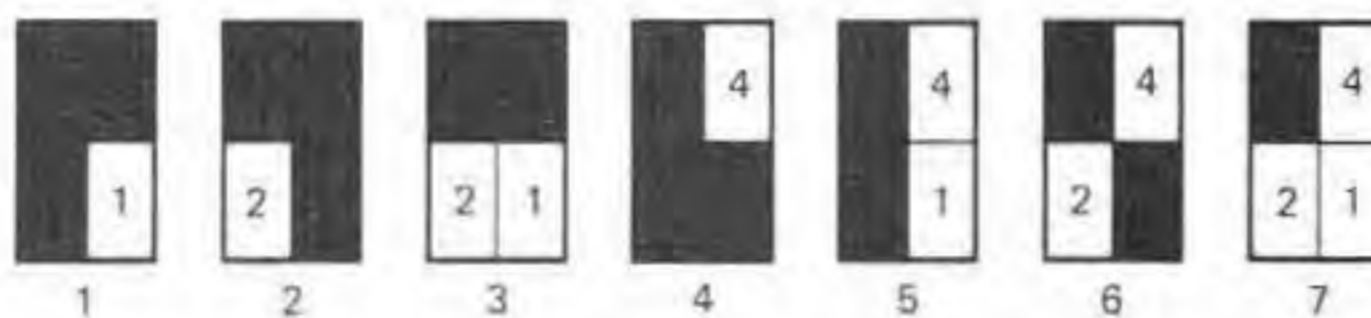
You will see more of CHR\$ as you read on through this book, especially in Chapter 10.

EXPERIMENT with this program. Make a green and yellow cat, an orange and blue flower, a many-colored dragon.

32. In case you haven't memorized the shapes and their numbers, don't bother. Here is an easy way to compute shape numbers.

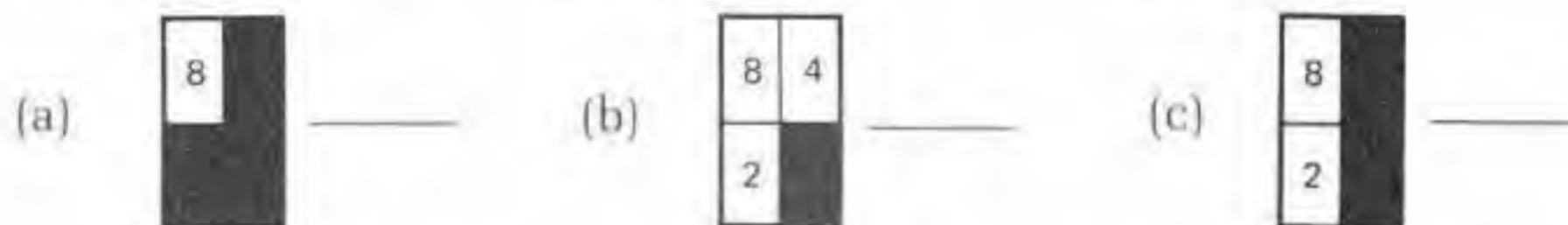


Here are shapes 1 through 7:

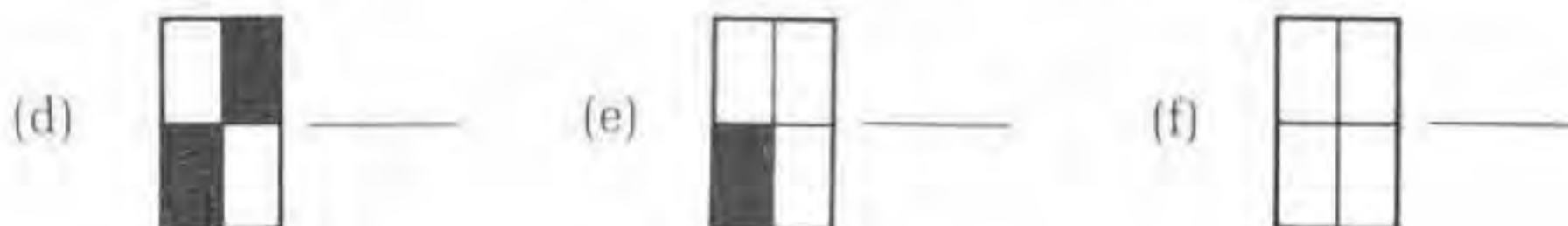


Got it? If not, note that $3 = 2 + 1$, $5 = 4 + 1$, and $7 = 4 + 2 + 1$. That's right, just add up the numbers in the blocks that aren't black.

Write the shape number for each graphics shape.



Now try it without the numbers, 1, 2, 4, and 8 marked in the boxes.



-
- (a) 8;
 - (b) $8 + 4 + 2 = 14$;
 - (c) $8 + 2 = 10$;
 - (d) $8 + 1 = 9$;
 - (e) $8 + 4 + 1 = 13$;
 - (f) $8 + 4 + 2 + 1 = 15$

Self-Test

During those final few frames, perhaps you were painting in anticipation of this colorful self-test. With keybrush in hand, try these questions.

1. There are two kinds of screen positions, or locations, called PRINT positions and SET positions.
 - (a) How many PRINT positions are there on the screen?_____
 - (b) How many SET positions are there on the screen?_____
 - (c) Each PRINT position contains (how many?)_____ SET positions.
 - (d) PRINT positions are numbered from ____ to ____.
 - (e) PRINT positions are arranged in (how many?)_____ rows and (how many?)_____ columns.
 - (f) SET positions are arranged in (how many?)_____ rows and (how many?)_____ columns.
 - (g) PRINT rows are numbered from ____ to ____.
 - (h) SET rows are numbered from ____ to ____.
 - (i) PRINT columns are numbered from ____ to ____.
 - (j) SET columns are numbered from ____ to ____.
2. Write SET statements to set PRINT position 337 to magenta. Remember, this PRINT position contains *four* SET positions.
3. Add SOUND to the inside FOR-NEXT loop in frame 15. This loop is shown below.

```

230 FOR OVER =0 TO 63
240   SET(OVER,DOWN,KOLOR)
250 NEXT OVER

```

Make the tone number equal to COLOR times OVER. Use 1 for the duration number.

4. Write a program to paint stripes on the screen. Use NS to mean Number of Stripes. READ NS from DATA statement. Then, for each stripe, READ the values of DOWN, L, R, and CLR and paint the stripe. Here is an outline of the program and some DATA to use when you try your program:

```

100 REM ** READ-DATA PAINTBRUSH
110 CLS 0

200 REM ** NS IS NUMBER OF STRIPES

300 REM ** PAINT NS STRIPES

400 REM ** DO NOTHING LOOP

700 REM ** STRIPE DATA
710 DATA 17
720 DATA 28, 0, 63, 3, 29, 0, 63, 3
730 DATA 30, 0, 63, 3, 31, 0, 63, 3
740 DATA 26, 24, 43, 5, 27, 24, 44, 5
750 DATA 25, 34, 34, 4, 24, 34, 34, 4
760 DATA 23, 26, 34, 4, 22, 27, 24, 4
770 DATA 21, 28, 34, 4, 20, 29, 34, 4
780 DATA 19, 30, 34, 4, 18, 31, 34, 4
790 DATA 17, 32, 34, 4, 16, 33, 34, 4
800 DATA 15, 34, 34, 4

```

5. Write a program to blink a graphics character on and off near the center of the screen. Your character should blink on a black screen. A RUN might go like this:

```
GC (128 TO 255)? 182 ■
```

Before pressing ENTER.

Now press ENTER.

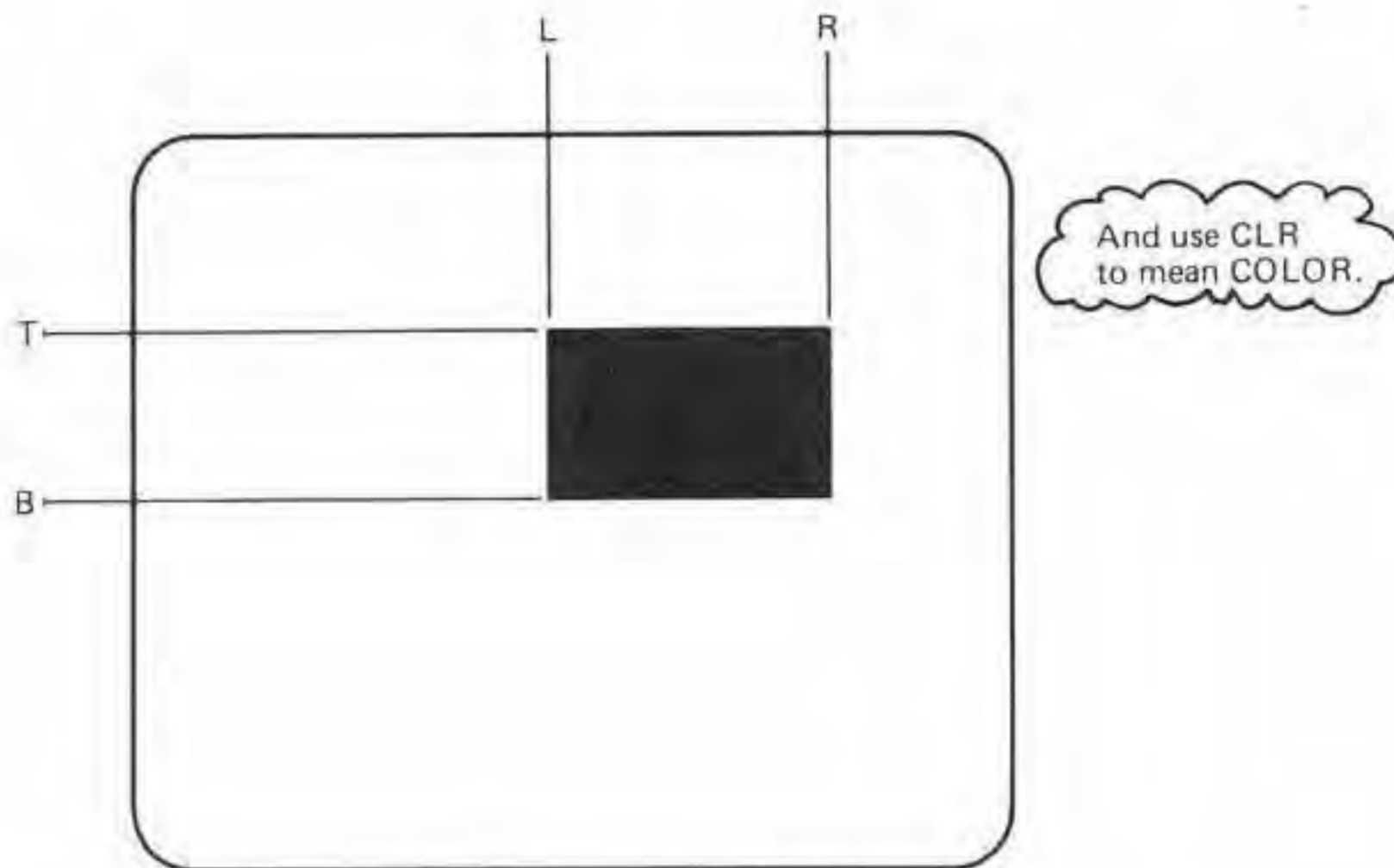
```

  Red
  □
Red □

```

This blinks on a black screen.

6. The screen sketch below shows a rectangle covering the portion of the screen from DOWN = T to DOWN = B and OVER = L to OVER = R (T for Top, B for Bottom, L for Left, and R for Right).



Complete the following program to paint rectangles.

```

100 REM ** PAINT RECTANGLES
110 CLS 0

200 REM ** DIALOG WITH USER
210 PRINT @0, CHR$(30); PRINT @0, ;
220 INPUT "T,B,L,R,CLR"; T, B, L, R, CLR

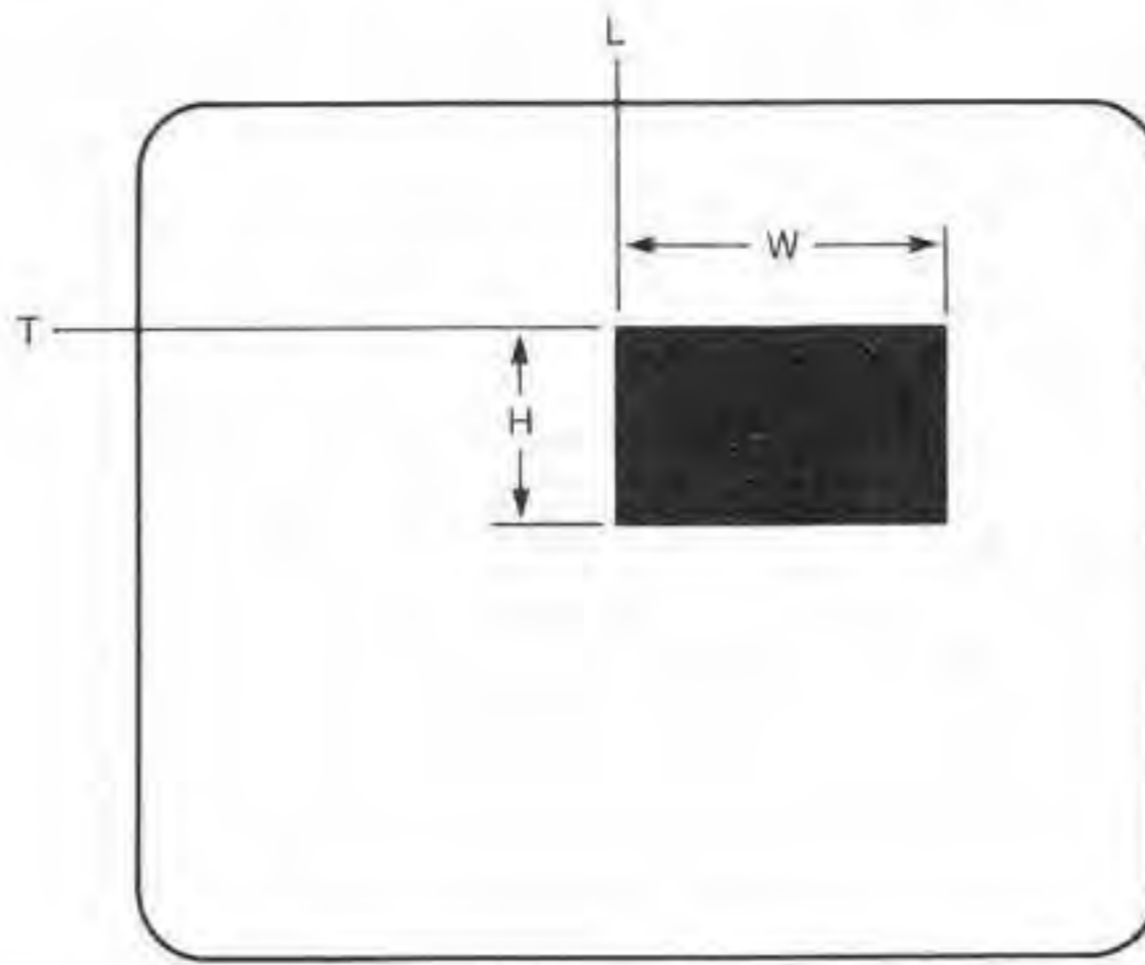
300 REM ** PAINT ONE RECTANGLE

400 REM ** DONE. SOUND OFF
410 SOUND 89, 10

500 REM ** GO BACK FOR MORE
510 GOTO 210
    
```

When you complete this program, use it to “draw” shapes on the screen.

7. Here is another way to define a rectangle on the screen.



In this diagram

L and T are the coordinates of the Left, Top corner of the rectangle.
H is the Height of the rectangle.
W is the Width of the rectangle.
CLR is the Color of the rectangle.

Modify your program of question 6 so the computer asks for and uses L, T, W, H, and CLR instead of T, B, L, R, and CLR.

8. Modify your program for question 6 so the computer asks for one thing at a time using only the top line ("text window") of the screen. It should ask the following things in the order shown.

(1) TOP?■

(2) BOTTOM?■

(3) LEFT?■

(4) RIGHT?■

(5) COLOR?■

9. Rewrite your program of question 6 so that information for NR rectangles is READ from DATA statements. For each rectangle, the computer should READ values of T,B,L,R, and CLR, then draw the rectangle.

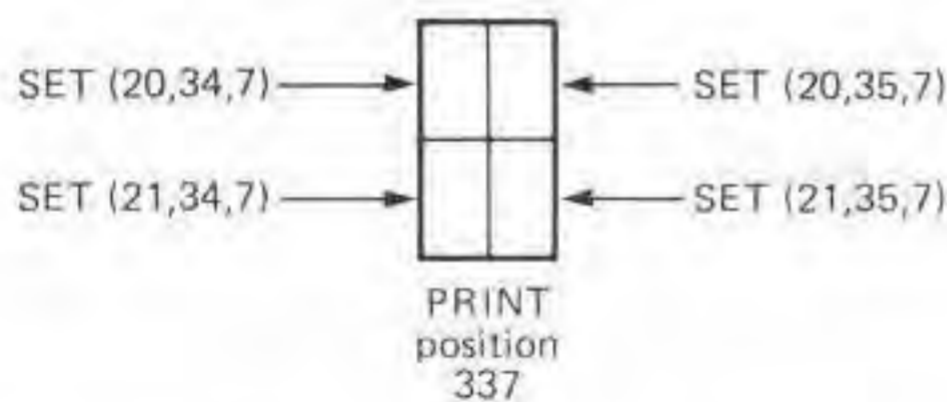
With suitable DATA statements, you can "paint" very interesting pictures on the screen. Use screen maps to plan your artwork.

10. Rewrite your program of question 6 so the computer READs information for NR rectangles from DATA statements.

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. (a) 512
 (b) 2048
 (c) 4
 (d) 0; 511
 (e) 16;32
 (f) 32;64
 (g) 0; 15
 (h) 0; 31
 (i) 0; 31
 (j) 0; 63 (frames 1, 2, 5)
2. SET(20,34,7): SET(20,35,7)
 SET(21,34,7): SET(21,35,7) (frames 3, 4, 5)



3. 245 SOUND COLOR*OVER, 1
 (frame 15)

```
4. 200 REM ** NS IS NUMBER OF STRIPES
    210 READ NS

    300 REM ** PAINT STRIPES
    310 FOR STRIPE = 1 TO NS
    320   READ DOWN, L, R, CLR
    330   FOR OVER=L TO R
    340     SET(OVER,DOWN,CLR)
    350     SOUND CLR*OVER, 1
    360   NEXT OVER
    370 NEXT STRIPE

    400 REM ** DO NOTHING LOOP
    410 GOTO 410
```

(frames 21, 22)

5. We did it like this:

```
    100 REM ** BLINK GRAPHICS CHARACTER

    200 REM ** ASK FOR GRAPHICS CODE
    210 CLS
    220 INPUT "GC(128 TO 255)"; GC

    300 REM ** BLINK IT ON
    310 CLS 0
    320 PRINT (@271, CHR$(GC));
    330 GOSUB 610

    400 REM ** BLINK IT OFF
    410 PRINT (@271, CHR$(128));
    420 GOSUB 610

    500 REM ** GO BLINK AGAIN
    510 GOTO 320

    600 REM ** TIME DELAY SUBROUTINE
    610 Z = 100
    620 FOR K = 1 TO Z: NEXT K
    630 RETURN
```

(frames 13, 26-32)

6-10. No answers. We recommend that you try some of these. Once you have them working, you and your friends can have lots of fun putting color patterns and pictures on the screen. If you get completely boggled, yell for help: Send a self-addressed stamped envelope to:

The Dymax Gazette
P.O. Box 310
Menlo Park, CA 94025

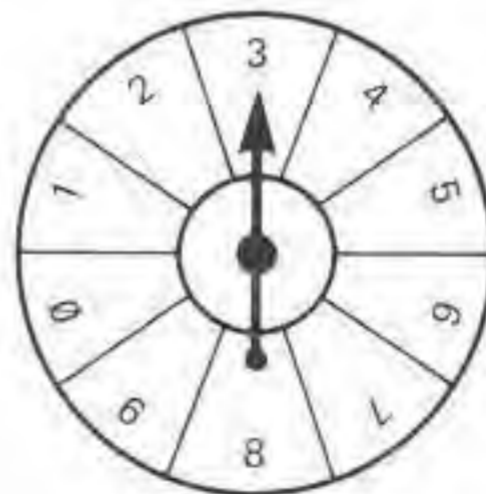
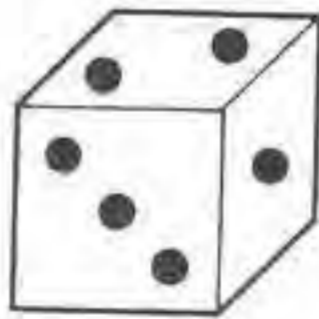
CHAPTER EIGHT

Meandering

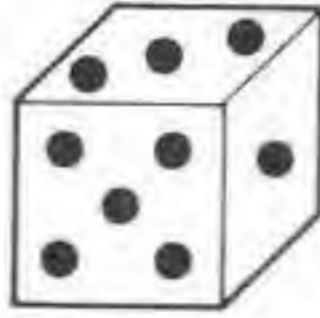
In this chapter, you will learn how to use the mysterious and unpredictable RND function. You will use the RND function to compute *random numbers* and use them in programs to surprise and delight yourself and your friends. With RND, your programs can take on an air of friendly unpredictability, so people will wonder . . . what next?

When you finish this chapter, you will be able to:

- Use the RND function to compute random whole numbers;
- Tell the computer to make random music that sounds awful;
- Make names and other strings or characters meander aimlessly about the screen;
- Invent your own random constellations (*you* get to name them);
- Splash random blips of color here, there, anywhere on the screen;
- Create ever-changing symmetric mandalas;
- Put random color stripes or rectangles on the screen (could this be pop art?);
- Put random graphics characters in random places on the screen;
- Create adventurers for games such as *Dungeons and Dragons*, *Runequest*, and *Tunnels and Trolls*.



1. Random numbers are numbers that are chosen at random from a given set of numbers. A die has 6 sides, numbered 1 through 6. Roll one die to get a random number from 1 to 6.



Flip a coin—two possibilities: heads or tails. If it comes up HEADS, call it 1; if it comes up TAILS, call it 2. Flipping a coin gets a random number from 1 to 2.

In Color BASIC, when you want a random number, use the *RND function*. RND is an abbreviation for RanNDom number. The following program uses the RND function to print random numbers on the screen.

```
10 CLS
20 PRINT RND(2) ← Here is the RND function.
30 GOTO 20
```

We ran the program twice, stopping each RUN by pressing the **BREAK** key. Here is what happened:

first RUN

```

2
2
1
1
2
2
2
2
BREAK IN 20
OK
■
```

second RUN

```

1
1
1
2
2
1
2
1
1
BREAK IN 20
OK
■
```

Two runs are shown. Did the second run produce the same list of numbers as the first run? _____

No. In fact, don't expect to enter our program into your TRS-80, type RUN, and get either list. That's the idea of random numbers. They are, well, random!



2. The program in the previous frame prints a list of random numbers. Each random number is 1 or 2. Why? Because line 20 tells the computer to PRINT RND(2).

RND(2) is a random number, 1 or 2.
 RND(3) is a random number, 1 or 2 or 3.
 RND(4) is a random number, 1 or 2 or 3 or 4.

In general, if N is a whole number greater than zero, then RND(N) will be a random number in the range 1 to N , inclusive.

RND(5) is a random number in the range 1 to 5, inclusive.

Your turn. Complete the following:

- (a) RND(6) is a random number in the range _____ to _____.
 (b) RND(10) is a random number in the range _____ to _____.
 (c) RND(100) is a random number from _____ to _____.

 (a) 1, 6; (b) 1, 10; (c) 1, 100

3. Since RND(2) is 1 or 2, you can probably guess that RND(2)-1 is 0 or 1.

RND(3) is 1, 2, or 3.
 RND(3)-1 is 0, 1, or 2.

RND(4) is 1, 2, 3, or 4.
 RND(4)-1 is 0, 1, 2, or 3.

RND(64) is a random number from 1 to 64.
 RND(64)-1 is a random number from 0 to 63.

Your turn. Complete the following.

- (a) $\text{RND}(10)-1$ is a random number from _____ to _____.
 (b) $\text{RND}(16)-1$ is a random number from _____ to _____.

(a) 0, 9; (b) 0, 15

4. Instead of a number, you can put a numeric variable in parentheses following the word RND.

The function: $\text{RND}(N)$

gives a random number from 1 to N .

If $N = 2$, then $\text{RND}(N)$ will be 1 or 2.

If $N = 3$, then $\text{RND}(N)$ will be 1, 2, or 3.

If $N = 6$, then $\text{RND}(N)$ will be 1, 2, 3, 4, 5, or 6.

Complete the following.

- (a) If $N = 4$, then $\text{RND}(N)$ will be _____.
 (b) If $N = 100$, then $\text{RND}(N)$ will be a random number in the range _____ to _____, inclusive.

(a) 1, 2, 3, or 4; (b) 1, 100

Any numeric variable may be used in the RND function.

If $A = 3$, then $\text{RND}(A)$ is 1, 2, or 3.

If $Z = 7$, then $\text{RND}(Z)$ is 1, 2, 3, 4, 5, 6, or 7.

5. Experiment. Use the following program to print random numbers from 1 to N , where you supply the value of N .

```

100 REM ** RANDOM NUMBERS, 1 TO N
200 REM ** DIALOG WITH USER
210 CLS
220 PRINT "I'LL PRINT RANDOM NUMBERS FROM"
230 PRINT "1 TO N. YOU ENTER VALUE OF N."
240 PRINT
250 INPUT "N = "; N

300 REM ** PRINT THE RANDOM NUMBERS
310 CLS
320 PRINT RND(N),
330 GOTO 320
  
```

;

The comma tells the computer to print two numbers per line.



Enter the program, type RUN, and press the **ENTER** key.

This is what you see. →

```
I'LL PRINT RANDOM NUMBERS FROM
1 TO N. YOU ENTER VALUE OF N.
N = ?■
```

Type your value of N and press **ENTER**. The Color Computer will print random numbers, two on each line, until you press **BREAK**.

Try values of N, including these:

- N = 2.9 The computer ignores the .9 and prints only the numbers 1 and 2.
- N = 0 What? Instead of whole numbers, you get decimals such as .249744686 or .871621619. In this book, we will not use this type of random number.

6. OK, so RND gives random numbers. But what are they good for? How about some random music?

```
100 REM ** RANDOM MUSIC
110 CLS
120 T = RND(255)
130 SOUND T, 10
140 GOTO 120
```

SOUND T, 10

↖
Tone number.

↖
Duration number.

- (a) In line 120, what are the possible values of T? _____
- _____
- (b) Where else is the variable T used? _____
- _____
- (c) What will happen when you RUN the program? _____
- _____

- (a) Random whole numbers in the range 1 to 255.
- (b) In the SOUND statement in line 130. So, the SOUND will be a “random sound.”
- (c) You will hear music (???) consisting of a succession of random tones. Sounds awful, doesn't it! Later you will learn how to make random music that sounds better. To eliminate scratchy high tones, change line 120 to:

```
120 T = RND(185)
```

↑
or your choice

7. Write a program to play random tones (T) with random duration (D). Values of D should be random numbers from 1 to 10.

```

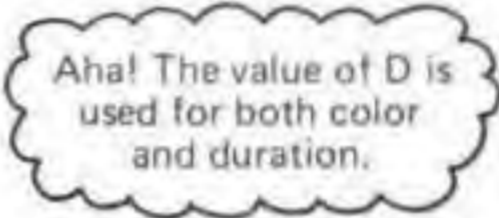
100 REM ** RANDOM MUSIC
110 CLS
120 T = RND(255)
130 D = RND(10)
140 SOUND T, D
150 GOTO 120
    
```

VARIATION: Change line 120 to 120 T=RND(185). This will eliminate those scratchy high notes. Also try numbers other than 185 in line 120.

8. Add color changes. Let the color correspond to the duration of the tone.

```

100 REM ** RANDOM MUSIC
110 T = RND(185)
120 D = RND(8)
130 CLS D
140 SOUND T, D
150 GOTO 110
    
```



- (a) In line 120, what are the possible values of D? _____
- (b) What "color" is the shortest possible tone? _____
 Longest possible tone? _____

- (a) 1,2,3,4,5,6,7, or 8;
- (b) green; orange

VARIATION: Reverse the correspondence between color and duration. Change line 130 to 130 CLS(9 - D).

9. Use random numbers to put a name here, there, anywhere on the screen. Here for a moment, there for a moment, somewhere else for a moment.

```

100 REM ** MEANDERING NAME
110 N$ = "KARL"

200 REM ** RANDOM SCREEN POSITION
210 SP = RND(511)

300 REM ** PRINT NAME IN RANDOM PLACE
310 CLS
320 PRINT @SP,N$:

400 REM ** VARIABLE TIME DELAY
410 Z = 450
420 FOR K = 1 TO Z: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 210

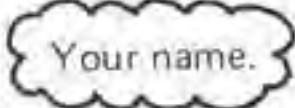
```

(a) In line 210, what are the possible screen positions?_____

(b) How can you change the program so *your* name is printed instead of Karl's name?_____

(c) How can you change the program so that, each time the name appears, it is printed on a randomly selected screen color?_____

(a) 1, to 511. Position 0 will not be used.

(b) Change line 110 to: 110 N\$ = "  "

(c) Changes: 220 C = RND(8)
310 CLS C

10. Make it easy to change the name that cavorts about the screen. Modify the program in the previous frame so that the computer asks for the name. When RUN, your program should start like this:

```

WHAT IS YOUR NAME? ■

```

Show your changes below:

We did it this way:

```
110 CLS
120 INPUT "YOUR NAME"; N$
```

Try this variation:

```
100 REM ** MEANDERING NAME
110 CLS
120 INPUT "YOUR NAME"; N$
130 CLS

200 REM ** RANDOM SCREEN POSITION
210 SP = RND(511)

300 REM ** PRINT N$ AT SP
310 PRINT @SP, N$:

400 REM ** VARIABLE TIME DELAY
410 Z = 10
420 FOR K=1 TO Z: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 210
```

11. This program will put N stars on the screen in random positions. They wait a few seconds while you stargaze, then start over.

```

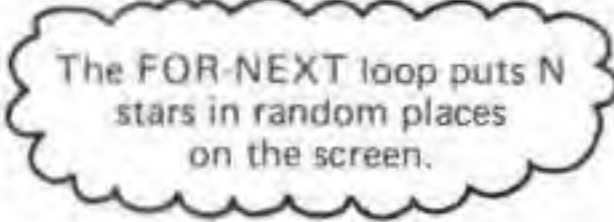
100 REM ** RANDOM STARS
200 REM ** TALK WITH USER
210 CLS
220 INPUT "HOW MANY STARS": N

300 REM ** PUT N STARS IN RANDOM PLACES
310 CLS 3
320 FOR STAR=1 TO N
330   SP = RND(510)
340   PRINT@SP, "*";
350 NEXT STAR

400 REM ** TIME DELAY
410 Z = 3000
420 FOR K=1 TO Z: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 210

```



The FOR-NEXT loop puts N stars in random places on the screen.

The FOR-NEXT loop could also be written as follows:

```

320 FOR STAR=1 TO N
330   PRINT (@RND(510), "*");
340 NEXT STAR

```

We increased the time delay by changing line 410 to: 410 Z = 30000 (more than one minute). Then we ran the program and entered 100 as the value of N. During the time delay, we counted the stars on the screen. There were only 97 stars.

Explain how we might see fewer stars than the number we asked for (it's OK to guess).

Each star is printed in a random place, 1 to 510, on the screen. It is possible that a star is printed in a place already occupied by a star. In this case, the new star replaces the old star.

For example, suppose the third star (STAR = 3) was printed in position 235. Then, later on, the thirty-seventh star (STAR = 37) was also printed in position 235. Although two stars had been printed in position 235, you would see only one.

12. How about some random color blips? This program puts random color blips at random places on a black screen:

```
100 REM ** RANDOM COLOR BLIPS
110 CLS 0

200 REM ** COLOR, OVER, AND DOWN ARE RANDOM
210 KOLOR = RND(8)
220 OVER = RND(64) - 1
230 DOWN = RND(32) - 1

300 REM ** PUT BLIP ON SCREEN
310 SET(OVER, DOWN, KOLOR)

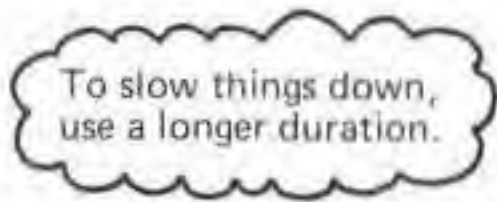
400 REM ** DO IT AGAIN
410 GOTO 210
```

- (a) In line 210, what are the possible values of KOLOR? _____
(b) In line 220, what are the possible values of OVER? _____
(c) In line 230, what are the possible values of DOWN? _____
-

- (a) 1,2,3,4,5,6,7, and 8.
(b) 0 to 63, inclusive. RND(64) gives 1 to 64, so RND(64) - 1 gives 0 to 63.
(c) 0 to 31, inclusive. RND(32) gives 1 to 32, so RND(32) - 1 gives 0 to 31.

Add some random sound. Try these;

```
240 T = RND(216)
320 SOUND T, 1
```



To slow things down,
use a longer duration.

13. A mandala is a symmetric pattern, nice to look at. A giant snowflake is beautifully symmetric about its center. Snowflakes are great mandalas but melt all too soon. Use this program to put an everchanging mandala on the screen:

```

100 REM ** MANDALA, EVER CHANGING
110 CLS 0

200 REM ** HORIZONTAL & VERTICAL OFFSET
210 H = RND(32) - 1
220 V = RND(16) - 1

300 REM ** RANDOM COLOR
310 KOLOR = RND(8)

400 REM ** TURN ON 4 BLIPS
410 SET(31 - H, 15 - V, KOLOR)
420 SET(31 - H, 16 + V, KOLOR)
430 SET(32 + H, 15 - V, KOLOR)
440 SET(32 + H, 16 + V, KOLOR)

500 REM ** DELAY, THEN DO MORE
510 Z = 10
520 FOR K=1 TO Z: NEXT K
530 GOTO 210

```

RUN the program. The computer turns on *four* lights at a time, symmetric about the center of the screen. If you don't see this happen, increase the time delay. Change line 510 to:

```
510 Z = 500
```

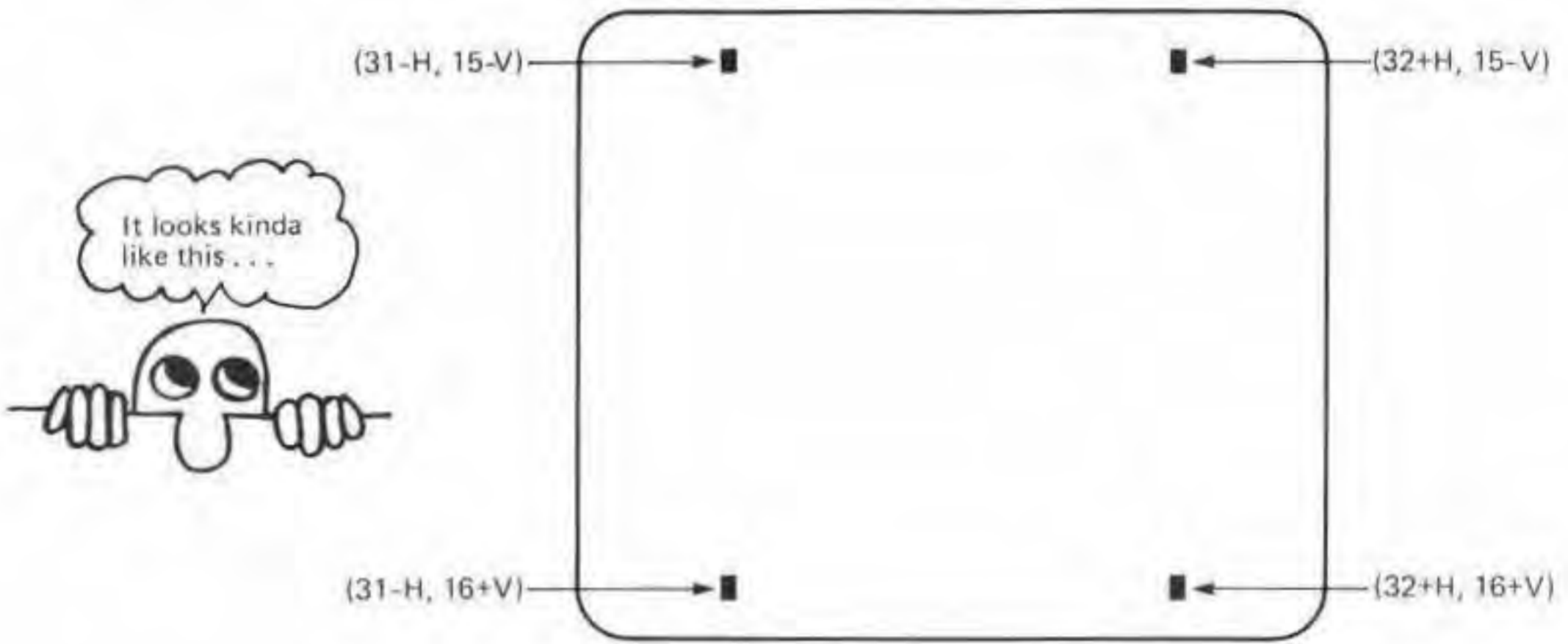
and RUN the program again. If you want the mandala to change more rapidly, delete lines 510 and 520, or change line 510 to: 510 Z = 1.

14. Lines 210 and 220 compute random values for H (horizontal offset) and V (vertical offset).

- What are the possible values of H? From ____ to ____.
- What are the possible values of V? From ____ to ____.
- What are the possible values of $31 - H$? From ____ to ____.
- What are the possible values of $32 + H$? From ____ to ____.
- What are the possible values of $15 - V$? From ____ to ____.
- What are the possible values of $16 + V$? From ____ to ____.
- Suppose, just suppose, that H is 17 and V is 12. Where will the four blips appear?
 (____,____), (____,____), (____,____), and (____,____)

(a) 0; 31 (b) 0; 15 (c) 0; 31 (d) 32; 63 (e) 0; 15 (f) 16; 31
 (g) (14,3) (14,28), (39,3), and (49,28)

Line 410 Line 420 Line 430 Line 440
 (31-H, 15-V)
 (32+H, 15-V)
 (31-H, 16+V)
 (32+H, 16+V)



15. Experiment! Try these variations:

VARIATION 1. Change *only* line 210, as follows:

```
210 H = RND(RND(32)) - 1
```

VARIATION 2. Change *only* line 220, as follows:

```
220 V = RND(RND(16)) - 1
```

VARIATION 3. Change both lines 210 and 220, as follows:

```
210 H = RND(RND(32)) - 1
220 V = RND(RND(16)) - 1
```

VARIATION 4. Change either line 210 or line 220, or both, as follows:

```
210 H = RND(RND(RND(32))) - 1
220 V = RND(RND(RND(16))) - 1
```

VARIATION 5. Change either line 210, or line 220, or both:

```
210 H = 32 = RND(RND(32))
220 V = 16 = RND(RND(16))
```


VARIATION 6. Change line 310:

```
310 KOLOR = RND(RND(8))
```

VARIATION 7. Anything suggested by the above variations.

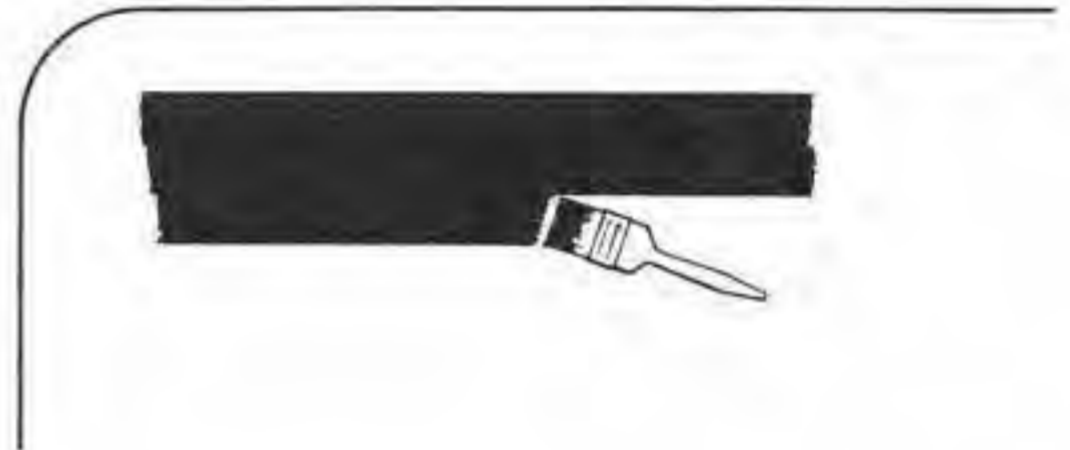
16. Put random horizontal stripes across the screen.

```
100 REM ** RANDOM STRIPES
110 CLS 0

200 REM ** RANDOM COLOR AND DOWN
210 KOLOR = RND(8)
220 DOWN = RND(32) - 1

300 REM ** PAINT HORIZONTAL STRIPE
310 FOR OVER = 0 TO 63
320   SET(OVER, DOWN, KOLOR)
330 NEXT OVER

600 REM ** DO IT AGAIN
610 GOTO 210
```



RUN this program; then add random vertical stripes:

```
400 REM ** RANDOM COLOR AND OVER
```

```
500 ** PAINT VERTICAL STRIPE
```

```
400 REM ** RANDOM COLOR AND OVER
410 KOLOR = RND(8)
420 OVER = RND(64) - 1

500 REM ** PAINT VERTICAL STRIPE
510 FOR DOWN=0 TO 31
520   SET(OVER, DOWN, KOLOR)
530 NEXT DOWN
```



17. Add sound. Try one of these:

```
325 SOUND 4*OVER + 1, 1
525 SOUND 255 - 8*DOWN, 1
```

(a) In line 325, since OVER goes from 0 to 63, the tone number will go from ____ to ____.

(b) In line 525, since DOWN goes from 0 to 31, the tone number will go from ____ to ____.

(a) 1; 252 (b) 255; 7

Also try these SOUND statements.

```
325 SOUND KOLOR*OVER/2 + 1, 1
525 SOUND 255 - KOLOR*DOWN, 1
```

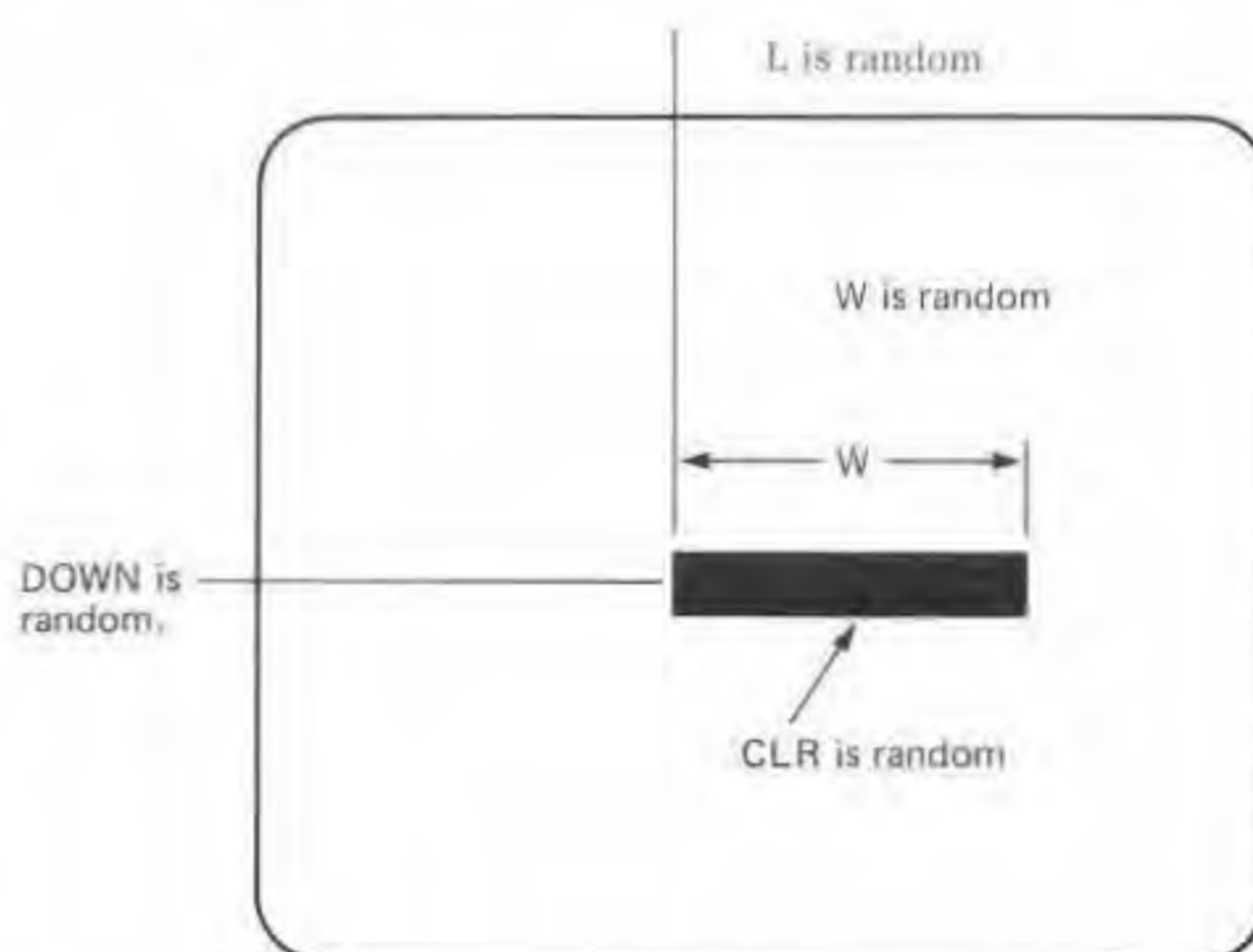
In line 325, $KOLOR*OVER/2$ means KOLOR times OVER, divided by 2.

In line 525, $255-KOLOR*DOWN$ means: Multiply DOWN by KOLOR, then subtract the result from 255.

And, what might happen if you then made these changes? Try them and find out. Experiment!

```
310 FOR OVER=0 TO 63 STEP 2
510 FOR DOWN=0 TO 31 STEP 2
```

18. This one is a little tricky, so read carefully! We want to put random horizontal stripes of *random width* on the screen.



```

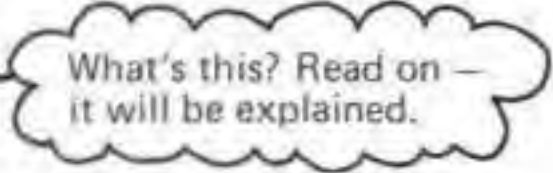
100 REM ** RANDOM STRIPES
110 CLS 0

200 REM ** RANDOM VALUES
210 KOLOR = RND(8)
220 DOWN = RND(32) - 1
230 L = RND(63) - 1
240 W = RND(63 - L)

300 REM ** PAINT HORIZONTAL STRIPE
310 FOR OVER=L TO L + W
320   SET(OVER, DOWN, KOLOR)
330 NEXT OVER

600 REM ** DO IT AGAIN
610 GOTO 210

```



What's this? Read on —
it will be explained.

In line 230, the possible values of L are 0 to 62. This, of course, is the left end of the stripe. In line 240, the number $63 - L$ is the distance from the left end of the stripe to the right edge of the screen. So, W will be a random number from 1 to $63 - L$. This insures that the Computer doesn't try to paint beyond the right edge of the screen. Instead, it paints (line 310) from L to L+W.

What would happen if the computer tried to paint beyond the right edge of the screen? _____

It would stop with an error message:

```
?FC ERROR IN 320
```

VARIATION: Slow down the action. Put a time delay between line 330 and line 600.

19. In Chapter 7, you learned about 16 *graphics characters* (shapes) available in 8 different colors. Graphics characters have ASCII codes from 128 to 255. Here is a program to put random graphics characters at random screen positions:

```
100 REM ** RANDOM GRAPHICS CHARACTERS
110 CLS 0

200 REM ** RANDOM CHARACTER & SCREEN POSITION
210 GC = RND(128) + 127
220 SP = RND(511) - 1

300 REM ** PUT CHARACTER ON SCREEN
310 PRINT (@SP, CHR$(GC));

400 REM ** TIME DELAY
410 Z = 100
420 FOR K=1 TO Z: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 210
```

(a) In line 210, what are the possible values of GC? _____

(b) In line 220, what are the possible values of SP? _____

(a) 128 to 255, inclusive. RND(128) gives 1 to 128, so RND(128)+127 gives 128 to 255.

(b) 0 to 510, inclusive. As usual, we avoid screen position 511 because printing there causes everything on the screen to scroll up one line.

RUN this program and watch the graphics shapes appear. Change line 410 to make things happen faster or more slowly.

20. Complete the following program to put *two* graphics characters side by side near the center of the screen. Each pair of characters will be on screen for about one second.

```

100 REM ** TWO GRAPHICS CHARACTERS
110 CLS 0

200 REM ** TWO RANDOM GRAPHICS CHARACTERS

210 G1 = _____
220 G2 = _____

300 REM ** PUT THEM ON SCREEN

310 PRINT @329, _____

400 REM ** TIME DELAY
410 Z = 460
420 FOR K=1 TO Z: NEXT K      About one second.

500 REM ** DO IT AGAIN
510 GOTO 110

```

```

210 G1 = RND(128) + 127
220 G2 = RND(128) + 127
310 PRINT @239, CHR$(G1); CHR$(G2);

```

Line 310 can also be written in the following ways:

```

310 PRINT @239, CHR$(G1) CHR$(G2);

310 PRINT @239, CHR$(G1) + CHR$(G2);

```

VARIATION: Print the values of G1 and G2 as well as the two graphics characters. Then, if you see something you like on the screen, quickly write down the codes for the two characters.

21. Have you heard of *Dungeons and Dragons*, *Runequest*, or *Tunnels and Trolls*? These are fantasy role-playing games. If you haven't heard about these games, ask any kid about them. In the unlikely event that he or she can't enlighten you, start with this booklet:

Basic Role-Playing: An Introductory Guide, by Greg Stafford and Lynn Willis.
 From: Chaosium, Inc., P.O. Box 6302, Albany, CA 94706.

For more information, try these:

Dungeons and Dragons (D&D) from TSR Hobbies, P.O. Box 765
Lake Geneva, WI 53147.

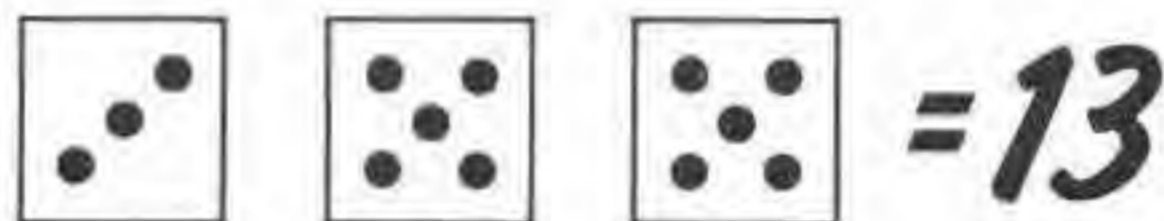
Runequest (RQ) from Chaosium, Inc., P.O. Box 6302, Albany, CA
94706.

Tunnels & Trolls (T&T) from Flying Buffalo, Inc., P.O. Box 1467,
Scottsdale, AZ 85252.

22. To play a role-playing game, you create one or more characters, then guide them through adventures in a universe created by a gamemaster. To create a character, you roll three six-sided dice several times. Each roll assigns a value to one of the basic characteristics of your character.

For starters, you will learn to use the computer to roll a *Runequest* character for the simple game described in *Basic Role-Playing*. Your character will have seven characteristics: strength (STR), intelligence (INT), power (POW), constitution (CON), dexterity (DEX), charisma (CHA), and size (SIZ). These characteristics determine a character's ability to use weapons, fight, learn, use magic, sustain damage, lead others, survive, and thrive while adventuring in the gamemaster's universe.

A six-sided die has six sides numbered 1 to 6. To determine a characteristic, you roll 3 dice and add the numbers showing on the individual dice. For example:



- (a) What is the smallest possible roll using 3 dice? _____
(b) What is the largest possible roll using 3 dice? _____

(a) 3 (1 + 1 + 1); (b) 18 (6 + 6 + 6)

An "average" roll is 10 or 11. The roll shown above (13) is above average.

23. Extra for people who know a little about probability. A roll of three dice can give any number from 3 to 18. However, the possible numbers are not *equally* possible. Here is a table showing the number of ways and probability for each number.

Roll	Number of Ways	Probability	
		Fraction	Decimal
3	1	1/216	.005
4	3	3/216	.014
5	6	6/216	.028
6	10	10/216	.046
7	15	15/216	.069
8	21	21/216	.097
9	25	25/216	.116
10	27	27/216	.125
11	27	27/216	.125
12	25	25/216	.116
13	21	21/216	.097
14	15	15/216	.069
15	10	10/216	.046
16	6	6/216	.028
17	3	3/216	.014
18	1	1/216	.005

What is the probability of a roll being in the range 9 to 12, inclusive?

.482 Rolls in this range give *near average* values of the characteristics. Expect to get a roll of 9 to 12 about half of the time.

24. Instead of rolling dice, use the Color Computer to roll a character. The following program will do it.

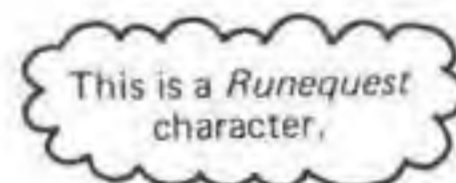
```

100 REM ** CREATE A CHARACTER
110 CLS

200 REM ** ROLL CHARACTERISTICS
210 GOSUB 310: PRINT "STR", DICE
220 GOSUB 310: PRINT "INT", DICE
230 GOSUB 310: PRINT "POW", DICE
240 GOSUB 310: PRINT "CON", DICE
250 GOSUB 310: PRINT "DEX", DICE
260 GOSUB 310: PRINT "CHA", DICE
270 GOSUB 310: PRINT "SIZ", DICE
280 PRINT
290 STOP

300 REM ** SUBROUTINE TO ROLL 3 DICE
310 D1 = RND(6)
320 D2 = RND(6)
330 D3 = RND(6)
340 DICE = D1 + D2 + D3
350 RETURN

```



Why STOP? Try the program without it!

This program calls the dice roll subroutine seven times. Look at the subroutine.

- (a) What are the possible values of D1? _____
- (b) What are the possible values of D2? _____
- (c) What are the possible values of D3? _____
- (d) What are the possible values of DICE? _____

- (a) 1,2,3,4,5, or 6.
- (b) 1,2,3,4,5, or 6.
- (c) 1,2,3,4,5, or 6.
- (d) Whole numbers in the range 3 to 18. Just what we wanted!

25. Use the program to roll a character. Your character might look like this (but probably won't):

STR	15
INT	10
POW	9
CON	12
DEX	14
CHA	7
SIZ	13
BREAK IN 290	
OK	
■	

We'll call him
Fibak the Fighter.

Here is your first adventurer. He is big (SIZ=13), strong (STR=15), above average in soaking up damage (CON=12), has good dexterity (D=14), has average intelligence (INT=10). He will not be a magic-user (POW=9) or a leader (CHA=7).

Looks like our character would be a good warrior.

26. Roll another character.

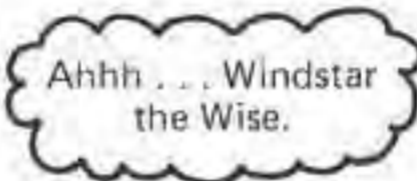
STR	14
INT	8
POW	15
CON	13
DEX	7
CHA	15
SIZ	14
BREAK IN 290	
OK	
■	

This has to be
Clutz the Charmed!

This is indeed a strange character! Big (SIZ=14), strong (STR=14), able to take damage (CON=13). But clumsy (DEX=7) and not very bright (INT=8). Look at power (POW=15). Very high. If our character had high intelligence, he could be a magic-user. However, he probably can't remember a spell and, even if he did, might use it on himself by accident. His power will show up as luck. Our character is very lucky.

Watch out! This character has high charisma (CHA=15). He will convince others to follow him into . . . what? A quandary . . . do we follow Clutz and trust to luck? Perhaps we follow a league behind. If we don't follow, does Clutz pick us up, put us under one arm and . . .?

27. Help! Roll another adventurer.

STR	10	
INT	17	
POW	15	
CON	12	
DEX	9	
CHA	16	
SIZ	11	
BREAK IN 290		
OK		
■		

Saved! Our group is saved! Windstar the Wise wandered by, saw our forlorn little group of adventurers, and decided to take charge. She is a magic-user and leader—someone to trust, learn from, and follow.

The group of characters now numbers three: FIBAK, CLUTZ, and WINDSTAR. Too small a group! They need at least four more adventurers to survive, and thrive, in the Gamemaster's World.

So, you roll four more characters. More are OK. Tell who each character is and how he or she relates to and works with the other adventurers.

28. In Tunnels and Trolls (T&T), a character has six characteristics: strength (STR), intelligence (IQ), luck (LK), constitution (CON), dexterity (DEX), and charisma (CHR). Modify the program in frame 14 so the computer "rolls" a T&T-character. Use the abbreviations shown above.

Change only block 200. We did it this way:

```

200 REM ** ROLL 6 CHARACTERISTICS
210 GOSUB 310: PRINT "STR", DICE
220 GOSUB 310: PRINT "IQ ", DICE
230 GOSUB 310: PRINT "LK ", DICE
240 GOSUB 310: PRINT "CON", DICE
250 GOSUB 310: PRINT "DEX", DICE
260 GOSUB 310: PRINT "CHR", DICE
270 PRINT
280 STOP

```

29. Here is a completely different program to roll a character. This one rolls a *Dungeons and Dragons* (D&D) character, which has six characteristics: strength (STR), intelligence (INT), wisdom (WIS), constitution (CON), dexterity (DEX), and charisma (CHA). Look for the abbreviations in the DATA statement, line 910.

```

100 REM ** CREATE A CHARACTER
110 CLS

200 REM ** ROLL THE CHARACTERISTICS
210 READ C$
220 GOSUB 310
230 PRINT C$, DICE
240 GOTO 210

300 REM ** SUBROUTINE TO ROLL 3 DICE
310 D1 = RND(6)
320 D2 = RND(6)
330 D3 = RND(6)
340 DICE = D1 + D2 + D3
350 RETURN

900 REM ** CHARACTERISTIC ABBREVIATIONS
910 DATA STR, INT, WIS, CON, DEX, CHA

```

When you RUN this program, it ends with an OD ERROR. That's OK, since all the desired information is on the screen.

How would you change the program to roll

- (a) a *Runequest* character, as in frame 21? _____
- (b) a *Tunnels and Trolls* character, as in frame 25? _____

-
- (a) Change line 910 to: 910 DATA STR, INT, POW, CON, DEX, CHA, SIZ
- (b) Change line 910 to: 910 DATA STR, IQ, LK, CON, DEX, CHR

Self-Test

Well, it looks as if you have meandered into another Self-Test.

1. For each of the following, what are the possible values of X?
 - (a) $X = \text{RND}(3)$ _____
 - (b) $X = \text{RND}(3) - 1$ _____
 - (c) $X = \text{RND}(3) - 2$ _____
 - (d) $X = \text{RND}(3) + 2$ _____
 - (e) $X = 2 * \text{RND}(3)$ _____
 - (f) $X = 2 * \text{RND}(3) - 1$ _____
 - (g) $X = 2 * (\text{RND}(3) - 1)$ _____
2. Complete the following
 - (a) If $N = 100$, then $\text{RND}(N) - 1$ will be a random number in the range _____ to _____, inclusive.
 - (b) If $N = 20$, then $\text{RND}(20) + 10$ will be a random number in the range _____ to _____, inclusive.
3. Complete each statement so X will be a random number in the range shown.

Range	Statement
(a) $X = 10, 11, 12, \text{ or } 13.$	$X =$ _____.
(b) $X = 2, 5, \text{ or } 8.$	$X =$ _____.

4. Where will the wandering question mark wander if you RUN the following program? _____

```

100 REM ** WANDERING CHARACTER
110 CH$ = "?"

200 REM ** RANDOM SCREEN POSITION
210 SP = RND(32) - 1

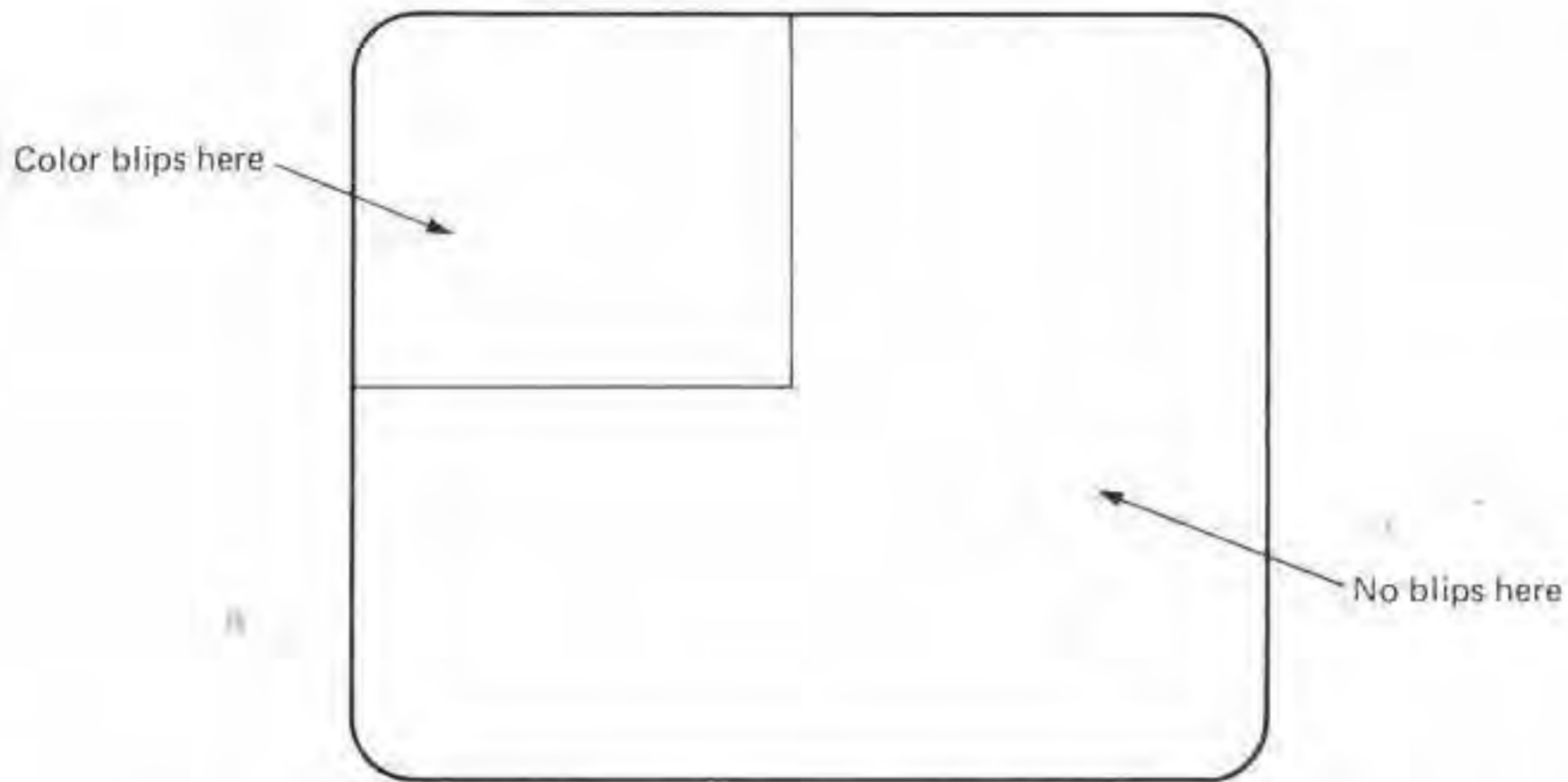
300 REM ** PRINT CH$ IN RANDOM PLACE
310 CLS
320 PRINT @SP, CH$:

400 REM ** TIME DELAY
410 Z = 100
420 FOR K=1 TO Z: NEXT K

500 REM ** DO IT AGAIN
510 GOTO 210

```

5. Write a program to put random color blips on the top left one-fourth of the screen, as indicated in the following diagram.



6. The following program puts 20 random numbers (1 or 2) on the screen.

```
10 CLS
20 FOR RN=1 TO 20
30   PRINT RND(2);
40 NEXT RN
50 GOTO 50
```

RND(2)

If you RUN the above program, you will usually (but not always) see about the same number of 1's and 2's on the screen. Go ahead, try it a few times. Count the 1's and 2's. Keep track of the results.

Now try the following program, which also puts 20 random 1's or 2's on the screen.

```
10 CLS
20 FOR RN=1 TO 20
30   PRINT RND(RND(2));
40 NEXT RN
50 GOTO 50
```

RND(RND(2))

RUN the second program several times. Count the 1's and 2's. Keep track of the results.

Describe the difference between RND(2) and RND(RND(2))._____

7. Before you RUN the following program, describe what will happen.

```

100 REM ** MYSTERY PROGRAM
110 CLS 0

200 REM ** RANDOM STUFF
210 OVER = RND(RND(64)) - 1
220 DOWN = RND(RND(32)) - 1
230 KOLOR = RND(8)

300 REM ** TURN ON A BLIP
310 SET(OVER, DOWN, KOLOR)

400 REM ** DO IT AGAIN
410 GOTO 210

```

8. Write a program to put a random graphics character in every screen position from 0 to 510.
9. Add SOUND to the Mandala program in frame 12. Make the sound depend in some way on the values of H and V. You may wish to make the sound depend on distance from the center of the screen. Hmmm, perhaps you would like to make sound depend on the color as well as position on the screen. Remember, you can make both tone number and duration number depend on things. Or, if you wish, you can make the sound completely random.
10. Write a program to put NR random color rectangles on the screen. Here are some ideas about ways to tell the computer what you want.
- The user gets to select how many (NR) rectangles.
 - User can select maximum height (H) and width (W) for rectangles.
 - Computer will keep rectangles within those limits. You can select tall, thin rectangles or short, fat rectangles, or . . .
 - User can select maximum values for positioning top left corner of rectangle. Call these OMAX (Over MAXimum) and DMAX (Down MAXimum).

Of course, user is responsible for choosing the above so that the computer does not try to go off the screen.

11. Rewrite the dice rolling subroutine in frame 21. Use a FOR-NEXT loop to “roll” three dice and compute their SUM (DICE).

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames on the chapter where the topic is discussed.

1. (a) 1, 2, or 3
 (b) 0, 1, or 2
 (c) -1, 0, or 1
 (d) 3, 4, or 5
 (e) 2, 4, or 6
 (f) 1, 3, or 5
 (g) 0, 2, or 4
 (frames 2, 3)
2. (a) 0 to 99
 (b) 11 to 30
 (frames 3, 4)
3. (a) $X = \text{RND}(4) + 9$
 (b) $X = 3 * \text{RND}(3) - 1$
 (frames 2, 3)
4. Here, there, anywhere—on the top line of the screen. To make it wander on the second line of the screen, change line 210 to:

```
210 SP = RND(32) + 31
```

Oh, you want it to wander on the third line? Try:

```
210 SP = RND(32) + 63
```

(frame 9)

5. We did it this way:

```
100 REM ** BLIPS ON THE QUARTER-SCREEN
110 CLS 0

200 REM ** RANDOM PLACE AND COLOR
210 OVER = RND(32) - 1
220 DOWN = RND(16) - 1
230 CLR = RND(8)

300 REM ** TURN ON A BLIP
310 SET(OVER, DOWN CLR)

400 REM ** GO DO IT AGAIN
410 GOTO 210
```

To turn on blips in the bottom right quadrant:
 210 OVER = 64 - RND(32)
 220 DOWN = 32 - RND(16)

(frame 12)

6. RND(2) gives a random number, 1 or 2, with each number having the same probability of occurrence. That is, 1 and 2 are equally likely, just as heads and tails are equally likely when you flip coins.

So, sometimes you get the same number of 1's and 2's, sometimes more 1's, sometimes more 2's.

RND(RND(2)) usually gives more 1's than 2's. On the average, RND(RND(2)) gives about three times as many 1's as 2's. (frame 15)

7. "AHA!" you exclaimed, "The screen will begin filling up with color blips." Then you continued, "Blips are more likely to appear near the upper left corner of the screen. The farther from the upper left corner, the fewer the blips."

Confidently, you entered the program and ran it. As expected, soon the part of the screen near the upper left corner contained lots of blips. The lower right part of the screen remained relatively barren, only now and then receiving a bright splash of color. (frame 12, 15)

You were absolutely correct. Please send further explanations to Bob and George. *Dymax Gazette*, P.O. Box 310, Menlo Park, CA 94025.

8. We did it this way:

```

100 REM ** GRAPHICS CHARACTERS EVERYWHERE
110 CLS 0

200 REM ** PUT AT 0 THRU 510
210 FOR SP=0 TO 510
220   GC = RND(128) + 127
230   PRINT (@SP, CHR$(GC));
240 NEXT SP

300 REM ** DO NOTHING LOOP
310 GOTO 310

```

(frames 19, 20)

- 9, 10, 11. No answers. We leave these entirely to you.

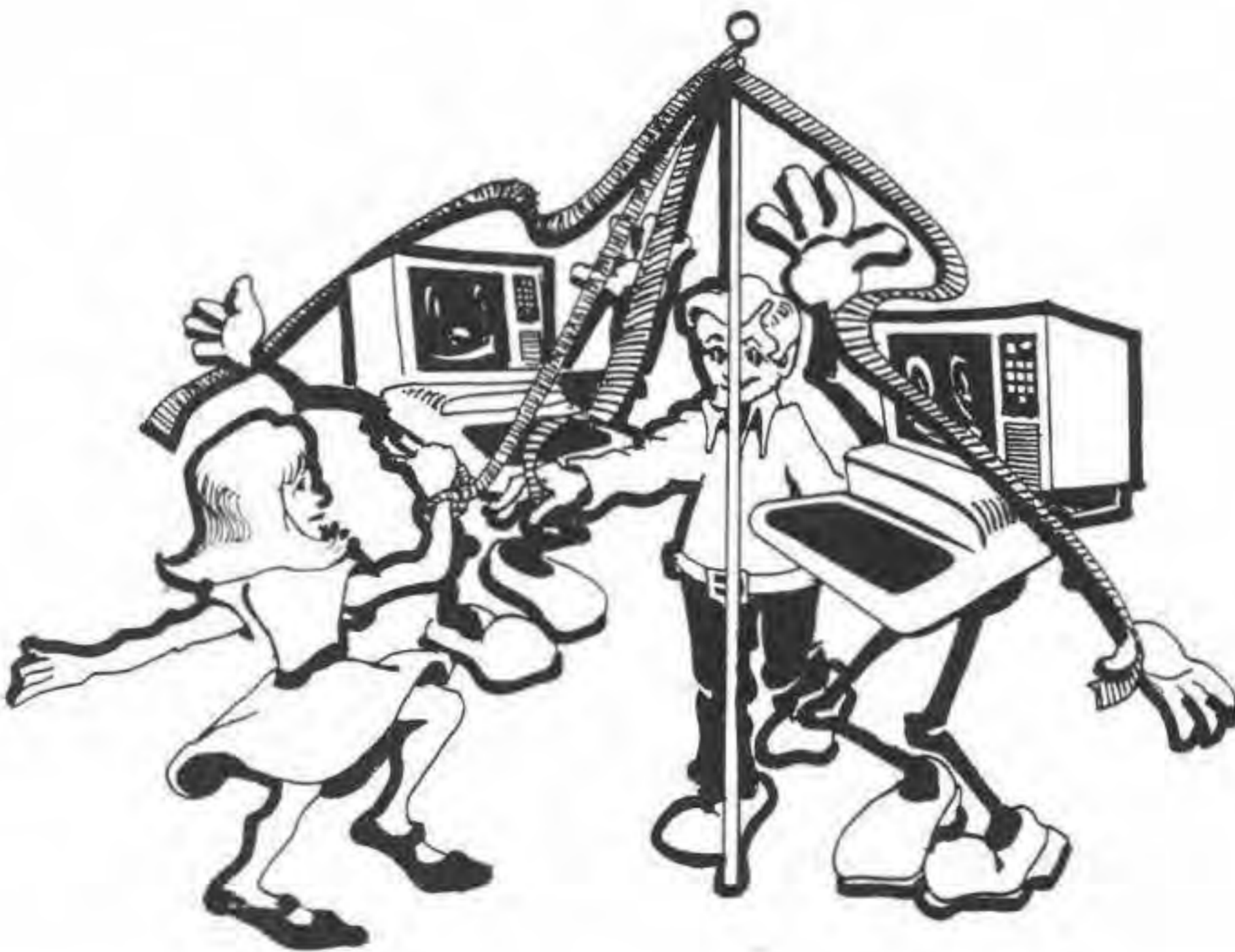
CHAPTER NINE

Playtime Junction

Welcome, wayfarer, to Worlds of IF. In this chapter, you will learn how to teach the Color Computer to make decisions, play games, and quickly respond to your nimble fingers as they race over the keyboard.

When you finish this chapter, you will be able to:

- Use the IF statement to tell the computer to make a decision;
- Use THEN and ELSE clauses with the IF statement;
- Create simple games using color, sound, letters, words, and numbers;
- Use the INKEYS function to get the computer to respond immediately to keys you press on the keyboard.



1. The IF statement tells the computer to make a very simple decision. It tells the computer to do a certain operation *if* a given *condition* is *true*. However, if the condition is *false* (not true), the operation will not be done. Here is an IF statement:

```
410 IF X>0 THEN PRINT "YOUR NUMBER IS POSITIVE"
```

This IF statement tells the computer:

- If the value of X is greater than zero (IF X>0), then *do* print the message, YOUR NUMBER IS POSITIVE.
- If the value of X is not greater than zero, *don't* print the message.

Here is another way to think about it:

- If the value of X is greater than zero, execute the statement following the word THEN.
- If the value of X is *not* greater than zero, *don't* execute the statement following THEN.

Hmmm . . . one more time.

This is the condition.

```
410 IF X>0 THEN PRINT "YOUR NUMBER IS POSITIVE"
```

Do this if the condition is *true*.
Don't do this if the condition is *false*.

- (a) What is the condition in the above IF statement? _____
- (b) Suppose the value of X is 3. Is the condition *true* or *false*? _____
- (c) Suppose X is -7. Is the condition *true* or *false*? _____
- (d) Suppose X is 0. Is the condition *true* or *false*? _____

-
- (a) X>0 or X is greater than zero.
- (b) True. The computer prints the message YOUR NUMBER IS POSITIVE.
- (c) False. The computer does *not* print the message.
- (d) False. The computer does *not* print the message.
-

2. The following program has three IF statements that tell the computer "to do or not to do."

```

100 REM**POSITIVE, NEGATIVE, OR ZERO
110 CLS

200 REM**TELL PERSON WHAT TO DO
210 PRINT "ENTER A NUMBER AND I WILL TELL"
220 PRINT "YOU WHETHER YOUR NUMBER IS"
230 PRINT "POSITIVE, NEGATIVE, OR ZERO."

300 REM**ASK FOR A NUMBER
310 PRINT
320 INPUT "YOUR NUMBER":X

400 REM**TELL PERSON ABOUT THE NUMBER
410 IF X>0 THEN PRINT "YOUR NUMBER IS POSITIVE"
420 IF X<0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
430 IF X=0 THEN PRINT "YOUR NUMBER IS ZERO"

500 REM**LOOP BACK FOR NEW NUMBER
510 GOTO 310

```

A RUN might go like this:

```

ENTER A NUMBER AND I WILL TELL
YOU WHETHER YOUR NUMBER IS
POSITIVE, NEGATIVE, OR ZERO.

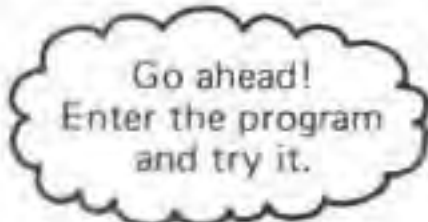
YOUR NUMBER? -7
YOUR NUMBER IS NEGATIVE

YOUR NUMBER? 3
YOUR NUMBER IS POSITIVE

YOUR NUMBER? 0
YOUR NUMBER IS ZERO

YOUR NUMBER? ■ ← Your turn. Carry on!

```



Go ahead!
Enter the program
and try it.

After you RUN the program, read on—we'll ask you some questions about the program.

Try these questions about the program called POSITIVE, NEGATIVE, or ZERO.

- (a) What is the condition in line 410? _____
- (b) What is the condition in line 420? _____
- (c) What is the condition in line 430? _____
- (d) Suppose X is -7. The condition in line 410 is (true or false?) _____, the condition in line 420 is (true or false?) _____, and the condition in line 430 is (true or false?) _____.
- (e) Suppose X is 3. The condition in line 410 is (true or false?) _____, the condition in line 420 is (true or false?) _____, and the condition in line 430 is (true or false?) _____.
- (f) Suppose X is 0. Which condition is true? Line _____, Which conditions are false? Lines _____ and _____.

-
- (a) $X > 0$ or X is greater than zero.
- (b) $X < 0$ or X is less than zero.
- (c) $X = 0$ or X is equal to zero.
- (d) false; true; false. The computer will print the message in line 420, but will not print the messages in lines 410 and 430.
- (e) true; false; false. The computer will print the message in line 410, but will not print the messages in lines 420 and 430.
- (f) 430; 410; 420. The computer will print the message in line 430, but will not print the messages in lines 410 and 420.

3. In general, the IF-THEN statement has the following form:

IF *condition* THEN *statement*

The *statement* could be almost any BASIC statement. The *condition* is usually a comparison between a variable and a number, between two variables, or between two BASIC expressions. Here is a handy table showing both BASIC symbols and math symbols for these comparisons:

BASIC Symbol	Comparison	Math Symbol
=	is equal to	=
<	is less than	<
>	is greater than	>
<=	is less than or equal to	≤
>=	is greater than or equal to	≥
<>	is not equal to	≠

Write each of the following conditions in proper BASIC.

- (a) KOLOR is less than 0. _____
 (b) GUESS is not equal to SECRET. _____
 (c) T is greater than 255. _____
 (d) OVER + H is less than or equal to 63. _____

- (a) KOLOR<0
 (b) GUESS<>SECRET
 (c) T>255
 (d) OVER + H <= 63

4. Game time! Here is our first computer game, a simple number-guessing game.

```

100 REM ** 1ST NUMBER GAME
200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "I WILL THINK OF A NUMBER."
230 PRINT "MY NUMBER IS 1 OR 2."

300 REM ** X IS SECRET NUMBER
310 X = RND(2)
320 PRINT: PRINT "GUESS MY NUMBER."

400 REM ** G IS PLAYER'S GUESS
410 PRINT: INPUT "YOUR GUESS (1 OR 2)"; G

500 REM ** TELL PLAYER IF RIGHT OR WRONG
510 IF G=X THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"
520 IF G<>X THEN PRINT "HA! THAT'S NOT IT."

600 REM ** PLAY AGAIN
610 PRINT "LET'S PLAY AGAIN."
620 GOTO 310

```

Enter the game and play for a short while. Well, it's not the world's most exciting computer game, but it's a beginning. Better stuff to come!

Just to make sure that you understand everything you know about 1ST NUMBER GAME, dash off answers to these questions:

- (a) In line 310, the computer "thinks" of a secret number called X. What are the possible values of X?_____
- (b) What variable holds the player's guess?_____
- (c) Which lines compare the guess with the secret number?_____
- (d) What is the condition in line 510?_____
- (e) Suppose this condition is *true*. What happens?_____
- _____
- (f) Suppose the condition is *false*. What happens?_____
- _____
- (g) What is the condition in line 520?_____
- (h) Suppose this condition is *true*. What happens?_____
- _____
- (i) Suppose the condition is *false*. What happens?_____
- _____

-
- (a) 1 or 2.
 - (b) G (used in lines 410, 510, and 520).
 - (c) Lines 510 and 520.
 - (d) $G=X$. (G is equal to X.)
 - (e) The computer prints THAT'S IT! YOU MUST HAVE ESP!
 - (f) Nothing. The computer moves on to line 520.
 - (g) $G<>X$. (G is not equal to X.)
 - (h) The computer prints HA! THAT'S NOT IT.
 - (i) Nothing. The computer moves on to line 610.
-

5. A condition can also be a relationship between two strings or two string variables, as shown in lines 510 and 520 of the following program. In this game, the computer "flips" a coin. You guess H (Heads) or T (Tails).

```
100 REM ** COIN FLIP GAME
200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "I WILL FLIP A COIN. GUESS"
230 PRINT "H FOR HEADS OR T FOR TAILS."

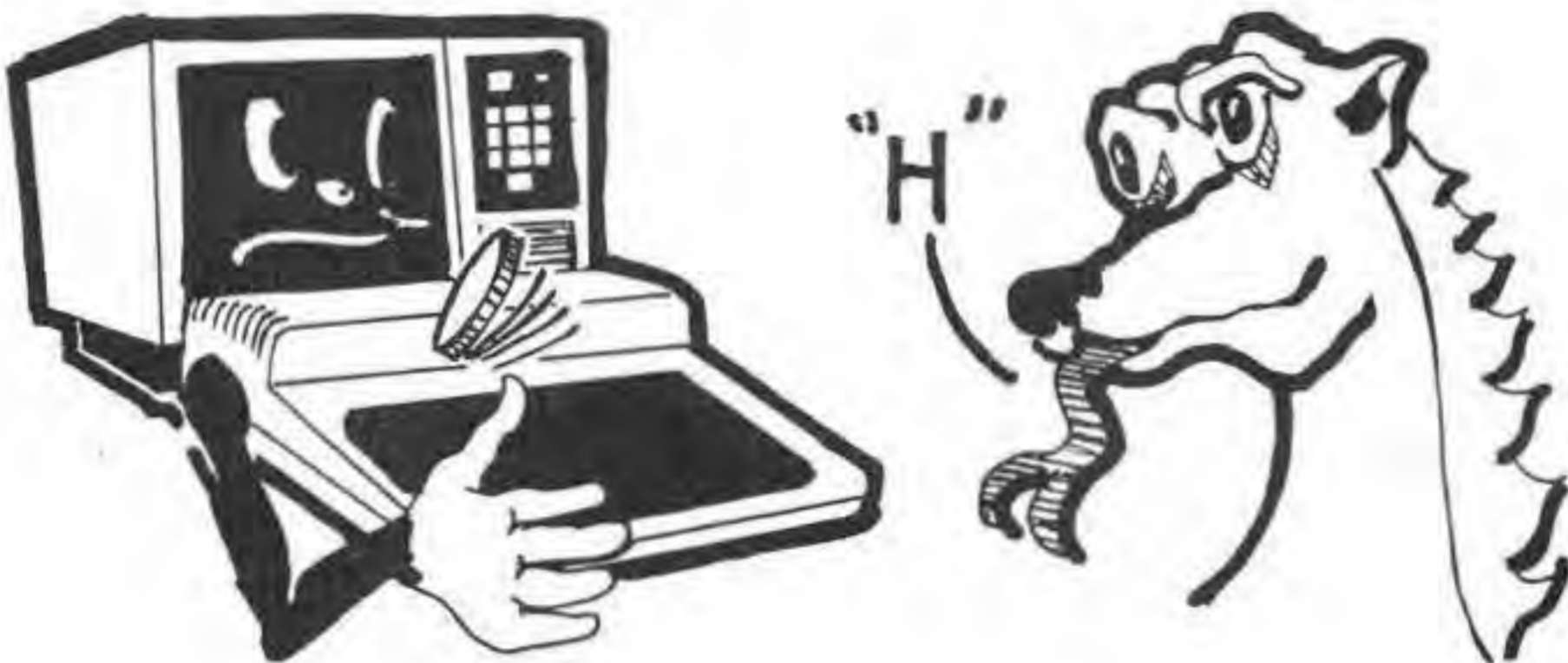
300 REM ** COMPUTER "FLIPS" COIN, C$
310 C = RND(2)
320 IF C=1 THEN C$ = "H"
330 IF C=2 THEN C$ = "T"
340 PRINT: PRINT "OK, I FLIPPED IT."

400 REM ** GET A GUESS, G$
410 PRINT: INPUT "YOUR GUESS (H OR T)": G$

500 REM ** TELL IF RIGHT OR WRONG
510 IF G$=C$ THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"
520 IF G$<>C$ THEN PRINT "HA! THAT'S NOT IT."

600 REM ** PLAY AGAIN
610 PRINT "LET'S PLAY AGAIN."
620 GOTO 310
```

Enter the program and play the game. The computer simulates (imitates) flipping a "fair" coin. It will give heads (H) or tails (T) with about equal probability, just as in flipping a real coin.



Now answer these questions about the COIN FLIP GAME:

- (a) In line 310, what are the possible values of C? _____
- (b) Suppose C is 1. What will be the value of the string variable C\$? _____
- (c) Suppose C is 2. What will be the value of C\$? _____
- (d) What string variable is used to hold the player's guess? _____

Lines 510 and 520 compare the player's guess with the computer's "coin."

- (e) What is the condition in line 510? _____
 - (f) What is the condition in line 520? _____
 - (g) Got it? _____
-

- (a) 1 or 2.
- (b) "H" This happens in line 320 because C=1 is true.
- (c) "T" This happens in line 330 because C=2 is true.
- (d) G\$ (lines 410, 510, and 520).
- (e) G\$ = C\$ or G\$ is equal to C\$.
- (f) G\$ <> C\$ or G\$ is not equal to C\$.
- (g) We hope you said yes. If not, compare this program with the one in frame 4. These are quite similar. Differences include the use of (in this program) string variables to hold the computer's secret flip and the player's guess. In lines 320 and 330, we use both a numeric variable (C) and a string variable (C\$).

6. When you play the COIN FLIP GAME, you will probably win about half the time and lose half the time. Play it 10 or 20 or 100 times and keep track. What? You played it 100 times and won 73 times? Well, maybe you *do* have ESP!

Have some fun with a friend. Make these changes to the COIN FLIP PROGRAM.

```
100 REM ** UNFAIR COIN FLIP GAME  
  
300 REM ** FLIP AN UNFAIR COIN  
310 C = RND(3)  
320 IF C <= 2 THEN C$ = "H"  
330 IF C = 3 THEN C$ = "T"
```

Now the computer will "flip" heads (H) about twice as often as tails (T). Have your friend keep track of wins and losses. Maybe she or he will discover that the computer is flipping an unfair coin.

Here are three more unfair variations:

```
310 C = RND(3)
320 IF C=1 THEN C$ = "H"
330 IF C>1 THEN C$ = "T"
```

```
310 C = RND(4)
320 IF C=1 THEN C$ = "H"
330 IF C>1 THEN C$ = "T"
```

```
310 C = RND(5)
320 IF C<=3 THEN C$ = "H"
330 IF C>3 THEN C$ = "T"
```

7. Next, a game of GUESS MY TONE. The computer computes a random tone number, 1 to 255, then plays the tone. You guess the tone number. The computer then gives you a hint to help you guess again. A run might go like this.

I'LL PLAY A TONE. YOU GUESS
MY TONE NUMBER. 1 TO 255.

MY LOWEST TONE IS 1
MY HIGHEST TONE IS 255

GUESS THIS TONE

YOUR GUESS? 100
TRY LOWER TONE


YOUR GUESS? 20
TRY HIGHER TONE

YOUR GUESS? 73
THAT'S IT. YOU GUESSED MY TONE.

Computer sounds these tones.

Computer sounds mystery tone before each guess.

Only three guesses? How unlikely!



After a short time delay, the game begins again with a new tone.

Here is the GUESS MY TONE Program. Enter it and play.

```
100 REM ** GUESS MY TONE
200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "I'LL PLAY A TONE. YOU GUESS"
230 PRINT "MY TONE NUMBER, 1 TO 255."
240 PRINT
250 PRINT "MY LOWEST TONE IS 1."
260 PRINT "MY HIGHEST TONE IS 255."

300 REM ** COMPUTE AND PLAY RANDOM TONE
310 T = RND(255)
320 PRINT
330 PRINT "GUESS THIS TONE"

400 REM** SOUND TONE & GET GUESS
410 PRINT : SOUND T, 20
420 INPUT "YOUR GUESS": G

500 REM ** IF NOT CORRECT, GIVE HINT.
510 IF G<T THEN PRINT "TRY HIGHER TONE": GOTO 410
520 IF G>T THEN PRINT "TRY LOWER TONE" : GOTO 410

600 REM ** WINNER!
610 PRINT "THAT'S IT! YOU GUESSED MY TONE."
620 SOUND T, 100

700 REM ** GO PLAY AGAIN
710 GOTO 210
```

Lines 510 and 520 show another useful feature of the IF statement.

```
510 IF G<X THEN PRINT "TRY HIGHER TONE": GOTO 410
```

All of this is done if the condition is *true*.
None of this is done if the condition is *false*.

When the condition is *true*, the computer simply moves on to line 520 without doing the PRINT or GOTO statements.

In line 510, the condition is: $G < X$.

- (a) Suppose G is 100 and T is 73. Is the condition true or false? _____
(b) Suppose G is 20 and T is 73. Is the condition true or false? _____
(c) Suppose G is 73 and T is 73. Is the condition true or false? _____
-
-

- (a) False. The computer does *not* do the statements to the right of THEN. It goes on to line 520.
- (b) True. The computer will print TRY A HIGHER TONE and then it will GOTO line 410.
- (c) False. The computer does *not* do the statements to the right of THEN. It goes on to line 520.

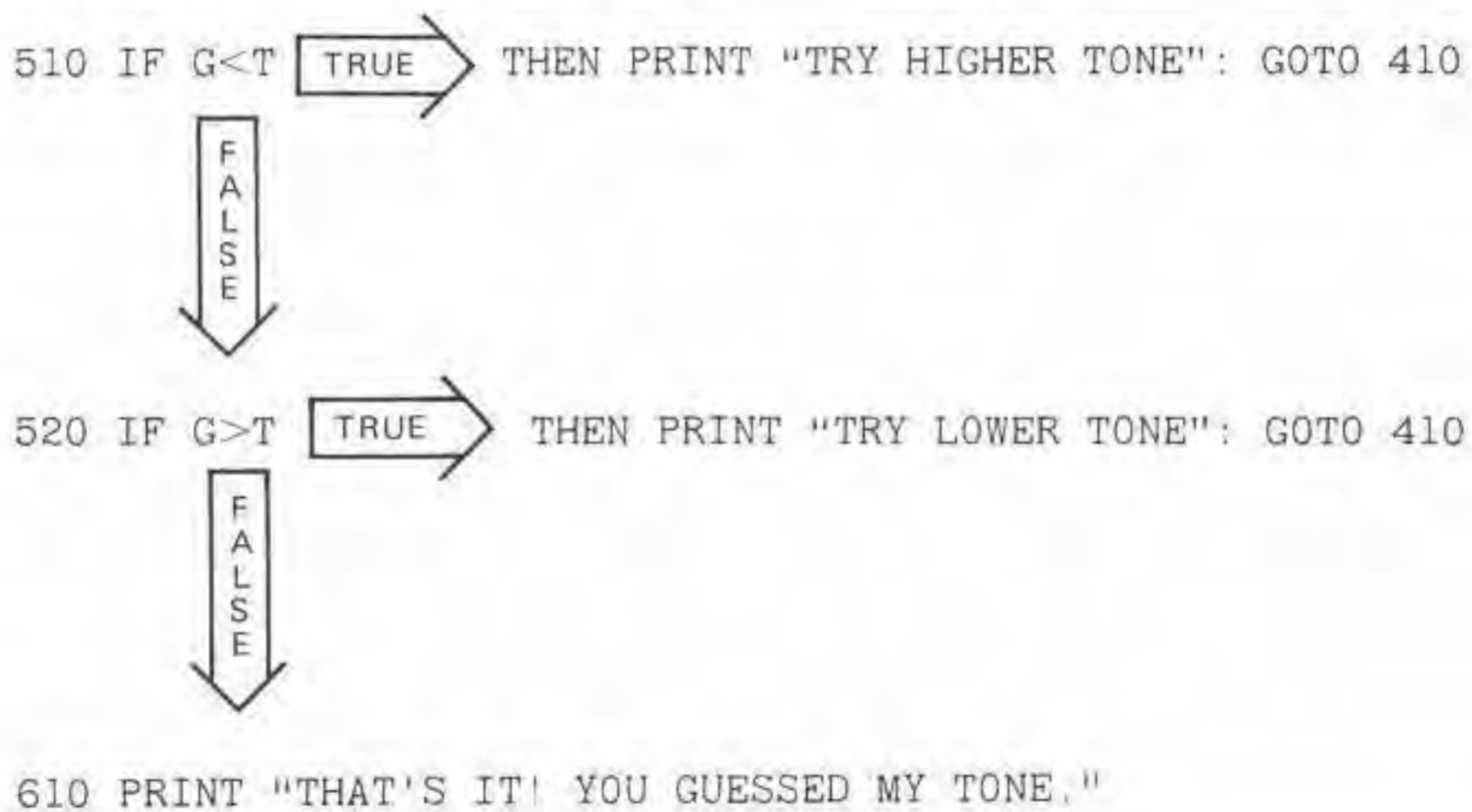
8. Suppose G is 100 and T is 73. Then $G < T$ is false and the computer goes on to line 520.

```
520 IF G>T THEN PRINT "TRY LOWER TONE": GOTO 410
```

What does the computer do in line 520? _____

Since G is 100 and T is 73, the condition in line 520 is *true*. The computer prints TRY A LOWER TONE and goes to line 410.

9. Eventually, the player will guess the tone number. Then, in line 510, $G < T$ is false and, in line 520, $G > T$ is also false. The computer finally arrives at line 610. Here is a diagram of the entire process of arriving at line 610. (Since the computer ignores REMs, so do we.)



What happens? _____

The computer prints THAT'S IT! YOU GUESSED MY TONE and moves on to line 620.

There are three possible paths through the program, beginning at line 410. In frame 7, color them red, blue, and green.

Red path: 410 to 420 to 510 to 410, and so on.

Blue path: 410 to 420 to 510 to 520, to 410, and so on.

Green path: 410 to 420 to 510 to 520 to 610 to 620 to 710 to 410, and so on.

10. Your turn. Try these variations of GUESS MY TONE or write your own variations.

VARIATION 1. Make these changes and additions:

```
110 H$ = "TRY A HIGHER TONE."  
120 L$ = "TRY A LOWER TONE."  
130 W$ = "THAT'S IT! YOU GUESSED MY TONE NUMBER."  
  
510 IF G<T THEN PRINT H$: GOTO 410  
520 IF G>T THEN PRINT L$: GOTO 410  
  
610 PRINT W$
```

Now it is much easier for you to change what the computer says. Simply put your strings in lines 110, 120, and 130. You can make the computer sound friendly, grumpy, aloof, or whatever suits your fancy.

VARIATION 2. You can use this with VARIATION 1 or without it.

```
140 LT = 89  
150 HT = 176  
  
230 PRINT "MY TONE NUMBER," LT "TO" HT ". "  
250 PRINT "MY LOWEST TONE IS" LT  
260 PRINT "MY HIGHEST TONE IS" HT  
  
310 T = RND(HT - LT + 1) + (LT - 1)
```

Boggled by line 310? We will try to disenboggle you so you can proceed unbeduddled.



- (a) Since $LT = 89$ and $HT = 176$, what is $HT - LT + 1$? _____
 (b) What are the possible values of $RND(HT - LT + 1)$? _____
 (c) In line 310, what are the possible values of T ? _____

- (a) $176 - 89 + 1 = 88$.
 (b) 1 to 88, inclusive.
 (c) 89 to 176, inclusive. Huh? How did we get that? Like this:
 Suppose $RND(HT - LT + 1)$ is 1.

$$T = 1 + (LT - 1) = 1 + (89 - 1) = 89$$

Suppose $RND(HT - LT + 1)$ is 88.

$$T = 88 + (LT - 1) = 88 + (89 - 1) = 176$$

Of course! (We hope we heard you exclaim.)

11. Tired of guessing games? Try THE COMPUTER AS MAD ARTIST.

```

100 REM ** THE COMPUTER AS MAD ARTIST
110 CLS 0

200 REM ** HORIZONTAL OR VERTICAL STRIPE
210 HV = RND(2)
220 IF HV = 1 THEN GOSUB 510
230 IF HV = 2 THEN GOSUB 710
240 GOTO 210

500 REM ** SUBROUTINE: HORIZONTAL STRIPE
510 L = RND(32) - 1
520 R = RND(32) + L
530 DOWN = RND(32) - 1
540 KOLOR = RND(8)
550 FOR OVER = L TO R
560   SET(OVER, DOWN, KOLOR)
570 NEXT OVER
580 RETURN
  
```



Hmmm . . . something missing? Of course, we forgot to write the subroutine to paint a vertical stripe. An excellent opportunity for you!

```

700 REM ** SUBROUTINE: VERTICAL STRIPE
  
```

We did it like this:

```
710 T = RND(16) - 1
720 B = RND(16) + T
730 OVER = RND(64) - 1
740 KOLOR = RND(8)
750 FOR DOWN = T TO B
760   SET(OVER, DOWN, KOLOR)
770 NEXT DOWN
780 RETURN
```

Experiment! Modify this program to get different patterns. For example, try

```
RND(RND(64)) - 1
```

12. We hope the artistic interlude in the preceding frame has rested your mind so you won't object to another guessing game. This time, the computer "thinks" of a number. You guess, then listen. The computer plays a tone. High tone means you are far away. Low tone means you are close. The lower the tone, the closer you are to the computer's secret number.

```

100 REM ** LISTEN, AND GUESS MY NUMBER
110 LO = 1
120 HI = 255

200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "I'LL THINK OF A NUMBER"
230 PRINT "FROM" LO "TO" HI
240 PRINT
250 PRINT "GUESS MY NUMBER, THEN LISTEN."
260 PRINT "HIGH TONE MEANS FAR AWAY."
270 PRINT "LOW TONE MEANS YOU ARE CLOSE."

300 REM ** COMPUTER 'THINKS' OF NUMBER
310 N = RND(HI - LO + 1) + (LO - 1) Boggled? See frame 10.

400 REM ** GET A GUESS
410 PRINT
420 INPUT "YOUR GUESS": G

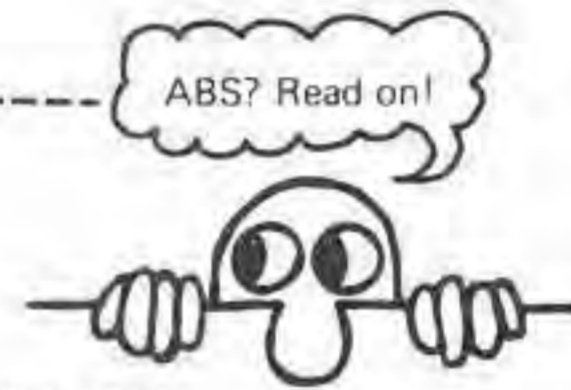
500 REM ** D IS DISTANCE FROM N
510 D = ABS(N-G) ←-----
600 REM ** CHECK FOR A WIN
610 IF D=0 THEN GOTO 810

700 REM ** NO WIN. SOUND HINT.
710 IF D>255 THEN SOUND 255, 30: GOTO 410.
720 SOUND D, 30: GOTO 410

800 REM ** WINNER!
810 PRINT "CONGRATULATIONS! YOU GOT IT..!"

910 REM ** TIME DELAY, THEN PLAY AGAIN
920 Z = 1000
930 FOR K=1 TO Z: NEXT Z
940 GOTO 210

```



In line 510, we used the ABS function.* ABS means "ABSolute value." In this program, we use ABS to compute the "distance" from the guess (G) to the computer's number (N). Here are some examples:

*For more information about ABS, see Chapter 13.

N	G	N-G	ABS(N-G)	
73	30	43	43	
30	73	-43	43	
10	1	99	99	
1	100	-99	99	
37	37	0	0	Winner!

(a) Suppose N is 85 and G is 33. What is ABS(N-G)? _____

(b) Suppose N is 33 and G is 85. What is ABS(N-G)? _____

(a) 52; (b) 52

13. In the game LISTEN, AND GUESS MY NUMBER (frame 12), how would you change the range of numbers to guess to:

(a) 1 to 100?

(b) 1 to 1000?

(c) -100 (Brrr!) to 212?

Easy! Change lines 110 and 120, as follows:

- (a) 110 LO = 1
120 HI = 100
- (b) 110 LO = 1
120 HI = 1000
- (c) 110 LO = -100
120 HI = 212
-

14. The MAD ARTIST MEANDERS. The Mad Artist meanders right, left, up, or down in the universe of the screen.

```

100 REM ** THE MAD ARTIST MEANDERS
110 CLS 0

200 REM ** THE MAD ARTIST APPEARS
210 OVER = 31
220 DOWN = 15
230 KOLOR = RND(8)
240 SET(OVER), DOWN, KOLOR)


300 REM ** WHITHER SHALL SHE/HE MEANDER?
310 WHERENEXT = RND(4)

400 REM ** THE MAD ARTIST MEANDERS
410 IF WHERENEXT = 1 THEN OVER = OVER + 1
420 IF WHERENEXT = 2 THEN OVER = OVER - 1
430 IF WHERENEXT = 3 THEN DOWN = DOWN + 1
440 IF WHERENEXT = 4 THEN DOWN = DOWN - 1

500 REM ** THE MAD ARTIST PAINTS
510 KOLOR = RND(8)
520 SET(OVER), DOWN, KOLOR)

600 REM ** KEEP MEANDERING
610 GOTO 310

```



To save time,
you might use
WH instead of
WHERENEXT.

The Mad Artist first appears near the center of a black screen, then meanders up, down, left, or right—painting as she or he goes. In line 310, WHERENEXT will be 1, 2, 3, or 4.

- (a) If WHERENEXT is 1, which direction does Mad Artist go? _____
 (b) If WHERENEXT is 2, which direction does Mad Artist go? _____
 (c) If WHERENEXT is 3, which direction does Mad Artist go? _____
 (d) If WHERENEXT is 4, which direction does Mad Artist go? _____

 (a) right (line 410); (b) left (line 420); (c) down (line 430); (d) up (line 440)

15. Eventually, Mad Artist will meander off the edge of the screen and you will see this: ?FC ERROR IN 520.

This happens if the value of OVER becomes less than 0 or more than 63. It also happens if DOWN becomes less than 0 or more than 31.

Add lines, beginning at line 450, to prevent the above from happening.


```
100 REM ** THE MAD ARTIST MEANDERS
110 CLS 0

200 REM ** THE MAD ARTIST APPEARS
210 OVER = 31
220 DOWN = 15
230 KOLOR = RND(8)
240 SET(OVER, DOWN, KOLOR)

300 REM** WHITHER SHALL SHE/HE MEANDER?
310 WHERENEXT = RND(4)

400 REM ** THE MAD ARTIST MEANDERS
410 IF WHERENEXT = 1 THEN OVER = OVER + 1
420 IF WHERENEXT = 2 THEN OVER = OVER - 1
430 IF WHERENEXT = 3 THEN DOWN = DOWN + 1
440 IF WHERENEXT = 4 THEN DOWN = DOWN - 1
450 _____
460 _____
470 _____
480 _____

500 REM ** THE MAD ARTIST PAINTS
510 KOLOR = RND(8)
520 SET(OVER, DOWN, KOLOR)

600 REM ** KEEP MEANDERING
610 GOTO 310
```

Here's one way: Yours may be different.

```
450 IF OVER<0 THEN OVER = 0
460 IF OVER>63 THEN OVER = 63
470 IF DOWN<0 THEN DOWN = 0
480 IF DOWN>31 THEN DOWN = 31
```

16. Remember MANDALA, EVER CHANGING? (See Chapter 8, Frame 13.) Here it is again, with something added.

```

100 REM ** MANDALA, EVER CHANGING
110 CLS 0

200 REM ** HORIZONTAL & VERTICAL OFFSET
210 H = RND(RND(32)) - 1
220 V = RND(RND(16)) - 1

300 REM ** RANDOM COLOR
310 KOLOR = RND(8)

400 REM ** TURN ON 4 BLIPS
410 SET(31 - H, 15 - V, KOLOR)
420 SET(31 - H, 16 + V, KOLOR)
430 SET(32 + H, 15 - V, KOLOR)
440 SET(32 + H, 16 + V, KOLOR)

500 REM ** TIME DELAY
510 Z = 10
520 FOR K=1 TO Z: NEXT K

600 REM ** IF S KEY NOT PRESSED, GOTO 210
610 IF INKEY$ <> "S" THEN GOTO 210 ← Something new.

700 REM ** THIS HAPPENS IF 'S' IS PRESSED
710 GOTO 710

```

RUN it. The mandala grows and changes on the screen. If you see one you like, press the **S** key. The mandala on the screen stops changing. Gaze upon it for as long as you wish, then press **BREAK** to stop the program.

RUN the program several times. Each time, press the **S** key when you see a pattern that you like. When you tire of meditating on your selected patterns, press **BREAK**.

17. The INKEY\$ function scans the keyboard. If you press a key, then the value of INKEY\$ is a *one character* string—the key you pressed.

If you press the **A** key, the value of INKEY\$ is "A"

If you press the **Z** key, the value of INKEY\$ is "Z"

If you press the **S** key, the value of INKEY\$ is "S"

Look at line 610 in the mandala program.

```
610 IF INKEY$ < >"S" THEN GOTO 210
```

This tells the Color Computer:

- If the value of INKEY\$ is *not* equal to "S", then GOTO 210. So, if *no* key is pressed, or a key other than the S key, the computer will go to line 210 and continue building a mandala.
- However, if someone presses the S key, then the condition INKEY\$<>"S" is false. The computer does *not* GOTO 210. Instead, it goes on to line 710, which says GOTO 710. So it goes to 710, which says GOTO 710 . . . of course, this goes on and on and on, until you press **BREAK**.

- (a) Rewrite line 610 so that pressing the Z keys stops the computer.

610 _____

- (b) This one is tricky! Rewrite line 610 so that pressing the space bar stops the computer.

610 _____

- (a) 610 IF INKEY\$<>"Z" THEN GOTO 210
 (b) 610 IF INKEY\$<>" " THEN GOTO 210

Put a space here
 between quotation marks.

The following will not work for (b).

```
610 IF INKEY$<>"SPACE" THEN 210
```

The value of INKEY\$ can be one, and only one, character.

18. Experiment. Try this short program, which uses INKEY\$:

```
100 REM ** EXPERIMENT WITH INKEY$
110 CLS

200 REM ** TELL WHAT TO DO
210 PRINT: PRINT "PRESS A KEY"

300 REM ** TIME DELAY
310 Z = 1000
320 FOR K=1 TO Z: NEXT K

400 REM ** SCAN KEYBOARD, PRINT KEY
410 PRINT "YOU PRESSED", INKEY$

500 REM ** DO IT AGAIN
510 GOTO 210
```



Here is what happened when we ran the program:

PRESS A KEY	←	We didn't press a key.
PRESS A KEY A	←	We pressed the A key.
PRESS A KEY	←	We didn't press a key.
PRESS A KEY	←	
PRESS A KEY 7	←	We pressed 7.
PRESS A KEY %	←	Quickly, SHIFT , 5 .

... and so on. Press a key!
Spend some time with this program.

- If you *don't* press a key, the computer prints YOU PRESSED, followed by empty space.
- If you press a key, the computer tells you what key you pressed. Well, as you will find out, some keys are “invisible.”

Try this: Quickly press two or three keys. The computer will see only the last key you pressed, just a tiny instant before it executed line 410. If you have trouble seeing this, increase the time delay in line 310.

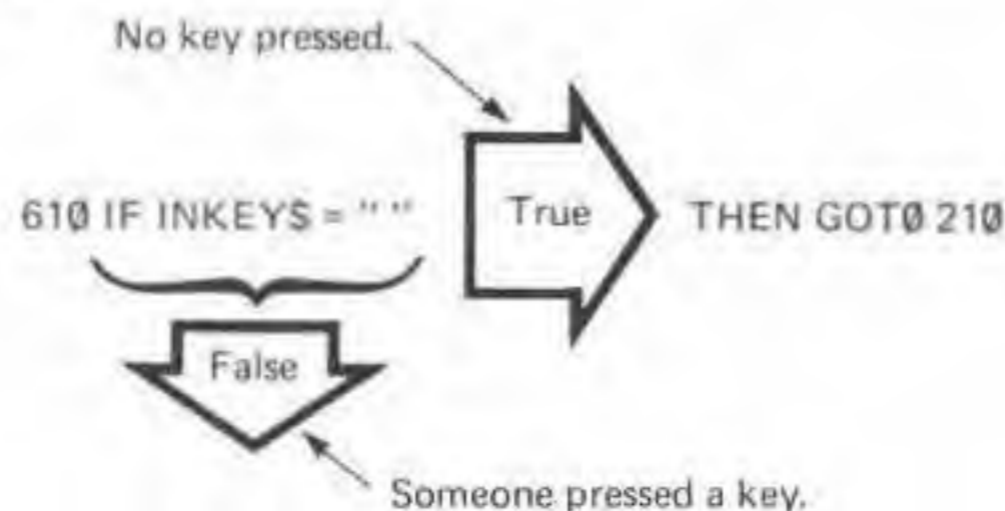
19. Here is a sneaky way to use almost any key to interrupt the computer.

```
600 REM ** IF NO KEY IS PRESSED, GOTO 210
610 IF INKEY$ = "" THEN GOTO 210
```

↑
Nothing in quotes,
not even a space.

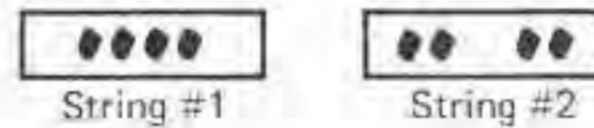
Line 610 says: If the value of INKEY\$ is “nothing,” then go to line 210. This happens if no key has been pressed.

If someone presses a key, the value of INKEY\$ will be “something.” In this case, the condition INKEY\$ = “” is *false* and the computer goes to the line following line 610.



Remember, in line 610 there is *nothing* between quotation marks. Since we use quotation marks to enclose strings, you can think of “” as the *empty string*.

How do these strings differ?



Your answer: _____

String #1 is the empty string with *nothing* between quotation marks. String #2 is a string with *something* between quotation marks. This string consists of one space.

20. In MANDALA, EVER CHANGING (frame 20), replace blocks 600 and 700, as follows:

```
600 REM ** IF NO KEY PRESSED, GOTO 210
610 IF INKEY$ = "" THEN GOTO 210

700 REM ** WAIT FOR KEY, THEN GOTO 210
710 IF INKEY$ = "" THEN GOTO 710
720 GOTO 210
```

By now, you should know . . . EXPERIMENT, TRY IT! RUN the program. Press almost any key to stop the action. Then, press almost any key and it continues from where you stopped it.

- RUN the program.
- Press any key to stop.
- Press any key to continue.

Exceptions are the **BREAK** key, which stops the program for good, and the **SHIFT** key. By itself, the **SHIFT** key does nothing. However, if you hold the **SHIFT** key down and press almost any other key, the pattern will stop or, if stopped, continue.

Line 710 tells the computer to keep scanning the keyboard until someone presses a key. If someone presses a key, the computer goes on to line 720.

What happens next? _____

The computer goes to line 210 and continues putting new blips of color on the screen, until someone presses a key.

21. Here is some shorthand to save wear and tear on your fingers:

Instead of: THEN GOTO 210
 You can write: THEN 210

For example, line 610 can be written:

```
610 IF INKEY$ = "" THEN 210
```

You rewrite line 710.

```
710 _____
```

```
710 IF INKEY$ = "" THEN 710
```

22. More shorthand, thanks to an especially useful statement called IF . . . THEN . . . ELSE . . .

Instead of:

```
710 IF INKEY$ = "" THEN GOTO 710
720 GOTO 210
```

You can write:

```
710 IF INKEY$ = "" THEN GOTO 710 ELSE GOTO 210
    ↑           ↑           ↑
    IF        THEN        ELSE
```

This tells the computer:

- If no key has been pressed, then go to line 710.
- Else (if a key has been pressed), go to line 210.

One more time? OK

```
710 IF [ INKEY$ = "" ] [ THEN GOTO 710 ] [ ELSE GOTO 210 ]
           ↑           ↑
           Do this if the   Do this if the
           condition is TRUE. condition is FALSE.
```

Instead of THEN GOTO 710, you can write THEN 710.

Instead of ELSE GOTO 210, you can write ELSE 210.

Rewrite line 710 using the above shorthand plus the shorthand you learned in frame 21.

710 _____

```
710 IF INKEY$ = "" THEN 710 ELSE 210
```

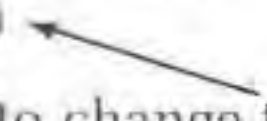
23. Game time again. Try our REACTION TIME GAME.

```
100 REM ** REACTION TIME PROGRAM
200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "HOW FAST ARE YOU? WHEN I START"
230 PRINT "COUNTING, PRESS THE SPACE BAR"
240 PRINT "AS FAST AS YOU CAN TO STOP ME."
250 PRINT
260 PRINT "PRESS ANY KEY AND I'LL BEGIN."
270 IF INKEY = "" THEN 270

300 REM ** CLEAR SCREEN, RANDOM DELAY
310 CLS
320 Z = RND(2000)
330 FOR K=1 TO Z: NEXT K

400 REM ** START COUNTING, SPACE STOPS IT.
410 X = 1
420 PRINT @240, X
430 IF INKEY$<> " " THEN X = X + 1: GOTO 420

500 REM ** TELL HOW TO PLAY AGAIN
510 PRINT @0, "TO PLAY AGAIN, PRESS ANY KEY."
520 IF INKEY$ = "" THEN 520 ELSE 210
```

You might want to change this to 310. 

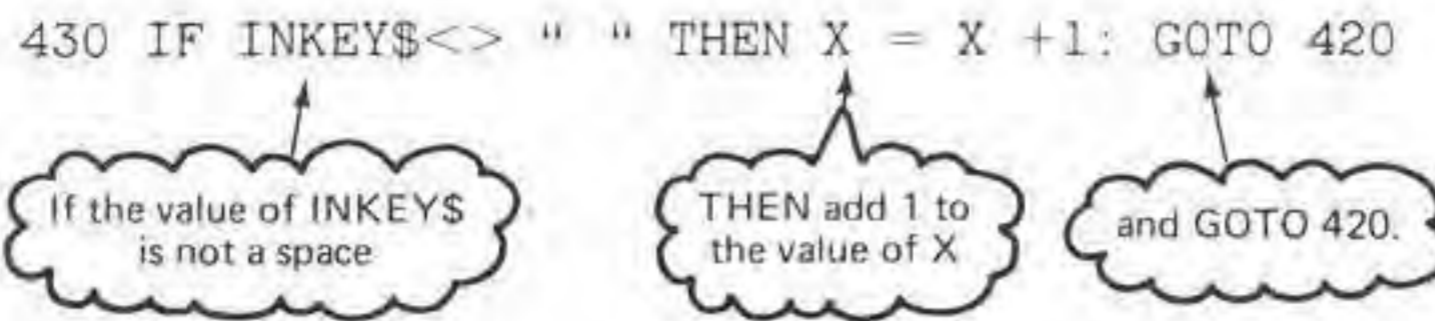
Play several times. An average of 10 is fairly fast. Congratulations! If your average is more than 20, well . . . maybe you are thinking about something else.

We played the game several times and discovered a way to cheat. We can stop the computer with a count of 1 everytime! We can do this, not because we are that fast, but because there is a flaw in the program.

Beat the computer! Figure out how to stop the computer at 1 everytime, just by pressing the space bar.

IMPORTANT NOTICE! This computer error is not the fault of the computer. Rather, as are almost all computer errors, it is the fault of the programmer!

Are you a tiny bit boggled at line 430? It goes like this:



So, if you press the space bar, the condition `INKEY$ <> " "` is false and the computer goes on to line 510.

- (a) How would you change line 430 so pressing the S key stops the counting?

430 _____

- (b) How would you change line 430 so that pressing almost any key stops the counting?

430 _____

(a) 430 IF INKEY\$ <> "S" THEN X = X + 1: GOTO 420

(b) 430 IF INKEY\$ = "" THEN X = X + 1: GOTO 420

24. Have you figured out how to beat the REACTION TIME program? Here's how. When the computer prints PRESS THE SPACE BAR AND I'LL BEGIN, you press the space bar *twice*. Or, press any key, then press the space bar immediately, *before* the computer starts counting near the middle of the screen.

Fix it like this—add the following line to the program.

340 X\$ = INKEY\$

This will clear out the value of INKEY\$ just before the computer begins counting in lines 410 through 430. Line 340 will be done so fast that it is unlikely that you can press the space bar while the computer is going from line 340 to line 410.

Now, get warmed up for the Self-Test by scribbling on the screen.

```
100 REM ** SCRIBBLE ON THE SCREEN
110 CLS 0
120 OVER = 31
130 DOWN = 15

200 REM ** TURN ON A BLIP
210 KOLOR = RND(8)
220 SET(OVER, DOWN, KOLOR)

300 REM ** WAIT FOR SOMEONE TO PRESS A KEY
310 PRINT @480, "PRESS R, L, D, OR U":
320 K$ = INKEY$: IF K$="" THEN 310

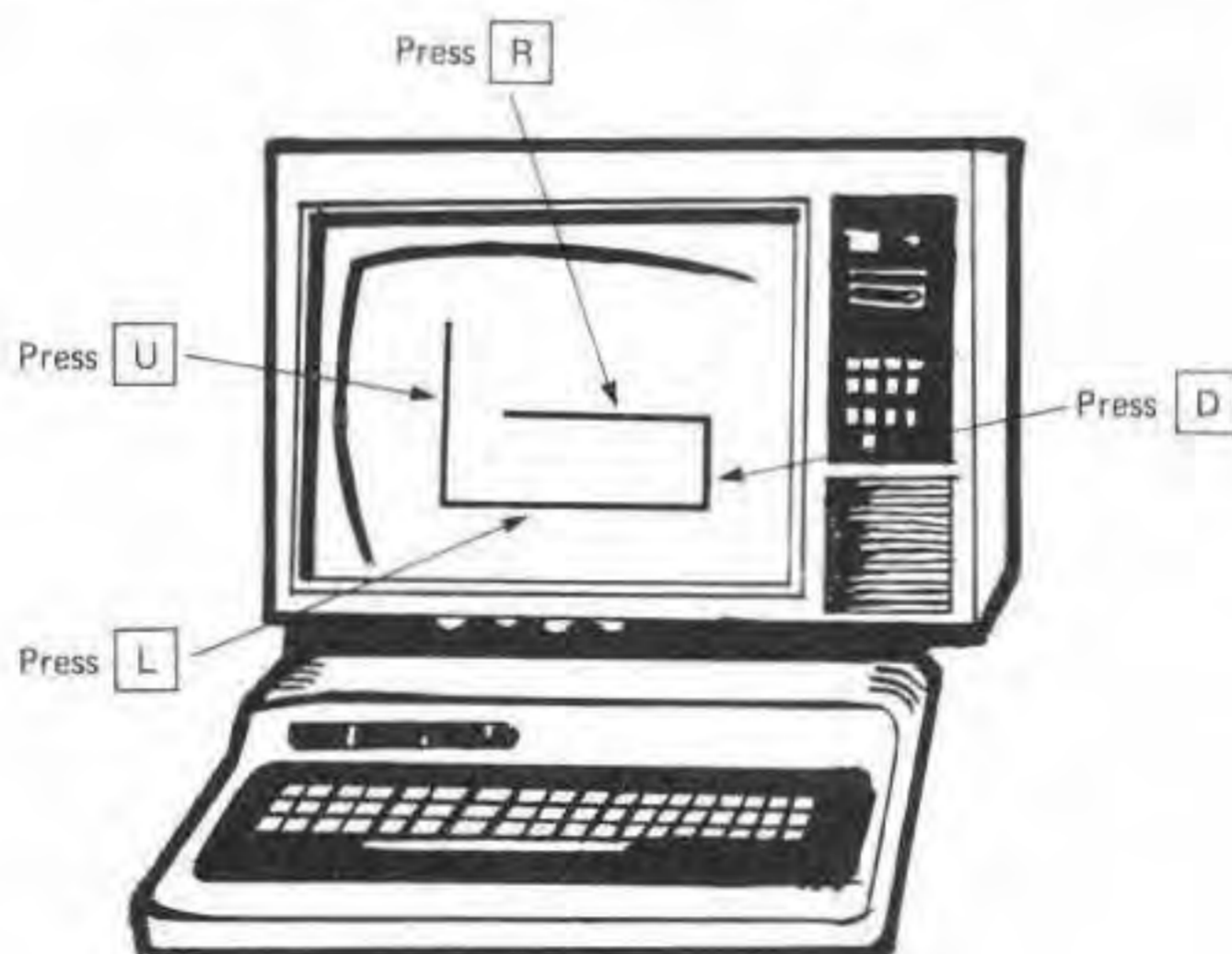
400 REM ** IF KEY WAS R OR L, CHANGE OVER
410 IF K$="R" THEN OVER=OVER+1
420 IF K$="L" THEN OVER=OVER-1

500 REM ** IF KEY WAS D OR U, CHANGE DOWN
510 IF K$="D" THEN DOWN=DOWN+1
520 IF K$="U" THEN DOWN=DOWN-1

600 REM ** GO SCRIBBLE
610 GOTO 210
```

When you RUN this program, a single tiny light will come on near the center of the screen. Press the 'R' key a few times. The computer moves the light to the right, and paints a line along the way.

Then press 'D' a few times. The line prints downward. OK, try 'L' for left and 'U' for up. Got it? Good . . . scribble away.



Beware! If you go too far right or too far left, or too far up or too far down, you will get an ?FC ERROR IN LINE 220. We will fix that someday, sometime. Or, better yet, you can fix it!

Self-Test

IF your mind is fuzzy from cogitating on this chapter, THEN take a break—sing, dance, play conga drums, listen to music, watch Sesame Street—ELSE plunge on into this Self-Test.

1. In each IF statement, underline the condition.
 - (a) If OVER < 0 THEN OVER=OVER+1
 - (b) IF T > 255 THEN 310
 - (c) IF N < =0 THEN PRINT "MUST BE POSITIVE": GOTO 220
 - (d) IF DOWN+1 > 31 THEN 430
 - (e) IF INKEY\$= "" THEN 710 ELSE 210

In doing the next two, guess!

 - (f) IF X < 10 OR X > 99 THEN PRINT "MUST BE 2 DIGITS":
GOTO 310
 - (g) IF X > = 10 AND X < = 99 THEN 410
2. Complete each sentence with *true* or *false*.
 - (a) If X is 73 and G is 50, then G<X is _____.
 - (b) If X is 73 and G is 80, then G<X is _____.
 - (c) If X is 73 and G is 80, then G=X is _____.
 - (d) If X is 73 and G is 80, then G>X is _____.
 - (e) If you have pressed the space bar, then INKEY\$<>" " is _____.
 - (f) If you have pressed the space bar, then INKEY\$="" is _____.
 - (g) If G\$ is "H" and C\$ is "T", then G\$<>C\$ is _____.
3. For each verbal statement, write a corresponding BASIC IF statement.
 - (a) If T is greater than 255, print TONE NUMBER TOO BIG and go to line 310.

 - (b) If D is equal to 1, print seven stars (*).

 - (c) If K\$ is not equal to "S", then go to line 410.

 - (d) If W\$ is equal to "ZZZ" then go to line 720; otherwise go to line 530.

4. Here is a guessing game called STARS. The program is not complete—blocks 200, 800 and 900 remain to be done.

```

100 REM ** STARS - A GUESSING GAME
200 REM ** TELL HOW TO PLAY ←———— Yours!
300 REM ** COMPUTER 'THINKS' OF NUMBER, X
310 X = RND(100)
400 REM ** GET GUESS, G
410 PRINT: INPUT "YOUR GUESS": G
500 REM ** D IS DISTANCE FROM X
510 D = ABS(X-G)
600 REM ** CHECK FOR A WIN
610 IF D=0 THEN 810
700 REM ** NO WIN PRINT HINT.
710 IF D>= 64 THEN PRINT "**": GOTO 410
720 IF D>= 32 THEN PRINT "***": GOTO 410
730 IF D>= 16 THEN PRINT "****": GOTO 410
740 IF D>= 8 THEN PRINT "*****": GOTO 410
750 IF D>= 4 THEN PRINT "*****": GOTO 410
760 IF D>= 2 THEN PRINT "*****": GOTO 410
770 PRINT "*****": GOTO 410
800 REM ** WINNER! ←———— Yours to do!
900 REM ** PLAY AGAIN? ←————

```

You write blocks 200, 800, and 900.

5. Modify the program for STARS – A GUESSING GAME (question 4) by changing block 700 as follows:
- If D is more than 31, print 32 stars across the screen and GOTO 410.
 - If D is less than or equal to 31, print D stars across the screen and GOTO 410.

Of course, you will also have to change block 200 which tells how to play.


6. Complete the following program to play random music selected from the seven tone numbers 89, 108, 125, 133, 147, 159, and 170. These are the numbers for Middle C, D, E, F, G, A, and B.

```

100 REM ** RANDOM MUSIC
110 CLS

200 REM ** SELECT RANDOM TONE NUMBER
210 R = RND(7)
220 IF R=1 THEN T = 89
230 IF R=2 THEN T = 108
240 IF R=3 THEN T = 125
250 IF R=4 THEN T = 133

```



Please complete
block 200.

```

400 REM ** SELECT DURATION
410 D = 1

500 REM ** PLAY A TONE
510 SOUND T, D

600 REM ** DO IT AGAIN

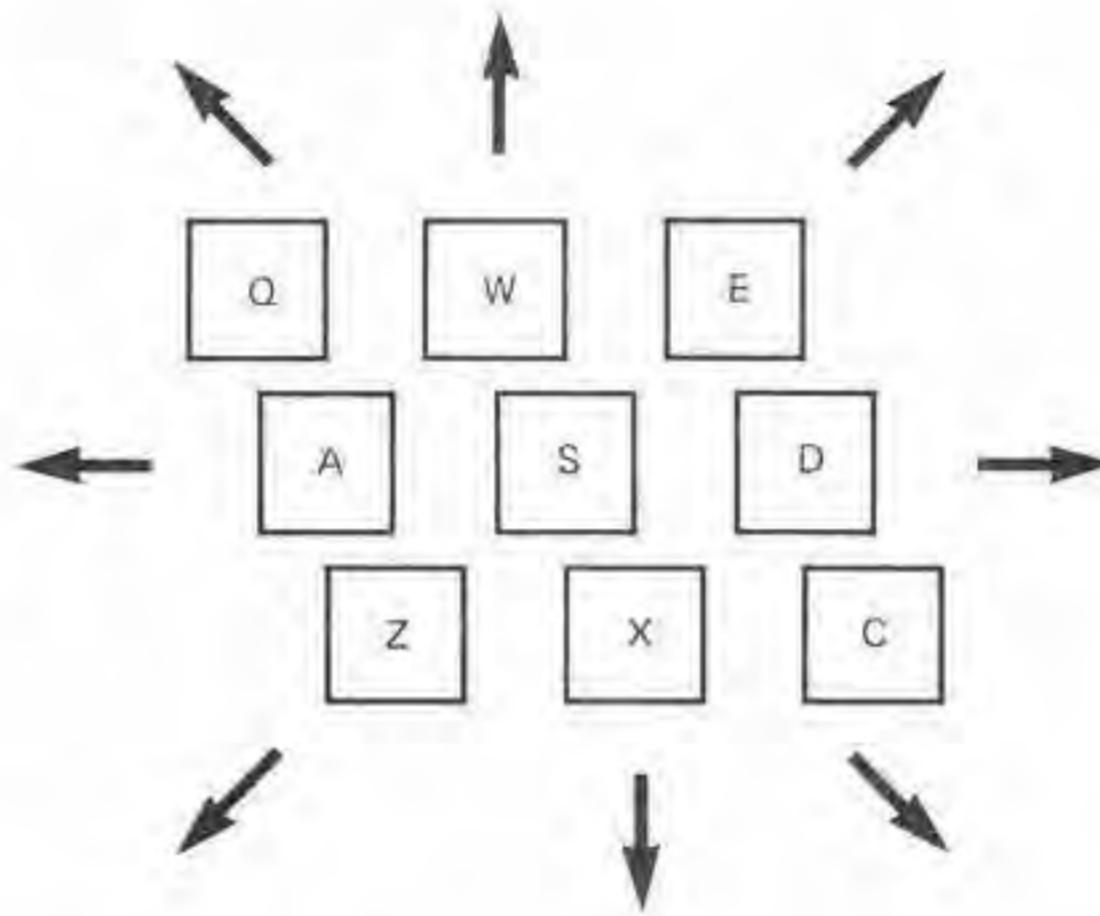
```

7. Suppose you especially like the notes C (T=89) and F (T=133). Rewrite block 200 so that the computer is more likely to play a C (T=89) or an F (T=133) than play a D (T=108), E (T=125), G (T=147), A (T=159), or B (T=170). Hint: Select at random from C, C, D, E, F, F, G, A, B.
8. In question 6, rewrite block 400 so the duration (D) is random.
- Make D equal to 1, 2, 3, or 4, at random.
 - Make D equal to 1, 2, or 4, at random. Musicians: you can think of this as quarter tone, half tone, full tone.
9. Modify the program for STARS – A GUESSING GAME (questions 4 and 5) by rewriting block 700, as follows:
- If D is more than 31, print 32 graphics characters across the screen and GOTO 410.
 - If D is less than or equal to 31, print D graphics characters across the screen and GOTO 410.

You pick the graphics character. You might want a solid character in some color. Then the length of the line in that color is related to the distance of the guess from the computer's secret number.

Also rewrite block 200, which tells how to play.

10. In SCRIBBLE ON THE SCREEN (frame 24) :
- Add statements to prevent going off-screen.
 - Add statements to allow the user to change color (instead of random color in line 210). Color selection by pressing keys 0 through 8. Aha! Key 0 (black) can be used to erase previous stuff. With this change, the computer pays attention to the keys: U, D, L, R, 0, 1, 2, 3, 4, 5, 6, 7, 8
 - Hmmm... wouldn't it be nice to move to a different place on the screen without changing anything on the screen? Of course! Do it. You could use the 9 key for this.
 - How about moving diagonally as well as Up, Down, Left, or Right? Do it. You might want to use a "keypad," as follows:



Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

- `IF OVER < 0 THEN OVER=OVER+1`
 - `IF T > 255 THEN 310`
 - `IF N <=0 THEN PRINT "MUST BE POSITIVE": GOTO 220`
 - `IF DOWN+1 > 31 THEN 430`
 - `IF INKEY$="" THEN 710 ELSE 210`
 - `IF X < 10 OR X > 99 THEN PRINT "MUST BE 2 DIGITS":
GOTO 310`

This condition is true when, for example, X is 9 or when X is 100. These are not 2 digit numbers.

- `IF X >= 10 AND X <= 99 THEN 410`

This is true when X is 10, or 11, or 12, or any number up to 99. For example, suppose X is 37. Then `X >= 10` is true AND `X <= 99` is true. So, the entire condition is true.

(frames 1, 2, 3)

2. (a) true
 (b) false
 (c) false
 (d) true
 (e) false
 (f) false
 (g) true
 (frames 1, 2, 16, 17, 19)
3. (a) IF T>255 THEN PRINT "TONE NUMBER TOO BIG": GOTO 310
 (b) IF D=1 THEN PRINT "*****"
 (c) IF K\$ <> "S" THEN 410 (or THEN GOTO 410)
 (d) IF W\$ = "ZZZ" THEN 720 ELSE 530
 (frames 1-3, 5)
4. We did it this way:

```

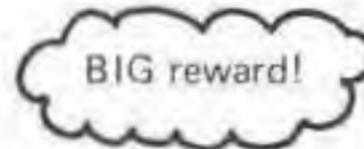
200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "WELCOME TO MY GALAXY. I'LL"
230 PRINT "THINK OF A NUMBER, 1 TO 100."
240 PRINT "YOU GUESS MY NUMBER. IF YOU"
250 PRINT "MISS, I'LL PRINT SOME STARS."
260 PRINT "THE CLOSER YOU ARE, THE MORE"
270 PRINT "STARS YOU WILL SEE."
280 PRINT "IF YOU SEE 7 STARS (*****)."
290 PRINT "YOU ARE VERY, VERY CLOSE!"

800 REM ** WINNER!
810 CLS
820 FOR K=1 TO 100
830   PRINT (@RND(510), "*");
840 NEXT K
850 PRINT @480, "YOU GOT IT, MY NUMBER WAS" X

900 REM ** PLAY AGAIN?
910 PRINT "TO PLAY AGAIN, PRESS ANY KEY."
920 IF INKEY $="" THEN 920 ELSE 210

```

(frame 7)



5. In block 200 lines 210 through 250 are the same as in our answer to question 4.

```

200 REM ** TELL HOW TO PLAY
.
.
.
260 PRINT "THE CLOSER YOU ARE, THE FEWER"
270 PRINT "STARS YOU WILL SEE."
280 PRINT "IF YOU SEE ONE STAR (*), YOU"
290 PRINT "ARE VERY, VERY CLOSE!"

700 REM ** NO WIN. PRINT HINT.
710 IF D 31 THEN N=32 ELSE N=D
720 FOR STAR=1 TO N
730   PRINT "*";
740 NEXT STAR
750 GOTO 410                                     (no particular frame)

```

6. This one was easy!

```

260 IF R=5 THEN T=147
270 IF R=6 THEN T=159
280 IF R=7 THEN T=170                             (no particular frame)

```

7. Here are not one, not two, but three ways to do it.

```

1. 210 R = RND(9)
    220 IF R=1 THEN T= 89 } Two ways to get C.
    230 IF R=2 THEN T= 89 }
    240 IF R=3 THEN T=108
    250 IF R=4 THEN T=125
    260 IF R=5 THEN T=133 } Two ways to get F.
    270 IF R=6 THEN T=133 }
    280 IF R=7 THEN T=147
    290 IF R=8 THEN T=159
    300 IF R=9 THEN T=170

2. 210 R = RND(9)
    220 IF R< =2 THEN T= 89 } Two ways to get C.
    230 IF R> =8 THEN T=133 } Two ways to get F.
    240 IF R=3 THEN T=108
    250 IF R=4 THEN T=125
    260 IF R=5 THEN T=147
    270 IF R=6 THEN T=159
    280 IF R=7 THEN T=170

```


3. 210 R = RND(9)
220 IF R=1 OR R=2 THEN T= 89 Two ways to get C.
230 IF R=3 THEN T=108
240 IF R=4 THEN T=125
250 IF R=5 OR R=6 THEN T=133 Two ways to get F.
260 IF R=7 THEN T=147
270 IF R=8 THEN T=159
280 IF R=9 THEN T=170

(no particular frame)

8. (a) Change line 410 to: 410 R = RND(4).
(b) Change line 410 to one of the following.

```
410 R = RND(4): IF R=3 THEN 410
```

or

```
410 R = RND(3): IF R=3 THEN R=4
```

Either of the above will give 1, 2, or 4 with equal probability.

- 9, 10. No answers. In every chapter, we leave one or more questions without answers. From time to time, solutions will appear in the "Computing Problems" section of *The Dymax Gazette*.
-

CHAPTER TEN

String Functions

You have already used strings and string variables in simple ways. Now you will learn how to use several *string functions* to do interesting and fun things with strings.

When you finish this chapter, you will be able to:

- Reserve memory space for strings;
- Compare two strings;
- Recognize and use ASCII codes for characters;
- Join two strings;
- Convert strings to numbers;
- Find and use *substrings* (strings within strings);
- Use the string functions: ASC, CHR\$, LEFT\$, LEN, MID\$, RIGHT\$, and VAL.

1. A string is any bunch of keyboard characters, typed one after another.

- A string can be a name: KARL
- A string can be a telephone number: 415-323-6117
- A string can be a message: TAKE A DRAGON TO LUNCH
- A string can be gibberish: AB#J%FD+Z

A string can be almost anything you can type on the keyboard.

It can also include graphics characters, as described in Chapter 7.

When you turn on the Color Computer, it automatically reserves 200 characters of memory space for strings. Thus, you may store one string with up to 200 characters or several shorter strings whose total number of characters does not exceed 200.

Occasionally, you may need more room for strings. Use the CLEAR statement to reserve string space.

- CLEAR 300 reserves 300 characters of memory space for strings.
- CLEAR 500 reserves 500 characters of memory space for strings.

- (a) Write a CLEAR statement to reserve 450 characters of memory space for strings: _____
- (b) Write a CLEAR statement to reserve 1000 characters of memory space for strings: _____
- (c) Write a CLEAR statement to reserve 25 characters of memory space for strings: _____
-

(a) CLEAR 450; (b) CLEAR 1000; (c) CLEAR 25 You might do (c) if your program needs lots of space for program statements and numbers, but not much for strings.

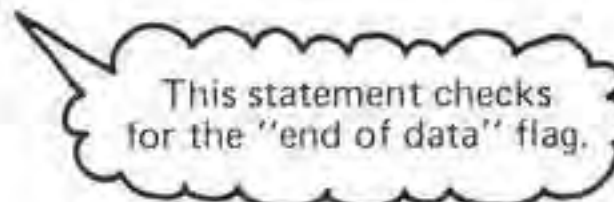
2. Just to make things easy, we'll begin with a program that has many familiar features plus a couple of new ideas. This program rolls *Tunnels and Trolls* characters—as many as you want.

```
100 REM** CREATE A T&T CHARACTER
110 CLS

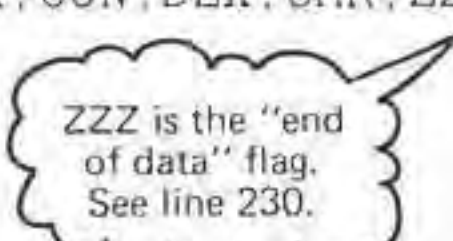
200 REM**ROLL THE CHARACTERISTICS
210 RESTORE
220 READ CH$
230 IF CH$ = "ZZZ" THEN 310
240 D1 = RND(6)
250 D2 = RND(6)
260 D3 = RND(6)
270 DICE = D1 + D2 + D3
280 PRINT CH$, DICE
290 GOTO 220

300 REM** PRESS 'SPACE' TO DO AGAIN
310 PRINT
320 PRINT "PRESS THE SPACE BAR TO GET"
330 PRINT "ANOTHER CHARACTER."
340 K$ = INKEY$ : IF K$ = "" THEN 340
350 IF K$ = " " THEN 110 ELSE 340

900 REM** CHARACTERISTICS
910 DATA STR,IQ,LK,CON,DEX,CHR,ZZZ
```



This statement checks for the "end of data" flag.



ZZZ is the "end of data" flag. See line 230.

What new things do you see in this program? _____

You might have mentioned any or all of the following:

- RESTORE in line 210
- CH\$ = "ZZZ" in line 230
- "end of data" flag in lines 230 and 910

If you came in late, and don't know about rolling characters, please relax—no mugging is intended. See Chapter 8, beginning with frame 21.

3. RUN the program. It might go like this:

```

STR      6
IQ       13
LK       14
CON      9
DEX      14
CHR      16

PRESS THE SPACE BAR TO GET
ANOTHER CHARACTER.
```



Want another character? Press the space bar.

```

STR      15
IQ       9
LK       11
CON      13
DEX      14
CHR      7

PRESS THE SPACE BAR TO GET
ANOTHER CHARACTER.
```

This character and the hobbit complement each other very well. Hmmm . . . they would be an incredible pair of thieves!

4. Now, on to those new statements in the program. The RESTORE statement in line 210 tells the computer to begin again at the first item of data, even if it has previously read all the data. Try this: delete line 210 (type 210 and press **ENTER**), then RUN the program. The computer will roll the first character.

Press the space bar. Oops! This is what you see.

```
?OD ERROR IN 220
OK
■
```

This happened because the computer has already READ all the DATA in rolling the first character. So, put RESTORE back into the program to tell the computer to begin at the first item in the first DATA statement (in case there are two or more DATA statements).

EXPERIMENT! Try both of these short programs:

```
10 CLS
20 READ A$
30 RESTORE
40 GOTO 20
90 DATA REINCARNATED!
```

RESTORED to Life!

```
10 CLS
20 READ A$
30 GOTO 20
90 DATA REST IN PEACE
```

Embalmed Forever.

Is the program below like RESTORED to Life or like Embalmed Forever?

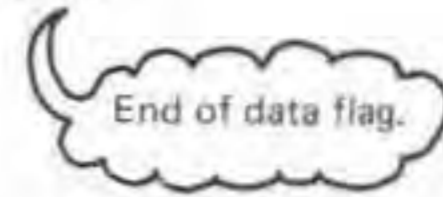
```
10 CLS
20 RESTORE
30 READ A$
40 GOTO 30
90 DATA WHAT HAPPENED?
```

It's like _____

Embalmed Forever. But if you change line 40 to GOTO 20, it will be like RESTORED to Life!

5. On to line 230. But first, look at the DATA.

```
910 DATA STR, IQ, LK, CON, DEX, CHR, ZZZ
```



The first six items (STR through CHR) are the six basic characteristics of a T&T adventurer. The last item (ZZZ) is a “flag,” which signals that all of the interesting data have been read. This flag is sensed by line 230.

```
230 IF CH$ = "ZZZ" THEN 310
```

$\underbrace{\hspace{10em}}$
 IF { the value of
 CH\$ is the
 flag, ZZZ } THEN go to line 310

For STR, IQ, LK, CON, DEX, and CHR, the condition is false. The computer rolls and prints the information for the characteristics. Then the computer reads ZZZ. Now the condition is true—the computer goes to line 310.

In the game of *Runequest*, the characteristics are STR, INT, POW, CON, DEX, CHA, and SIZ. Modify the program so the computer rolls a *Runequest* character.

 Easy! Change only two lines.

```
100 REM** CREATE A RONEQUEST CHARACTER
910 DATA STR, INT, POW, CON, DEX, CHA, SIZ, ZZZ
```

6. Next, a game of GUESS MY WORD. The computer chooses a three-letter word from a list of words in DATA statements. You guess the word. After an incorrect guess, the computer tells you to guess a “lower” word or a “higher” word. This program is similar in structure to GUESS MY TONE in Chapter 9, frame 11.

Here is the first part of the program and the list of words.

```

100 REM ** GUESS MY WORD

300 REM ** TELL HOW TO PLAY
310 CLS
320 PRINT "I'LL THINK OF A 3-LETTER WORD."
330 PRINT "MY WORD IS BETWEEN AAA AND ZZZ."
340 PRINT
350 PRINT "MY LOWEST 'WORD' IS AAA."
360 PRINT "MY HIGHEST 'WORD' IS ZZZ."

400 REM ** PICK RANDOM WORD FROM DATA
410 RESTORE
420 READ NW
430 RW = RND(NW)
440 FOR K=1 TO RW
450   READ W$
460 NEXT K

```

Don't RUN it yet.
We aren't finished.

```

1000 REM ** VALUE OF NW AND NW WORDS
1010 DATA 44
1020 DATA AHA, ARK, BAH, CAT, DIG, DUO
1030 DATA EBB, ELF, FLY, FUN, GNU, HAW
1040 DATA HEX, JET, JOY, LAX, LEU, MEW
1050 DATA MUD, NIX, NUT, OAF, ODD, ORB
1060 DATA PAL, PLY, RAM, RED, ROC, SHH
1070 DATA SKY, SOL, TAX, TOO, UGH, VIA
1080 DATA WAG, WAX, WHO, WOW, YAK, YOU
1090 DATA ZIP, ZOO

```

Lines 1020-1090
contain 44 words.

In line 320, the computer reads the number of words (NW). Since line 320 is preceded by a RESTORE, the value of NW will be the first item in the first DATA statement.

- (a) What is the value of NW? _____
- (b) In line 430, what are the possible values of RW? _____
- _____
- (c) Suppose RW is 13. What will be the value of W\$ after completion of the FOR-NEXT loop in lines 440 through 460? _____
- _____
- (d) How would you change the program so the computer uses *your* list of words? _____
- _____
- _____
- _____

(a) 44; (b) 1 to 44, inclusive; (c) HEX; (d) Put *your* words in lines 1020, 1030, and so on. Then put the number of words (value of NW) in line 1010.

7. The computer has selected a random word from its DATA list. What next?

```
500 REM ** GET GUESS
510 PRINT: INPUT "YOUR GUESS"; G$

600 REM ** IF NOT CORRECT, GIVE CLUE
610 IF G$<W$ THEN PRINT "TRY HIGHER WORD": GOTO 510
620 IF G$>W$ THEN PRINT "TRY LOWER WORD": GOTO 510
```

Aha! If the guess (G\$) is "too low," then $G\$ < W\$$ is true and the computer prints: TRY HIGHER WORD. But if the guess is too "high," then $G\$ < W\$$ is false and $G\$ > W\$$ is true. In this case, the computer prints: TRY LOWER WORD.

A lower word is lower in the alphabet (towards AAA).
A higher word is higher in the alphabet (towards ZZZ).

- (a) Which is lower, JAM or RED? _____
 (b) Which is lower, CAT or CAR? _____
 (c) Which is higher, ANT or ZOO? _____

 (a) JAM; (b) CAR; (c) ZOO



8. When you guess the computer's word, this happens:

```
700 REM ** WINNER!
710 PRINT "THAT'S IT! YOU GUESSED MY WORD."
720 SOUND 89, 20: SOUND 108, 20: SOUND 125, 20
730 SOUND 176, 40

800 REM ** TELL HOW TO PLAY AGAIN
810 PRINT
820 PRINT "TO PLAY AGAIN, PRESS ANY KEY."
830 IF INKEY$ = "" THEN 830 ELSE 710
```


Put it all together, enter the program, and play. You may wish to change the hints in lines 510 and 520. Make that easier to do by these changes to the program.

```
110 M1$ = "TRY HIGHER WORD" } Try some other messages
120 M2$ = "TRY LOWER WORD" } here.
610 IF G$<W$ THEN PRINT M1$: GOTO 510
620 IF G$>W$ THEN PRINT M2$: GOTO 510
```

9. For the very young, a game called GUESS MY LETTER. Here is the first piece.

```
100 REM ** GUESS MY LETTER
110 ABC$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
120 M1$ = "TRY HIGHER LETTER"
130 M2$ = "TRY LOWER LETTER"

200 REM ** TELL HOW TO PLAY
210 CLS
220 PRINT "I'LL THINK OF A LETTER, A TO Z."
230 PRINT "YOU GUESS MY LETTER."
240 PRINT
250 PRINT "MY 'LOWEST' LETTER IS A."
260 PRINT "MY 'HIGHEST' LETTER IS Z."
```

Describe line 110. _____

Line 110 assigns to the string variable ABC\$ a string value consisting of all 26 letters of the alphabet. Thus, ABC\$ is a string in which the first character is A, the second character is B, the third character is C, and so on. The 26th character is Z.

10. Perhaps you have guessed that we are going to tell the computer to select, at random, *one letter* from ABC\$. To do so, we will use the MID\$ function, as follows.

```
200 REM ** PICK RANDOM LETTER FROM ABC$
310 RL = RND(26)
320 L$ = MID$(ABC$, RL, 1)
```

- (a) What are the possible values of RL? _____
 (b) Where else does the numeric variable RL appear? _____

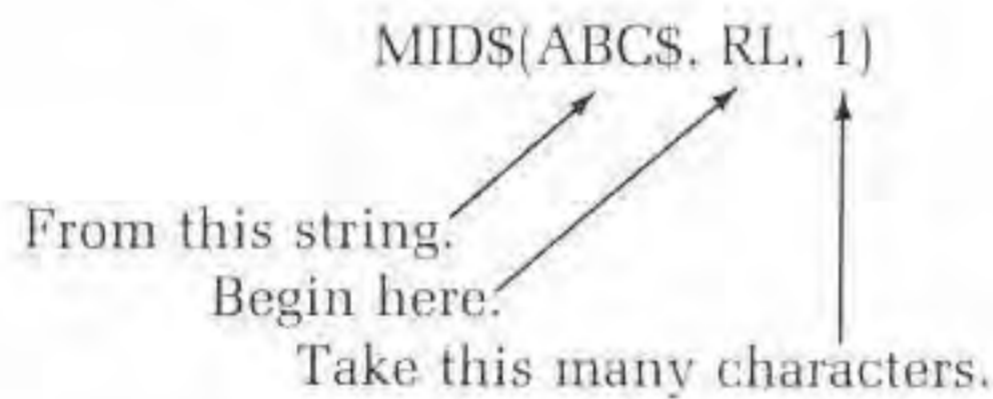
The MID\$ function in line 320 selects a one-character substring from ABC\$. This one-character substring is a single letter and is assigned to L\$.

Which letter? Why, of course, the RLth letter. If RL is 1, the letter A is assigned to L\$. If RL is 2, the letter B is assigned to L\$. If RL is 3, the letter C is assigned to L\$.

- (c) Guess. If RL is 13, what letter is assigned to L\$? — What if RL is 25? —
 26? —
 (d) Guess. Why does MID\$ select only one letter, instead of a bunch of
 letters? _____

 (e) Guess. Why does MID\$ select a substring (whatever *that* is!) from
 ABC\$ instead of from some other string? _____

(a) 1 to 26; (b) MID\$(ABC\$, RL, 1); (c) M; Y; Z; (d), (e) The MID\$ function tells the computer to take a *substring* of one or more characters from a string.



11. Complete the program in a fashion similar to GUESS MY TONE and GUESS MY WORD.

```

400 REM ** GET A GUESS

500 REM ** IF NOT CORRECT, GIVE HINT

600 REM ** WINNER

700 REM ** TELL HOW TO PLAY AGAIN

```

We did it this way:

```

410 PRINT: INPUT "YOUR GUESS": G$
510 IF G$ < L$ THEN PRINT M1$: GOTO 410
520 IF G$ > L$ THEN PRINT M2$: GOTO 410

610 PRINT
620 PRINT "YOU GUESSED MY LETTER—IT WAS" L$
630 FOR K=1 TO 1000
640   PRINT (@RND(510)). L$:
650 NEXT K

710 CLS
720 PRINT "TO PLAY AGAIN, PRESS ANY KEY."
730 IF INKEY$ = "" THEN 730 ELSE 210

```

12. Boggled by MIDS? Try this ancient game called *Words within Words*.

—The word PROVERB contains these shorter words.

PRO PROVE ROVE ROVER OVER VERB

Now let WORDS = "PROVERB"

—MIDS(WORDS, 1, 3) is "PRO" PROVERB
 —MIDS(WORDS, 3, 4) is "OVER" PROVERB
 —MIDS(WORDS, 4, 4) is "VERB" PROVERB

- (a) What is MIDS(WORDS, 1, 5)? _____
 (b) What is MIDS(WORDS, 2, 5)? _____
 (c) Write the MIDS function for "ROVE": _____
 (d) Write the MIDS function for "PROVERB": _____

 (a) PROVE; (b) ROVER; (c) MIDS(WORDS, 2, 4); (d) MIDS(WORDS, 1, 7)

13. Deep down inside the computer, each keyboard character has its very own numeric code, called an ASCII code. ASCII means American Standard Code for Information Interchange. Almost all computers use it—the Color Computer uses it—so we will use it.

—The ASCII code for A is 65.
 —The ASCII code for B is 66.
 —The ASCII code for C is 67.

You can guess the ASCII code for D. ____ E? ____ Z? ____

68; 69; 90 The ASCII codes for A through Z are the numbers 65 through 90.

14. BASIC provides a built-in function, called ASC, which gives the ASCII code for any character. Clear the screen and try these:

You type: PRINT ASC("A")
It prints: 65

The letter A enclosed in quotation marks.

You type: PRINT ASC("B")
It prints: 66

You type: PRINT ASC("Z")
It prints: 90

You type: PRINT ASC("*")
It prints: 42

You type: PRINT ASC(" ")
It prints: 32

↑
space

Experiment! Use the following program to find the ASCII codes for keyboard characters.

```
100 REM ** ASCII CODES FOR KEYBOARD CHARACTERS
110 CLS
120 PRINT : INPUT "KEYBOARD CHARACTER" : KEY$
130 CODE = ASC(KEY$)
140 PRINT "THE ASCII CODE IS " CODE
150 GOTO 120
```

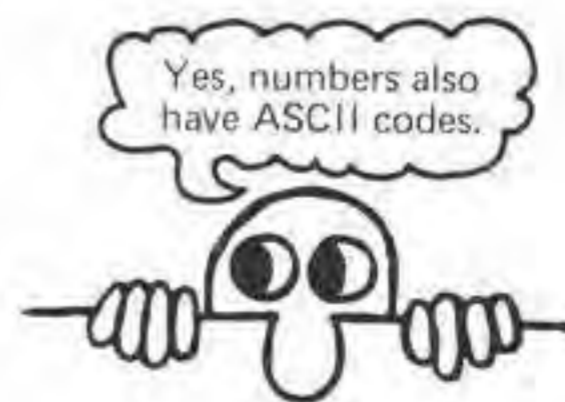
Here is what happened when we ran the program:

```
KEYBOARD CHARACTER? A
THE ASCII CODE IS 65




KEYBOARD CHARACTER? *
THE ASCII CODE IS 42

KEYBOARD CHARACTER? 1
THE ASCII CODE IS 49

KEYBOARD CHARACTER? " "
THE ASCII CODE IS 32
```



To enter a space, you must use quotation marks.

Try some keys yourself. For most keys, the program will work. If your exploration is quite thorough, you may run into some problems. For example, try the , , or  key. For each of these, the computer prints ?FC ERROR IN 130.

For a complete list of ASCII codes, see Appendix 1.

15. ASCII characters can be letters A to Z, or numerals 0 to 9, or punctuation marks (. , ; and so on), or special characters (*, %, and so on), and *lots more*. To help you really explore the world of ASCII, use the CHR\$ function.

CHR\$(CODE)

ASCII code, 0 to 255.

The CHR\$ function gives the character that corresponds to the ASCII code enclosed in parentheses.

You type: PRINT CHR\$(65) You type: PRINT CHR\$(90)
It prints: A It prints: A

You type: PRINT CHR\$(42) You type: PRINT CHR\$(55)
It prints: * It prints: 7

Use Appendix 1 or your Color Computer to help you complete the following.

(a) You type: PRINT CHR\$(66) (b) You type: PRINT CHR\$(34)
It prints: _____ It prints: _____

(a) B; (b) "

16. Experiment. Use the following program to print characters that correspond to ASCII codes.

```
100 CLS
110 INPUT "FIRST ASCII CODE": FIRST
120 INPUT "LAST ASCII CODE": LAST

130 CLS
140 FOR CODE=FIRST TO LAST
150 PRINT CODE; CHR$(CODE);
160 NEXT CODE

170 IF INKEY$="" THEN 170 ELSE 100
```

Here is what happened when we ran the program using 32 for FIRST ASCII CODE and 45 for LAST ASCII CODE:

```

32      33 !
34 "    35 #
36 $    37 %
38 &    39 '
40 (    41 )
42 *    43 +
44 ,    45 -

```

Suppose you want to show the codes and characters for the letters A through Z.

- (a) What value do you enter for FIRST? _____
 (b) What value do you enter for LAST? _____

 (a) 65; (b) 90

To show more codes and characters on the screen, add these statements to the program:

```

135 SP = 0
150 PRINT (@SP, CODE: CHR$(CODE):
155 SP = SP + 8

```

Now the screen will hold up to 64 codes and characters. ASCII codes 128 to 255 are "graphics characters." Use the program to put some of these on the screen. Before you do this, change line 100 to CLS 0.

17. In GUESS MY LETTER (frames 9 and 10), we used the following statements to pick a random letter.

```

110 ABC$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
310 RL = RND(26)
320 L$ = MID$(ABC$, RL, 1)

```

Using CHR\$, we can do it like this, without line 110:

```

310 RL = RND(26) + 64
320 L$ = CHR$(RL)

```

Line 110 is no longer needed. Delete it.

- (a) In line 310, what are the possible values of RL? _____
- (b) In line 320, what are the possible values of L\$? _____

-
- (a) 65 to 90. ($1 + 64 = 65$ and $26 + 64 = 90$)
 - (b) Since the possible values of RL are the ASCII codes for A to Z, the possible values of L\$ are the letters, A to Z.

18. Hmmm . . . ASCII codes for letters are numbers. And tones are numbers. Do you suppose . . .? Of course. Let's make music from words. Actually, we will make music from strings of keyboard characters. We will use the LEN function.

LEN means LENgth. It computes the length of a string.
 LEN ("") is zero (0). The empty string!
 LEN ("A") is one (1).
 LEN ("B") is two (2).
 LEN ("ABC") is three (3).

What is:

- (a) LEN("ABCD")? _____
- (b) LEN("MUSIC")? _____
- (c) LEN ("ABCDEFGHIJKL M")? _____
- (d) LEN ("+-*/")? _____
- (e) LEN("12345679")? _____

(a) 4; (b) 5; (c) 13; (d) 4; (e) 8 Ha! Bet we fooled you. There are 8, not 9, characters between quotation marks.

Spaces count. LEN(" ") is 1. LEN("A B") is 3.

↑
 1 space

↑
 1 space

19. And now . . . TA-RUM, TA-RUM . . . MUSIC FROM ASCII CHARACTERS.

```

100 REM ** MUSIC FROM ASCII CHARACTERS
200 REM ** TALK TO PERSON
210 CLS
220 PRINT "TYPE SOME STUFF, THEN PRESS"
230 PRINT "THE ENTER KEY. I'LL PLAY"
240 PRINT "SOME MUSIC MADE FROM THE STUFF"
250 PRINT "YOU TYPED."

300 REM ** GET SOME STUFF
310 PRINT: INPUT "WHAT SHALL I PLAY": STUFF$

400 REM ** HOW MUCH STUFF
410 LSTUFF = LEN(STUFF$)

500 REM ** PLAY, MAESTRO, PLAY!
510 FOR K=1 TO LSTUFF
520   CH$ = MID$(STUFF$, K, 1)
530   T = ASC(CH$)
540   SOUND T, 10
550 NEXT K

600 REM ** DO IT AGAIN
610 GOTO 210

```

RUN the program. First, try one keyboard character at a time. Some will work—others won't. Try some SHIFT characters.

Then, type a string of several characters. Is it music? You decide.

VARIATION: Change line 610 to GOTO 510. Your "melody" will play again and again and . . .

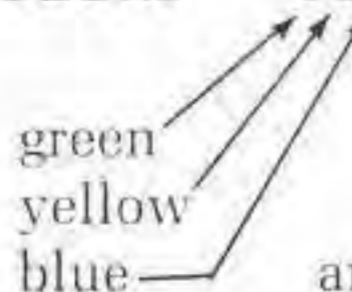
VARIATION: Change line 530 to $T = 5 * (\text{ASC}(\text{CH\$}) - 64)$. Then, use only letters, A to Z, when the computer asks: WHAT SHALL I PLAY?

OK, it's not very musical. Patience. We will try again in Chapter 11.

20. Instead of music, make the screen play colors in response to strings of color numbers. For example,

KOLORS = "12345678087654321"

green
yellow
blue



and so on.

We didn't use COLORS because COLOR is a reserved word.

The ASCII codes for number keys are 48 through 57.

The ASCII code for 0 is 48.

The ASCII code for 1 is 49.

The ASCII code for 2 is 50.

You can guess the ASCII code for 3. _____ 4?
_____ 9? _____

51; 52; 57

Thus, $\text{ASC}("9")$ is 57, $\text{ASC}("1")$ is 49, and so on. Aha! If we subtract 48 from the ASCII code for a number key, we get the number itself.

$\text{ASC}("0") - 48$ is 0

$\text{ASC}("1") - 48$ is 1

$\text{ASC}("2") - 48$ is 2

.

.

.

$\text{ASC}("9") - 48$ is 9



21. Complete the following program to do COLOR FROM STRINGS

```

100 REM ** COLOR FROM STRINGS
200 REM ** TELL WHAT TO DO
210 CLS
220 PRINT "ENTER A STRING CONSISTING"
230 PRINT "ONLY OF NUMBERS, 0 TO 8."
240 PRINT "I'LL 'PLAY' YOUR COLORS."

300 REM ** GET A COLOR STRING
310 PRINT: PRINT "YOUR STRING": KOLOR$

400 REM ** LENGTH OF KOLOR$
410 _____

500 REM ** FLASH THE COLORS IN KOLOR$
510 _____
520 _____
530 _____
540 _____
550 _____
560 _____

600 REM ** DO IT AGAIN      Or GOTO 510 to repeat the color
610 GOTO 210 ←             sequence.

700 REM ** TIME DELAY SUBROUTINE
710 Z = 460
720 FOR K=1 TO Z: NEXT K
730 RETURN

```

```

410 LKOLOR = LEN(KOLOR$)
510 FOR K=1 TO LKOLOR
520   CH$ = MID$(KOLOR$,K,1)
530   KOLOR = ASC(CH$) - 48
540   CLS KOLOR
550   GOSUB 710
560 NEXT K

```

22. Color BASIC provides the function called VAL to convert strings to numbers. Here are some examples:

VAL("1") is 1. VAL("2") is 2.
 VAL("3") is 3. VAL("9") is 9.

So, line 530 in the COLOR FROM STRINGS program can be rewritten, as follows:

```
530 KOLOR = VAL(CH$)
```



Also:

```
VAL("12") is 12.           VAL("123") is 123
VAL("1234") is 1234.      and so on
```

However,

```
VAL("ABC") is 0  Huh?
VAL("123ABC") is 123
VAL("123ABC45") is 123
```

VAL will deal only with numbers, and only if they occur before any non-number. Otherwise, VAL simply ignores you and gives you the old zero sign. Well, that's better than one of those pesky ERROR notices.

Complete the following.

- (a) You type: X\$ = "123"
PRINT VAL(X\$)
It types: _____
- (b) You type: X\$ = "100"
PRINT VAL(X\$)-1
It types: _____

- (a) 123; (b) 99 because VAL(X\$) is 100 and 100 - 1 is 99.
What? You like ERROR notices? Try: PRINT VAL(123)

23. The following table shows the names and abbreviations for characteristics in the role-playing game systems *Dungeons and Dragons!* (D&D), *Runequest* (RQ), and *Tunnels and Trolls* (T&T).

D&D	RQ	T&T
Strength (STR)	Strength(STR)	Strength(STR)
Intelligence(INT)	Intelligence(INT)	Intelligence(IQ)
Wisdom(WIS)	Power(POW)	Luck (LK)
Constitution(CON)	Constitution(CON)	Constitution(CON)
Dexterity(DEX)	Dexterity(DEX)	Dexterity(DEX)
Charisma(CHA)	Charisma(CHA)	Charisma(CHR)
	Size(SIZ)	

Here is the beginning of a program to roll a character in any of the game systems:

```

100 REM ** CREATE A CHARACTER FOR
110 REM ** D&D, RUNEQUEST, OR T&T

200 REM ** CHARACTERISTICS STRINGS
210 DD$ = " 6  STR INT WIS CON DEX CHA "
220 RQ$ = " 7  STR INT POW CON DEX CHA SIZ "
230 TT$ = " 6  STR IQ LK CON DEX CHR "

```

In lines 210 through 230, each string variable (DD\$, RQ\$, TT\$) holds a string consisting of the characteristic abbreviations for one game.

Each characteristic abbreviation is a *substring*. Each string also has, as a substring, the number of characteristics in the string.

Within each string, exactly four positions are used to hold each individual substring.

```

220 RQ$ = " [7] [STR] INT [POW] CON DEX CHA [SIZ] "

```

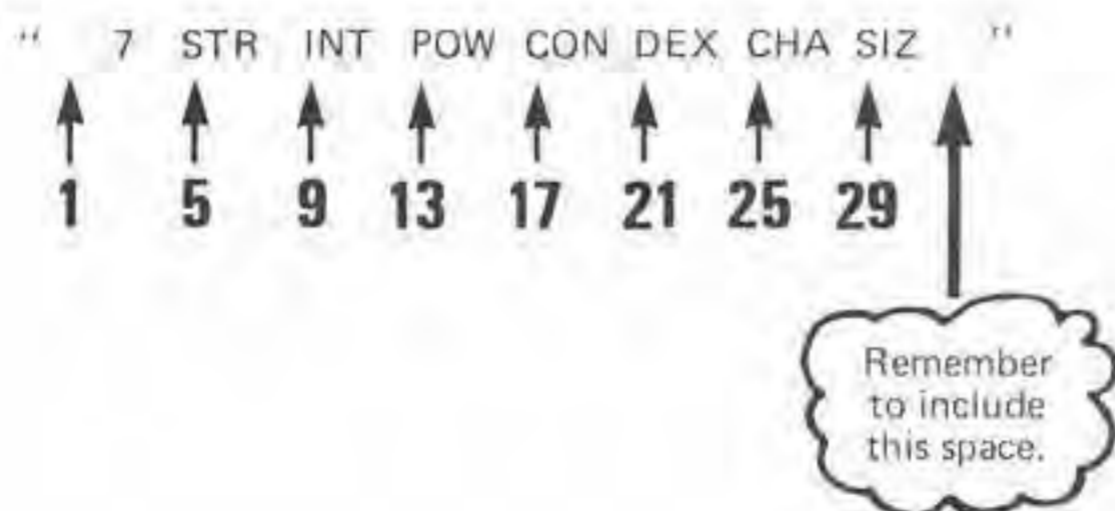
↙ ↘
↙ ↘
↙ ↘

4 positions
4 positions
4 positions

In line 220, positions 1 through 4 consist of: space, space, 7, space. STR is in positions 5 through 8 (S, T, R, space).

- (a) INT is in positions _____ through _____ (I, N, T, space).
 (b) SIZ is in positions _____ through _____ (S, I, Z, space).

 (a) 9;12; (b) 29;32



24. Let's move on. Here is more of the program.

```
300 REM ** TELL ABOUT PROGRAM
310 CLS
320 PRINT "I CAN CREATE A CHARACTER FOR"
330 PRINT
340 PRINT " DUNGEONS AND DRAGONS (DD)"
350 PRINT " RUNEQUEST           (RQ)"
360 PRINT " TUNNELS AND TROLLS  (TT)"

400 REM ** FIND OUT WHICH GAME
410 PRINT
420 PRINT "WHICH GAME (DD, RQ, OR TT)"; GAME$
430 IF GAME$ = "DD" THEN CH$ = DD$: GOTO 510
440 IF GAME$ = "RQ" THEN CH$ = RQ$: GOTO 510
450 IF GAME$ = "TT" THEN CH$ = TT$: GOTO 510
460 PRINT "I ONLY KNOW DD, RQ, OR TT."
470 GOTO 410
```

In line 420, the computer asks which game you want.

- (a) If you enter DD, what happens? _____

- (b) If you enter RQ, what happens? _____

- (c) If you enter TT, what happens? _____

- (d) If you enter MONOPOLY, what happens? _____

-

- (a) The condition in line 430 is true. The computer sets CH\$ equal to DD\$ and goes on to line 510.
- (b) The condition in line 440 is true. The computer sets CH\$ equal to RQ\$ and goes on to line 510.
- (c) The condition in line 450 is true. The computer sets CH\$ equal to TT\$ and goes on to line 510.
- (d) The conditions in lines 430, 440, and 450 are all false. The computer gets to line 460, prints I ONLY KNOW DD, RQ, AND TT, then goes back to line 410 and patiently asks again: WHICH GAME (DD, RQ, OR TT)?
-

25. If you enter DD, RQ, or TT, the computer moves on to line 510 after first setting CH\$ to the appropriate characteristic string. Time to roll a character.

```

500 REM ** PRINT A HEADING
510 CLS
520 PRINT "HERE IS YOUR" CH$ "CHARACTER"
530 PRINT

600 REM ** ROLL THE CHARACTERISTICS
610 NC$ = MID$(CH$, 1, 4)
620 NC = VAL(NC$)
630 FOR K=1 TO NC
640   C = 4*K + 1
650   DICE = RND(6) + RND(6) + RND(6)
660   PRINT MID$(CH$, C, 1), DICE
670 NEXT K

```



Remember, CH\$ is set to DDS, RQS, or TT\$ in lines 430 through 450. For example, if you enter RQ in response to line 420, then CH\$ will be set equal to RQ\$.

CH\$ = " 7 STR INT POW CON DEX CHA SIZ "

Suppose CH\$ is as shown above.

- (a) What is the value of NC\$ in line 610? _____
 (b) What is the value of NC in line 620? _____

(a) " 7 "; (b) 7

26. If NC is 7, the FOR-NEXT loop in lines 630 through 670 will be done for K=1 to 7.

K=1 $C = 4 * K + 1 = 4 * 1 + 1 = 5$
 MID(CH$, C, 4) = MID$(CH$, 5, 4) = "STR"$

K=2 $C = 4 * K + 1 = 4 * 2 + 1 = 9$
 MID(CH$, C, 4) = MID$(CH$, 9, 4) = "INT"$

Complete the following.

K=3

(a) $C = 4 * K + 1 = \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$

(b) $MID\$(CH\$, C, 4) = \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$

K=7

(c) $C = \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$

(d) $MID\$(CH\$, C, 4) = \underline{\hspace{2cm}} = \underline{\hspace{2cm}}$

- (a) $4 * 3 + 1$; 13
- (b) $MID\$(CH\$, 13, 3)$; "POW"
- (c) $4 * 7 + 1$; 29
- (d) $MID\$(CH\$, 29, 4)$; "SIZ"

Finally, the last piece of the program.

```

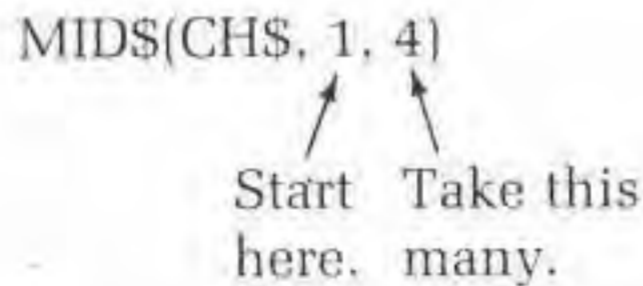
700 REM ** ANOTHER CHARACTER?
710 PRINT
720 PRINT "FOR ANOTHER CHARACTER."
730 PRINT "PRESS THE SPACE BAR."
740 K$ = INKEY$: IF K$ = "" THEN 740
750 IF K$=" " THEN 310 ELSE 740
    
```



27. Here again is line 610 of the program shown in the preceding few frames:

```
610 NC$ = MID$(CH$, 1, 4)
```

The MID\$ function, in this case, chooses the *leftmost* four characters of CH\$.



We could have written line 610 as follows:

```
610 NC$ = LEFT$(CH$, 4)
```


As you have probably guessed, the LEFT\$ function picks off a *left* substring. LEFT\$(CH\$, 4) consists of the leftmost four characters of CH\$. Suppose

A\$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Then

LEFT\$(A\$, 1) is "A"
 LEFT\$(A\$, 2) is "AB"
 LEFT\$(A\$, 3) is "ABC"

- (a) What is LEFT\$(A\$, 4)? _____
 (b) What is LEFT\$(A\$, 7)? _____

 (a) "ABCD"; (b) "ABCDEFG"

28. It's only fair that, in addition to LEFT\$ and MID\$, the computer also provides RIGHT\$.

A\$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

RIGHT\$(A\$, 1) is "Z"
 RIGHT\$(A\$, 2) is "YZ"
 RIGHT\$(A\$, 3) is "XYZ"

- (a) What is RIGHT\$(A\$, 4)? _____
 (b) What is RIGHT\$(A\$, 7)? _____

 (a) "WXYZ"; (b) "TUVWXYZ"

29. When doing arithmetic, the computer uses + to add. This same symbol (+) may be used to *catenate*, or *join together*, two or more strings. Try this program:

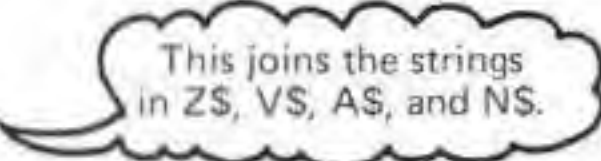
```
100 REM ** MAKE A SENTENCE
110 Z$ = "THE DRAGON"

200 REM ** ASK FOR WORDS
210 CLS
220 PRINT
230 PRINT "PLEASE GIVE ME:"
240 INPUT "      A VERB": V$
250 INPUT "      A NOUN": N$
260 INPUT "    AN ARTICLE": A$
270 INPUT "  AN ADJECTIVE": M$

300 REM ** CATENATE STRINGS
310 SENT$ = Z$ + V$ + A$ + N$

400 REM ** PRINT THE SENTENCE
410 PRINT
420 PRINT SENT$

500 REM ** PRESS ANY KEY TO DO AGAIN
510 IF INKEY$ = "" THEN 510 ELSE 210
```



This joins the strings
in Z\$, V\$, A\$, and N\$.

Line 310 catenates (puts together) the strings Z\$, V\$, A\$, and N\$, then assigns the resulting string to SENT\$.

Here is what happened when we tried to program:

```
PLEASE GIVE ME:
      A VERB?  ATE
      A NOUN?  KNIGHT
    AN ARTICLE?  THE
  AN ADJECTIVE?  LAZY
THE DRAGONATEHELAZYKNIGHT
```

Oops! A little crowded. We forgot to include spaces. Please fix the program.

You could change line 310, as follows:

```
310 SENT$ = Z$ + CHR$(32) + V$ + CHR$(32) + A$ + CHR$(32) + N$
           ↑           ↑           ↑
           space      space      space
```

Or, like this:

```
120 S$ = CHR$(32)
310 SENT$ = Z$ + S$ + V$ + S$ + A$ + S$ + N$
```

30. Complete the following program to generate random three-letter words. Each word should consist of a consonant, a vowel, and a consonant. Some will be real dictionary words; others won't be.

```
100 REM ** WORD MAKER
110 V$ = "AEIOU"
120 C$ = "BCDFGHJKLMNPQRSTVWXYZ"
130 CLS

200 REM ** RANDOM CONSONANT, VOWEL, CONSONANT
210 R = RND(21)

220 L1$ = _____
230 R = RND(5)

240 L2$ = _____
250 _____
260 _____

300 REM ** PUT LETTERS TOGETHER & PRINT WORDS
310 WORD$ = _____

320 PRINT WORD$ CHR$(32) :
400 REM ** GO DO ANOTHER
410 GOTO 210
```

This prints a space after WORD\$.

```
220 L1$ = MID$(C$,R,1)
240 L2$ = MID$(V$,R,1)
250 R = RND(21)
260 L3$ = MID$(C$,R,1)
310 WORD$ = L1$ + L2$ + L3$
```

VARIATION: Five-letter words: Consonant, vowel, consonant, vowel, consonant.

VARIATION: Five-letter words: Consonant, consonant, vowel, consonant, vowel.

31. Try this program to make five-letter words consisting of consonant, vowel, consonant, vowel, consonant.

```

100 REM ** WORD MAKER
110 V$ = "AEIOUY"
120 C$ = "BCDFGHJKLMNPQRSTVWXYZ"
130 CLS

200 REM ** BEGIN WITH WORD$ = EMPTY STRING
210 WORD$ = ""

300 REM ** MAKE A WORD
310 GOSUB 810
320 GOSUB 910
330 GOSUB 810
340 GOSUB 910
350 GOSUB 810

500 REM ** PRINT THE WORD & 3 SPACES
510 PRINT WORD$ CHR$(32) CHR$(32) CHR$(32)

600 REM ** GO MAKE ANOTHER
610 GOTO 210

800 REM ** SUBROUTINE: ADD CONSONANT
810 R = RND(21)
820 CONS$ = MID$(C$, R, 1)
830 WORD$ = WORD$ + CONS$
840 RETURN

900 REM ** SUBROUTINE: ADD VOWEL
910 R = RND(6)
920 VOW$ = MID$(V$, R, 1)
930 WORD$ = WORD$ + VOW$
940 RETURN

```

We will use Y as both
vowel and consonant.

Consonant, vowel, consonant,
vowel, consonant.

Use this program to experiment with different arrangements of vowels and consonants. Invent your own word-making rules. Remember, you can use IF statements and other features of BASIC to tell the computer about your rules. Try these (C means consonant, V means vowel):

CCVCVCVC
VCCVCV

CVVCC
VCVCVC

Self-Test

Use your string magic to wrap-up this Self-Test.

1. Start with easy stuff by answering these questions or completing the sentences.
 - (a) What does CLEAR 600 tell the computer to do? _____
 - (b) What does RESTORE tell the computer to do? _____
 - (c) Which is true: "STAR" < "STARS" or "STARS" < "STAR"? _____
 - (d) Which is false: "ZZZY" < "ZZZ" or "ZZZ" < "ZZZY"? _____
 - (e) What is LEN("ABCDEFGH")? _____
 - (f) What is LEN("")? _____
 - (g) What is ASC(CHR\$(43))? _____
 - (h) What is CHR\$(ASC("A"))? _____
 - (i) Since ASC("*") is 42, then CHR\$(42) must be _____

2. If WS = "DRAGONSMOKE", what is:
 - (a) LEFT\$(WS, 6)? _____
 - (b) RIGHT\$(WS, 5)? _____
 - (c) MID\$(WS, 2, 3)? _____
 - (d) MID\$(WS, 9, 2)? _____
 - (e) MID\$(WS, 4, 6)? _____
 - (f) LEFT\$(WS, 6) + RIGHT\$(WS, 5)? _____
 - (g) LEFT\$(WS, 7) + CHR\$(32) + RIGHT\$(WS, 5)? _____
 - (h) RIGHT\$(WS, 5) + CHR\$(44) + CHR\$(32) + LEFT\$(WS, 7) + CHR\$(33)? _____
 - (i) RIGHT\$(WS, 5) + ", " + LEFT\$(WS, 7) + "!"? _____
 - (j) MID\$(WS, 1, 6)? _____
 - (k) MID\$(WS, 7, 5)? _____
 - (l) LEN(WS)? _____
 - (m) LEN(LEFT\$(WS, 6))? _____
 - (n) LEN(RIGHT\$(WS, 5))? _____
 - (o) LEN(MID\$(WS, 4, 6))? _____
 - (p) MID\$(WS, 7, LEN(WS) - 6)? _____

3. What is the printed result of the following program? *Don't RUN it. Look at it, puzzle over it, figure out what it does.*

```
100 REM ** MYSTERY PROGRAM
200 REM ** HERE IS THE MYSTERY
210 NW = 0
220 READ W$
230 IF W$ <> "ZZZ" THEN NW = NW + 1 : GOTO 220

300 REM ** PRINT THE RESULT AND STOP
310 CLS
320 PRINT NW
330 END

1000 REM ** WORDS
1010 DATA ARK, CAT, ELF, FUN, GNU
1020 DATA HEX, JET, JOY, MEW, MUD
1030 DATA NUT, OAF, PAL, RAM, RED
1040 DATA SOL, TOO, UGH, VIA, WAG
1050 DATA YAK, ZIP, ZOO, ZZZ
```

The computer prints one number. What is it? _____

4. Now, using what you learned in question 3, rewrite the GUESS MY WORD game in frames 6 through 8. Follow the outline of REM statements.

```
100 REM ** GUESS MY WORD
200 REM ** COMPUTE NW=NUMBER OF WORDS
300 REM ** TELL HOW TO PLAY
400 REM ** PICK RANDOM WORD FROM DATA
500 REM ** GET GUESS
600 REM ** IF NOT CORRECT, GIVE CLUE
700 REM ** WINNER!
800 REM ** TELL HOW TO PLAY AGAIN
1000 REM ** LIST OF WORDS + ZZZ
```

Put as many three-letter words here
as you want with ZZZ after last word.

5. Write a program to reverse a three-character string. A RUN might go like this:

```
YOUR STRING? ABC
THE REVERSE IS CBA

YOUR STRING? HA
MUST BE 3 CHARACTERS

YOUR STRING? ++
THE REVERSE IS ++

YOUR STRING? "HA "
THE REVERSE IS  AH
```

Accept *only* strings
of LENgth equal to 3.

and so on.

6. Complete the following program to change screen color according to color numbers (0 to 8) packed into a string. The number 9 is an "end of data" flag. When it is sensed, the computer should repeat the colors in the string.

```
100 REM ** STRING COLORS
110 KOLOR$ = "1234567809"

200 REM ** SHOW COLORS IN KOLOR$
210 K = 1
220 C$ = MID$(KOLOR$, K, 1)
230 IF C$ = "9" THEN 210

240 C = _____
250 CLS _____
260 _____
270 K = _____
280 GOTO 220
```

To change the sequence of
colors, rewrite line 110.

7. WORD'S WORTH #1. Assign number scores to letters, as follows:

A = 1	G = 7	M = 13	S = 19	Y = 25
B = 2	H = 8	N = 14	T = 20	Z = 26
C = 3	I = 9	O = 15	U = 21	
D = 4	J = 10	P = 16	V = 22	
E = 5	K = 11	Q = 17	W = 23	
F = 6	L = 12	R = 18	X = 24	

A word's worth is its numerical value, obtained by adding the values of the letters in the word. For example, HOBBIT is worth 56 points, DRAGON is worth 59 points, and WIZARD is worth 81 points.

Write a program to compute a word's worth. A RUN of your program might look like this:

```
YOUR WORD? WIZARD
YOUR WORD IS WORTH 81 POINTS

YOUR WORD? ISN'T
YOUR WORD IS WORTH 62 POINTS

YOUR WORD? FLIP-FLOP
YOUR WORD IS WORTH 92 POINTS

YOUR WORD? ABCD
YOUR WORD IS WORTH 10 POINTS

YOUR WORD? 3#AB%*Z
I DON'T UNDERSTAND

YOUR WORD? and so on . . .
```

Your program should compute the score, or worth, of any word or even any string of letters, even if it isn't a word.

Contractions and hyphenated words are OK.

Your program should accept strings which include:

- Letters, A through Z
- Apostrophes (')
- Hyphens (-)
- Spaces

THIRTEEN is worth 99 points. Are there numbers whose English name has a word's worth equal to the number? If so, what is the smallest such number?

AHA is worth 10 points. AHA is a palindrome, but . . . sigh . . . 10 is not a palindrome. Are there words that are palindromes for which the word's worth is also a palindrome?

8. WORD'S WORTH #2. This problem is inspired by a contest that appeared in *Games Magazine* some time ago. Assign number values to letters as in question 7. Compute a WORD'S WORTH by multiplying the number values.

HOBBIT is worth $8 \times 15 \times 2 \times 2 \times 9 \times 20 = 86,400$ points

DRAGON is worth $4 \times 18 \times 1 \times 7 \times 15 \times 14 = 105,840$ points

WIZARD is worth $23 \times 9 \times 26 \times 1 \times 18 \times 4 = 387,504$ points

Write a program to compute a word's worth. Remember, contractions and hyphenated words are OK.

Can you find a word whose word's worth is exactly 1000? Exactly 10,000? Exactly 100,000? Exactly 1,000,000?

Here is a game for two or more people. Pick a target number, 1000 or 10,000 or 1,000,000 or 1,234,567 or whatever you agree on. Agree on a dictionary. Pick a word from that dictionary and enter it. Winner is the person whose word is closest to the target number.

9. SCRABBLE SCORES. In the game of SCRABBLE, each letter has a number score, as shown below.

A = 1	G = 2	M = 3	S = 1	Y = 4
B = 3	H = 4	N = 1	T = 1	Z = 10
C = 3	I = 1	O = 1	U = 1	
D = 2	J = 8	P = 3	V = 4	
E = 1	K = 5	Q = 10	W = 4	
F = 4	L = 1	R = 1	X = 8	

A word score is the sum of the scores of the letters in the word. For example, HOBBIT is worth 13 points and WIZARD is worth 19 points.

Write a program to compute the score for a word, or string of letters, using the letter scores shown above. A RUN of your program might look like this:

```
YOUR WORD?      HOBBIT
YOUR SCORE IS 13

YOUR WORD?      WIZARD
YOUR SCORE IS 19

YOUR WORD?      3#AB%
I DON'T UNDERSTAND
```

And so on. The computer should accept any "word" consisting of a string of letters, even if it is not a real dictionary word, but reject any string containing characters other than the letters A through Z.

10. EXOTIC NAMES IN FANTASY ADVENTURELAND

Suppose you are creating a fantasy adventure such as a *Dungeons and Dragons*, *Runequest*, or *Tunnels and Trolls*. You may wish to use unusual names for your heroes, wizards, monsters and other creatures. You could, of course, borrow names from fantasy adventure books such as *Lord of the Rings*. But perhaps you prefer to invent your own.

Why not use your computer to help you invent names? Sounds OK, but how do we get the computer to print names that are pronounceable (or almost so) and seem to be unusual, exotic, or even fantastic?

That's the problem. Write a program to generate and print or display random names that might be used in a fantasy or science fiction game or story.

Yes, this does sound more like a *project* than a *problem*. Good! We hope you will project your best efforts into this problem . . . er, project . . . and send us some cunningly contrived programs to generate fantastic names. Let's see now—how many orcs would have to type for how many years to write *Lord of the Rings*?

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

1. (a) Reserve 600 characters of memory space for strings. (frame 1)
- (b) Reset the DATA pointer to the first item in the first DATA statement. (frame 4)
- (c) "STAR"<"STARS" is true. "STARS"<"STAR" is false. (frame 7)
- (d) "ZZZY"<"ZZZ" is false. "ZZZ"<"ZZZY" is true. (frame 7)
- (e) 7 (frame 18)
- (f) 0
- (g) 43
- (h) A
- (i) *
- (g), (h), (i) (frames 13–15)
2. (a) DRAGON
- (b) SMOKE
- (c) RAG
- (d) OK
- (e) GONSMO (This is OMSNOG spelled backwards.)
- (f) DRAGONSMOKE
- (g) DRAGONS SMOKE
- (h) SMOKE, DRAGONS!
- (i) SMOKE, DRAGONS!
- (j) DRAGON
- (k) SMOKE
- (l) 11
- (m) 6
- (n) 5
- (o) 6
- (p) SMOKE

You can verify any answer on the computer. For example, to verify our answer for (m),

```
You type:          WS = "DRAGONSMOKE"
You type:          PRINT MIDS(WS, 7, LEN(WS) - 6))
It prints:         SMOKE
(frames 10, 12–16, 18, 27–29)
```

3. 23. This program counts the words in DATA statements up to, but not including, ZZZ. (no particular frame)
4. The program is similar to the program in frames 6–8. Here are the additions and changes:

```

200 REM ** COMPUTE NW=NUMBER OF WORDS
210 NW = 0
220 READ W$
230 IF W$ <> "ZZZ" THEN NW=NW+1: GOTO 220

```

Look familiar?
See question 3!

Block 300 (frame 6) is unchanged.

Block 400 (frame 6). Delete line 420 (420 READ NW) since NW is now computed in block 200.

Blocks 500, 600, 700, and 800 (frames 7,8) are unchanged.

Block 1000 (frame 6). Delete line 1010 (1010 DATA 44) and change line 1090 to:

```
1090 DATA ZIP, ZOO, ZZZ ← End of data flag.
```

Or, put your own list of words in DATA statements following line 1000. (frames 6–8, question 4)

5. We did it this way:

```

100 REM ** REVERSE 3-CHARACTER STRING
110 CLS

200 REM ** GET AND CHECK STRING
210 PRINT: INPUT "YOUR STRING": S$
220 IF LEN(S$)=3 THEN 310
230 PRINT "MUST BE 3 CHARACTERS"
240 GOTO 210

300 REM ** RVS IS REVERSE OF SS
310 RV$ = RIGHTS$(S$, 1) + MIDS$(S$, 2, 1) + LEFT$(S$, 1)

400 REM ** PRINT REVERSE STRING
410 PRINT "THE REVERSE IS" RV$

500 REM ** GO BACK FOR MORE
510 GOTO 210

```

Hmmm . . . how would you tell the computer to print the reverse of a string with 4 characters? 5 characters? as many characters as you enter? Try your hand at these larger problems. (frames 10, 12, 18, 27–29)

```
6. 240 C = VAL(C$)
    250 CLS C
    260 FOR K=1 TO 100: NEXT K
    270 K = K + 1
```

Try some other color sequences:

```
110 KOLOR = "10203040506070809"
110 KOLOR$ = "324100"
```

etc.

Make it easier to change the sequence of colors. Use an INPUT statement to get the value of KOLOR\$. (frame 22)

7,8,9,10. We leave these for your amusement. You should have little trouble with problems 7 and 8. Problem 9 might cause a few wrinkles. You can solve it, though, with BASIC tools from the first 10 chapters. There are at least two elegant, short programs to do it.

In problem 10, use your dictionary as a resource. How might you make names sound Roman, or Norse, or Celtic, or Elvish?

If all else fails, yell for HELP! Send a self-addressed stamped envelope to

Dymax Gazette
P.O. Box 310
Menlo Park, CA 94025

CHAPTER ELEVEN

Subscripted Variables

In this chapter you will learn to use *subscripted variables* and bunches of subscripted variables, called *arrays*. You will use arrays of subscripted variables in programs to twinkle stars, play music, simulate coin flipping and dice rolling, and play games. Subscripted variables add a new dimension to your ability to make the computer do what you want it to do.

When you finish this chapter, you will be able to:

- Recognize and use subscripted numeric variables;
- Recognize and use subscripted string variables;
- Use the DIM (for DIMension) statement to reserve memory space for arrays (lists) of subscripted variables;
- Enter numbers into numeric arrays in several ways;
- Enter strings into string arrays in several ways.

Arrays of subscripted variables are powerful and useful tools.

1. You have used simple *numeric variables and string variables*.

—Numeric variables: A B CODE KOLOR T1 X
—String variables: A\$ B\$ GAMES RQ\$ X\$

Now you will meet and learn how to use a new type of variable called a *subscripted variable*. Subscripted variables come in two flavors, shown below.

—Subscripted numeric variable: T(3)
—Subscripted string variable: CH\$(2)

A subscripted variable consists of a variable name followed by a number enclosed in parentheses.

- A(3) is a subscripted variable
- A3 is not a subscripted variable. It is a simple numeric variable.
- N\$(7) is a subscripted variable.
- N\$ is not a subscripted variable. It is a simple string variable.
- N7\$ is not a subscripted variable.

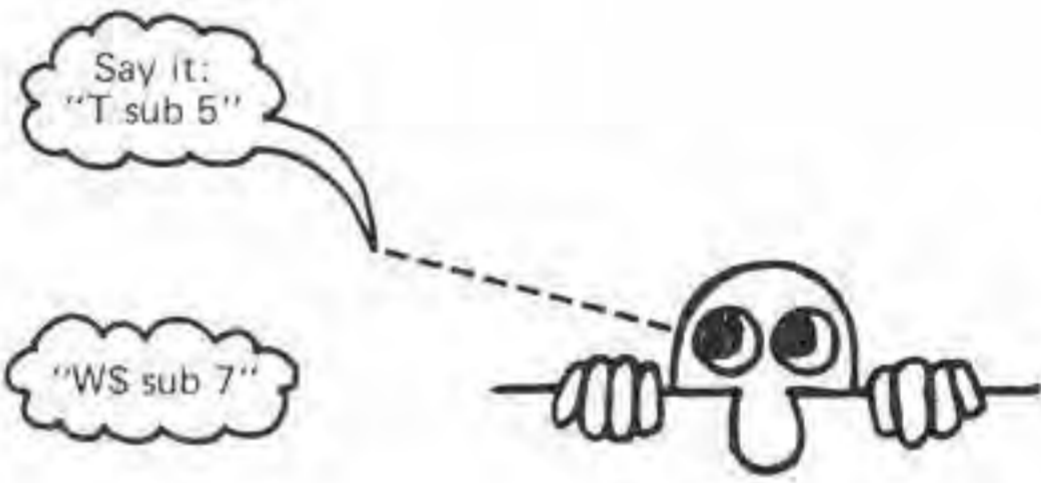
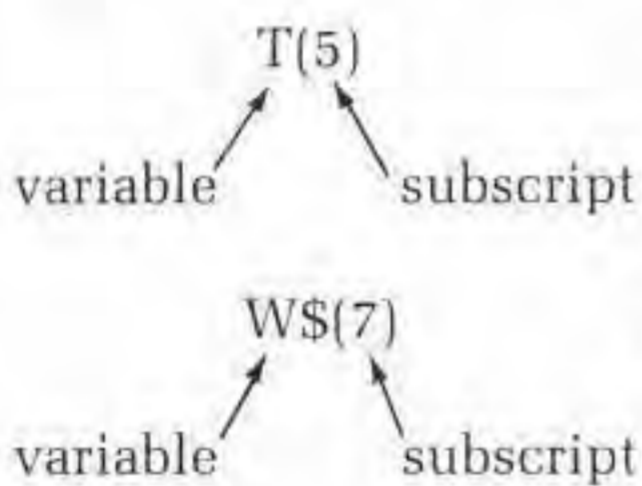
For each of the following, tell whether it is a subscripted variable or a simple variable.

T_____	T1_____
T(1)_____	T1(1)_____
KOLOR(3)_____	KOLOR_____
KEY\$_____	K\$(4)_____

T	simple	T1	simple
T(1)	subscripted	T1(1)	subscripted
KOLOR(3)	subscripted	KOLOR M	simple
KEY\$	simple	K\$(4)	subscripted

2. A subscripted variable consists of:

	Numeric	String
a variable name	T	W\$
left parenthesis	T(W\$(
a subscript	T(5	W\$(7
right parenthesis	T(5)	W\$(7)



The subscript can be a *numeric* variable.

T(R) "T sub R"
 W\$(K) "W\$ sub K"

The subscript can be a *numeric* expression.

T(N + 1) "T sub N plus 1"
 W\$(2*X + 1) "W\$ sub 2 times X plus 1"

For each subscripted variable, write OK or NOT OK.

- (a) KOLOR(C\$)_____
- (b) KOLOR(C)_____
- (c) W\$(L\$)_____
- (d) W\$(K + 1)_____

- (a) NOT OK Subscript must be numeric.
 (b) OK
 (c) NOT OK Subscript must be numeric.
 (d) OK

3. Now you know what subscripted variables look like. But what are they good for? Patience—we are slowly getting to it.

An *array* is a list of subscripted variables.

T(1) This is an *array* of three subscripted
 T(2) variables. Each subscripted variable
 T(3) is an *element* of the array, T.

- (a) In the array: D(1) D(2) D(3) D(4)
 How many elements? _____
 (b) In the array: W\$(0) W\$(1) W\$(2)
 How many elements? _____
 (c) Write the elements of an array C which has seven (7) elements,
 beginning with C(0).

(a) 4; (b) 3; Yes, zero subscripts are OK.

(c) C(0) C(1) C(2) C(3) C(4) C(5) C(6)

Count them. Seven elements, beginning with C(0).

IMPORTANT NOTICE! Subscripts are whole numbers
 0, 1, 2, 3, and so on.



4. Once more, think of the computer's memory as a bunch of boxes, called number boxes (Chapter 4) and string boxes (Chapter 5). Now you also have arrays of subscripted number boxes and arrays of subscripted string boxes. For example, here is an array of number boxes.

T(1)
 T(2)
 T(3)
 T(4)
 T(5)
 T(6)
 T(7)

Seven subscripted variables are
 names for seven number boxes.

Take pencil in hand and assign numbers to the subscripted variables, as follows:

$$\begin{array}{llll} T(1) = 89 & T(2) = 108 & T(3) = 125 & T(4) = 133 \\ T(5) = 147 & T(6) = 159 & T(7) = 170 & \end{array}$$

T(1)	<input type="text" value="89"/>
T(2)	<input type="text" value="108"/>
T(3)	<input type="text" value="125"/>
T(4)	<input type="text" value="133"/>
T(5)	<input type="text" value="147"/>
T(6)	<input type="text" value="159"/>
T(7)	<input type="text" value="170"/>

Do these numbers look familiar?
They are the tone numbers for
Middle C, D, E, F, G, A, and B.

5. Let's write a program that uses subscripted variables to put some twinkling stars on the screen. First, read seven star positions into the variables SP(1), SP(2), SP(3), SP(4), SP(5), SP(6), and SP(7).

```

100 REM ** TWINKLING STARS
110 CLS

200 REM ** READ STAR POSITIONS INTO ARRAY SP
210 FOR STAR = 1 to 7
220   READ SP(STAR)
230 NEXT STAR
240 DATA 198, 171, 207, 243, 250, 313, 308
    
```

In this program, we will not use SP(0).

In the FOR-NEXT loop, when STAR is 1, SP(STAR) is SP(1); when STAR is 2, SP(STAR) is SP(2); when STAR is 3, SP(STAR) is SP (3); and so on—up to SP(7).

After the computer has completed the FOR-NEXT loop, what values will be in SP(1) through SP(7)? Show them below.

SP(1)	<input type="text"/>	SP(2)	<input type="text"/>	SP(3)	<input type="text"/>	SP(4)	<input type="text"/>
SP(5)	<input type="text"/>	SP(6)	<input type="text"/>	SP(7)	<input type="text"/>		

SP(1)	<input type="text" value="198"/>	SP(2)	<input type="text" value="171"/>	SP(3)	<input type="text" value="207"/>	SP(4)	<input type="text" value="243"/>
SP(5)	<input type="text" value="250"/>	SP(6)	<input type="text" value="313"/>	SP(7)	<input type="text" value="308"/>		

To see where the stars will appear on the screen, use a screen map for PRINT positions. Put an asterisk (*) in each screen position.

		COLUMNS																																		
		0	1	2	3	4	5	6	7	8	9	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3			
ROWS		0	32	64	96	128	160	192	224	256	288	320	352	384	416	448	480																			
0	0																																			
1	32																																			
2	64																																			
3	96																																			
4	128																																			
5	160																																			
6	192																																			
7	224																																			
8	256																																			
9	288																																			
10	320																																			
11	352																																			
12	384																																			
13	416																																			
14	448																																			
15	480																																			

6. The star positions are now stored in an array of subscripted variables. Next, we want a program block to turn on the stars.

```

300 REM ** TURN ON THE STARS
310 FOR STAR = 1 TO 7
320   PRINT@SP(STAR), "*";
330 NEXT STAR

```

The computer will turn on a star at screen position SP(1), then turn on a star at SP(2), then turn on a star at SP(3), and so on, until it turns on a star at SP(7). Of course, this will happen very quickly, in a small fraction of a second.

How can you slow things down so you can see the stars coming on, one by one? _____

Put a time delay in the FOR-NEXT loop. For example, try this:

```

325   FOR K=1 TO 500: NEXT K

```

7. The stars are on, but are not yet twinkling. So pick a random star and twinkle it.

```

400 REM ** RANDOM STAR
410 RS = RND(7)

500 REM ** TWINKLE IT OFF
510 PRINT @SP(RS), " ";
520 GOSUB 810

600 REM ** TWINKLE IT ON
610 PRINT @ SP(RS), "*";
620 GOSUB 810

700 REM ** DO IT AGAIN
710 GOTO 410

800 REM ** TIME DELAY SUBROUTINE
810 Z = 200
820 FOR K=1 TO Z: NEXT K
830 RETURN

```

In line 410, the possible values of RS are 1 to 7.

- (a) If RS is 2, what is the value of SP(RS)? _____
 (b) If RS is 7, what is the value of SP(RS)? _____

-
- (a) When RS is 2, SP(RS) is SP(2). The value of SP(2) is 171 (see frame 5).
 (b) When RS is 7, SP(RS) is SP(7). The value of SP(7) is 308 (see frame 5).

8. If you ran the program, or plotted the stars on a screen map, you know that the screen shows the Big Dipper. Two of the stars in the Big Dipper point to the North Star. So, let's add the North Star to the list of stars in the DATA statement.

```

240 DATA 198, 171, 207, 243, 250, 313, 308, 30

```

Now there are eight stars instead of seven. Modify the program so the computer will turn on and twinkle eight stars.

Make these changes:

```
210 FOR STAR=1 TO 8
310 FOR STAR=1 TO 8
410 RS = RND(8)
```

9. Hmm . . . it would be a lot easier to change the star pattern if we put the number of stars (call it NS) in a DATA statement, as follows:

```
Value of NS.
      ↓
250 DATA 8
260 DATA 198, 171, 207, 243, 250, 313, 308, 30
```

Modify the program in frames 5, 6, and 7 so the computer reads the value of NS, then reads the values of SP(1) through SP(NS), then turns on the stars, then twinkles them.

```
200 REM ** READ STAR POSITIONS INTO ARRAY SP
210 _____
220 _____
230 _____
240 _____
250 DATA 8
260 DATA 198, 171, 207, 243, 250, 313, 308, 30
310 _____
410 _____
```

Make these changes to the program in frames 5, 6, and 7.

```
200 REM ** READ STAR POSITIONS INTO ARRAY SP
210 READ NS
220 FOR STAR =1 TO NS
230   READ SP(STAR)
240 NEXT STAR

310 FOR STAR=1 TO NS

410 RS = RND(NS)
```

10. Instead of READ and DATA statements, you can use INPUT statements to enter star positions.

```

100 REM ** TWINKLING STARS
110 CLS

200 REM ** ASK FOR STAR POSITIONS
210 INPUT "HOW MANY STARS": NS
220 PRINT
230 FOR STAR=1 TO NS
240   INPUT "STAR POSITION (0 TO 510)": SP(STAR)
250 NEXT STAR

300 REM ** TURN ON THE STARS
310 CLS
320 FOR STAR=1 TO NS
330   PRINT @ SP(STAR), "*"
340 NEXT STAR

400 REM ** RANDOM STAR, 1 TO NS
410 RS = RND(NS)

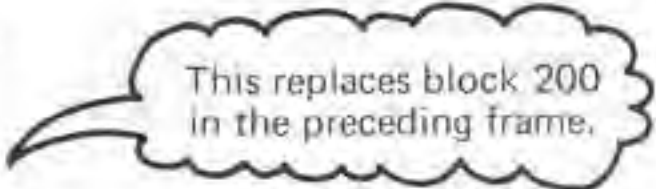
500 REM ** TWINKLE IT OFF
510 PRINT @ SP(RS), " ";
520 GOSUB 810

600 REM ** TWINKLE IT ON
610 PRINT @ SP(RS), "*"
620 GOSUB 810


700 REM ** DO IT AGAIN
710 GOSUB 410

800 REM ** TIME DELAY SUBROUTINE
810 Z = 100
820 FOR K=1 TO Z: NEXT K
830 RETURN

```



This replaces block 200
in the preceding frame.



Change line 810
for faster or
slower twinkling.

Use this program to enter the star positions for the Big Dipper. You will find them in the DATA statement (line 240) in frame 5. A RUN should begin like this:

```

HOW MANY STARS? 7

STAR POSITION (0 TO 510)? 198
STAR POSITION (0 TO 510)? 171

and so on

```

11. And now . . . a galactic encore! We are pleased to present, for your stargazing pleasure, a small spiral galaxy (very small—only 19 stars). The star positions are shown below.

```

47, 85, 105, 121, 175, 178, 186, 204, 229, 240,
243, 267, 273, 281, 333, 342, 368, 371, 391

```

Watch what happens when we enter the star positions.

```

HOW MANY STARS? 19
STAR POSITION (0 TO 510)? 47 ← SP(1)
STAR POSITION (0 TO 510)? 85 ← SP(2)
.
.
.
STAR POSITION (0 TO 510)? 240 ← SP(10)
STAR POSITION (0 TO 510)? 243 ← SP(11)
?BS ERROR IN 240 ← Oops!
OK
■

```

We got a Bad Subscript ERROR in line 240 when we tried to enter a number into SP(11). Unless you tell it otherwise, the computer reserves memory space for subscripts 0 to 10. It refused to accept our value of 243 for SP(11).

Add the following statement to the program in frame 10.

```
120 DIM SP (100)
```

Now you can enter up to 100 star positions into SP(1) to SP(100). RUN the program, enter the 19 spiral galaxy star positions, and stargaze.

The statement:

```
120 DIM SP(100)
```

tells the computer to reserve memory space for the array SP with up to 101 elements. The elements of the array are the subscripted variables SP(0), SP(1), SP(2), and so on, up to SP(100).

```
120 DIM SP(100)
```

↖
Largest subscript
allowed in the
array SP.

A DIM statement should be placed early in a program, before values are put into subscripted variables.

- (a) The statement:

```
130 DIM T(33)
```

reserves space for a numeric array T consisting of (how many?) _____ elements. The smallest possible subscript is _____ and the largest possible subscript is _____.

- (b) The statement:

```
110 DIM WORD$(20)
```

reserves space for a string array WORD\$ consisting of (how many?) _____ elements. The smallest subscript is _____ and the largest is _____.

- (c) Write a DIM statement to reserve space for a numeric array consisting of elements LV(0), LV(1), LV(2), . . . , LV(26).

```
110 _____
```

- (a) 34; 0; 33. The elements are T(0), T(1), T(2), . . . , T(33).
(b) 21; 0; 20. The elements are WORD\$(0), WORD\$(1), . . . , WORD\$(20).
(c) DIM LV(26)

12. In frame 9, you modified the TWINKLING STARS program so the number of stars (NS) is READ from a DATA statement. Our modification looked like this:

```
200 REM ** READ STAR POSITIONS INTO ARRAY SP
210 READ NS
220 FOR STAR=1 TO NS
230   READ SP(STAR)
240 NEXT STAR
250 DATA 7
260 DATA 198, 171, 207, 243, 250, 313, 308
```

Someone has to count the stars and put the value of NS in the first DATA statement. People don't like to count things and sometimes make mistakes. Computers don't mind counting things and rarely make a mistake. So let's have the computer count the stars.

Do this:

I

```

210 NS = 0
220 STAR = 1
230 READ SP(STAR)
240 IF SP(STAR)=-1 THEN 310
250 STAR = STAR + 1
260 NS = NS + 1
270 GOTO 230

900 REM ** STAR POSITIONS
910 DATA 198, 171, 207, 243, 250, 313, 308, -1

```

Test for flag.



Line 250 increases the value of STAR by 1 and line 260 increases the value of NS by 1, each time a new star position is read. Here is another way to do it:

II

```

210 STAR = 0
220 STAR = STAR + 1
230 READ SP(STAR)
240 IF SP(STAR)<> -1 Then 220
250 NS = STAR - 1

900 REM ** STAR POSITIONS
910 DATA 198, 171, 207, 243, 250, 313, 308, -1

```

When the computer finishes either **I** or **II** above, the seven star positions are in SP(1) through SP(7), and NS is equal to 7. What is in SP(8)?

The flag (-1), which, when waving, signals "no more data." We used -1 as the flag because a screen position can never be -1.

13. Now let's apply what we learned about arrays by programming a horrendous tune. Complete the following program to play random music, using an array T of tone numbers entered from the keyboard. Make duration also random. Fill in line 250, then write block 300.

```

100 REM ** RANDOM MUSIC FROM AN ARRAY
110 DIM T(30)

200 REM ** ASK FOR TONE NUMBERS
210 CLS
220 INPUT "HOW MANY TONES": NT
230 PRINT
240 FOR K=1 TO NT
250 _____
260 NEXT K
300 REM ** PLAY A RANDOM TONE

```

```

400 REM ** DO IT AGAIN
410 GOTO 310

```

```

250 INPUT "TONE NUMBER(0 TO 255)": T(K)

```

Here are three ways to write block 300:

```

310 RT = RND(NT)
320 T = T(RT)
330 D = RND(4)
340 SOUND T, D

310 RT = RND(NT)
320 D = RND(4)
330 SOUND T(RT), D

310 T = T(RND(NT))
320 D = RND(4)
330 SOUND T, D

```

Remember:
T and T() are different variables.
T is a simple variable.
T() is a subscripted variable.

14. Hmm... a tone number can't be -1. So use -1 as a flag to signal that there are no more tones, as we did in frame 12. Rewrite block 200 of RANDOM MUSIC FROM AN ARRAY so the computer counts the number of tones, stopping when someone types -1.

```

200 REM ** ASK FOR TONE NUMBERS

```

The rest of the
program is in
frame 13.

We did it like this:


```

210 CLS
220 PRINT "TO QUIT, TYPE -1 AS TONE NUMBER"
230 PRINT
240 NT = 0: K = 1
250 INPUT "TONE NUMBER (0 TO 255)"; T(K)
260 IF T(K) = -1 THEN 310
270 K = K + 1 : NT = NT + 1
280 GOTO 250

```

A RUN might go like this:

```
TO QUIT, TYPE -1 AS TONE NUMBER
```

```

TONE NUMBER (0 TO 255)? 89
TONE NUMBER (0 TO 255)? 108
TONE NUMBER (0 TO 255)? 125
TONE NUMBER (0 TO 255)? 133
TONE NUMBER (0 TO 255)? 89
TONE NUMBER (0 TO 255)? 159
TONE NUMBER (0 TO 255)? 170
TONE NUMBER (0 TO 255)? -1

```

If you especially like a certain tone, you can enter it more than once and hear it more often.



15. For you people with lazy fingers or no coin, here is a program that flips coins and also counts how many Heads (H) and how many Tails (T).

```

100 REM ** COIN FLIPPER AND COUNTER
110 C$(1) = "H "
120 C$(2) = "T "

200 REM ** TALK TO A PERSON
210 CLS
220 INPUT "HOW MANY FLIPS (1 TO 100)"; NF
230 IF NF < 1 THEN 220
240 IF NF > 100 THEN 220

300 REM ** START FROM ZERO
310 N(1) = 0
320 N(2) = 0

400 REM ** FLIP AND COUNT
410 CLS
420 FOR K=1 TO NF
430   FLIP = RND(2)
440   PRINT C$(FLIP);
450   N(FLIP) = N(FLIP) + 1
460 NEXT K

500 REM ** DISPLAY RESULTS
510 PRINT: PRINT
520 PRINT N(1) "HEADS AND" N(2) "TAILS"

600 REM ** DO NOTHING
610 GOTO 610

```

This program is a *simulation*. It causes the computer to *simulate*, or *imitate*, coin flipping.

Go ahead. Enter the program carefully and correctly. Then use it to flip 10 coins, 20 coins, 100 coins. If you don't trust the computer's HEADS and TAILS count, count them yourself.

When you have finished flipping out (oops! sorry), answer the following questions about COIN FLIPPER AND COUNTER.

- (a) In line 230, the computer prints HOW MANY FLIPS (1 TO 100)?. Suppose you enter 101 as your answer. What happens?_____
- (b) In line 430, what are the possible values of FLIP?_____
- (c) Suppose FLIP is 1. In line 440, what does the computer print?_____
- (d) If FLIP is 1, what happens in line 450?_____
- (e) What variable counts the number of heads flipped by the computer?
- (f) What variable counts the number of tails flipped by the computer?

- (a) The computer patiently again asks: HOW MANY FLIPS (1 TO 100)? This happens because line 240 sends the computer back to line 220 whenever NF is greater than 100.
- (b) 1 or 2
- (c) The value of C\$(1). This value is H and a space.
- (d) The value of N(1) is increased by one.
- (e) N(1)
- (f) N(2)

16. Instead of flipping a coin, roll a six-sided die. Don't print each roll on the screen. Instead, as the computer "rolls" the die again and again, count how many 1s, how many 2s, and so on. After NR rolls, print the number of 1s, 2s, and so on, as shown in the following RUN.

HOW MANY ROLLS? 100	
OUTCOME	FREQUENCY
1	15
2	18
3	13
4	16
5	20
6	18
FOR MORE ROLLS, PRESS ANY KEY	

FREQUENCY means
Number of Times

Write your program using the following outline of REM statements.

```

100 REM ** DIE ROLLER AND COUNTER
200 REM ** TALK TO PERSON
300 REM ** START COUNTS AT ZERO
400 REM ** ROLL AND COUNT
500 REM ** SHOW RESULTS
600 REM ** WAIT FOR KEY PRESS, DO AGAIN
  
```

Don't read on until *after* you have written your program. Then read on.

17. We did blocks 100, 200, and 300, as follows:

```

100 REM ** DIE ROLLER AND COUNTER

200 REM ** TALK TO PERSON
210 CLS
220 INPUT "HOW MANY ROLLS": NR

300 REM ** START COUNTS AT ZERO
310 FOR K = 1 TO 6
320   D(K) = 0
330 NEXT K

```



- (a) In your own words, describe what happens in block 300. _____

- (b) So far, is a DIM statement necessary? _____

- (a) Block 300 assigns the value zero (0) to each and every subscripted variable D(1) through D(6).
- (b) No, because no subscript is more than 10. The largest subscript used is 6 in D(6).

18. Here's the big one, block 400!

```

400 REM ** ROLL AND COUNT
410 FOR R = 1 TO NR
420   SPOTS = RND (6)
430   D(SPOTS) = D(SPOTS) + 1
440 NEXT R

```

- (a) Suppose NR is 100. How many times will lines 420 and 430 be executed by the computer? _____
- (b) In line 420, what are the possible values of SPOTS? _____

- (c) Suppose SPOTS is 1. What happens in line 430? _____

- (d) Suppose SPOTS is 3. What happens in line 430? _____

- (e) Suppose SPOTS is 6. What happens in line 430? _____

- (f) What variable keeps track of the number of times the "die" came up 5?

- (a) 100 times, thanks to line 410.
- (b) 1, 2, 3, 4, 5, or 6. This simulates one roll of a six-sided die.
- (c) The value of D(1) is increased by 1.
- (d) The value of D(3) is increased by 1.
- (e) The value of D(6) is increased by 1.
- (f) D(5). Also, D(1) keeps track of the number of times the die came up 1, D(2) keeps track of the number of times it came up 2, and so on up to D(6).

19. Blocks 500 and 600 are easy.

```
500 REM ** SHOW RESULTS
510 PRINT
520 PRINT "OUTCOME", "FREQUENCY"
530 PRINT
540 FOR SPOTS=1 TO 6
550   PRINT @ SPOTS, D(SPOTS)
560 NEXT SPOTS

600 REM ** WAIT FOR KEY PRESS, DO AGAIN
610 PRINT
620 PRINT "FOR MORE ROLLS, PRESS ANY KEY"
630 IF INKEY$="" THEN 630 ELSE 210
```

Try some variations.

—Loaded die. Make 5 more likely to occur.

```
420 SPOTS=RND(7): IF SPOTS=7 THEN SPOTS=5
```

Want to make 5 even more likely? Try this:

```
420 SPOTS=RND(8): IF SPOTS > 6 THEN SPOTS=5
```

RUN the program for each variation above and see whether 5 occurs more frequently. Try this on an unsuspecting friend. He or she will probably say that the computer made a mistake. Ha! Computers don't make mistakes—programmers make mistakes.

—Oh, you don't like 5? Your favorite number of spots is 3? OK, change line 420 so 3 is more likely.

```
420 _____
```

```
420 SPOTS=RND(7): IF SPOTS=7 THEN SPOTS=3
420 SPOTS=RND(8): IF SPOTS>6 THEN SPOTS=3
```

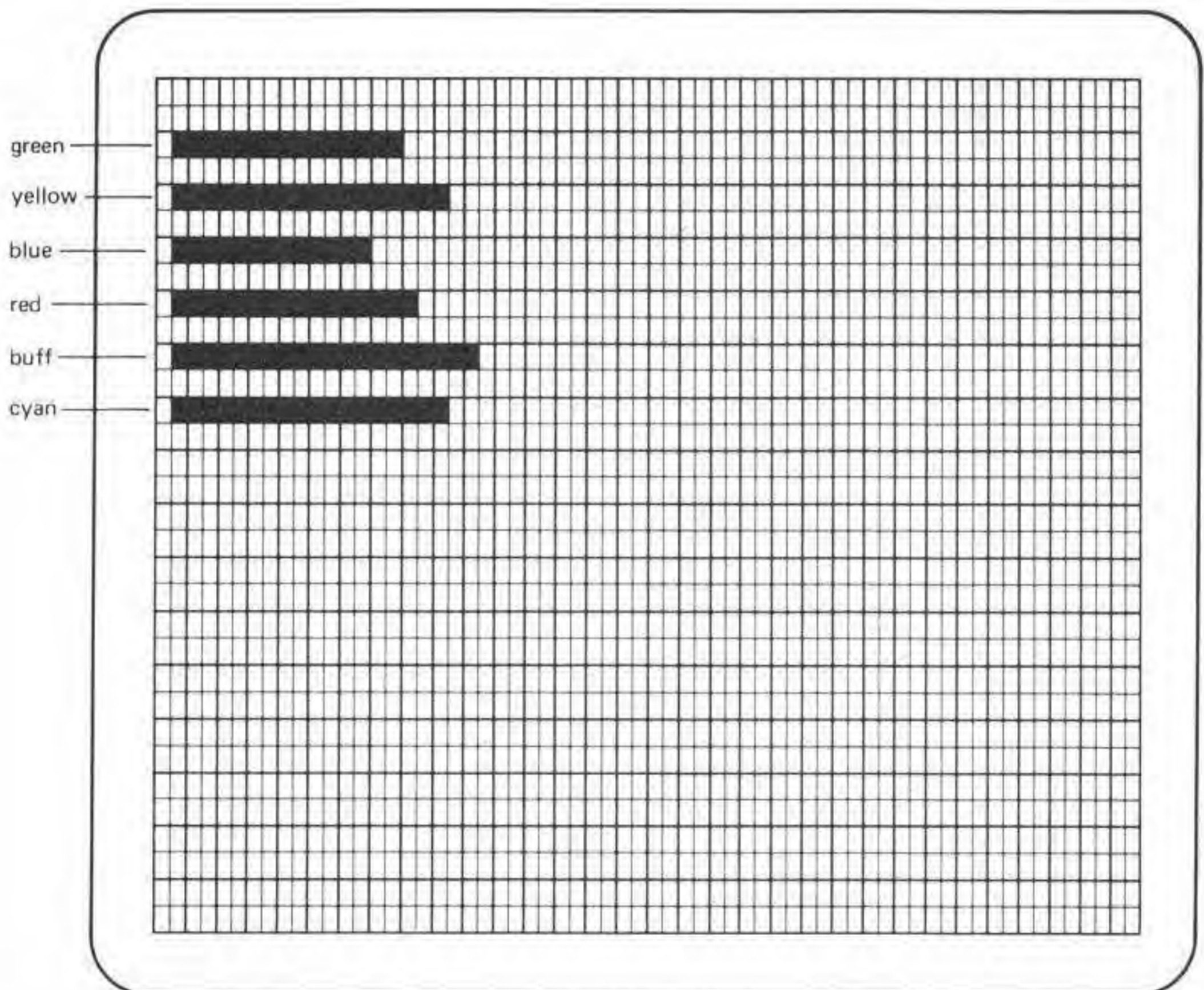
20. Here is another, more colorful way to show the results of NR rolls of a single die.

```

500 REM ** PAINT THE RESULTS
510 CLS 0
520 FOR SPOTS=1 TO 6
530   DOWN = 2*SPOTS
540   FOR OVER=1 TO D(SPOTS)
550     SET(OVER, DOWN, SPOTS)
560   NEXT OVER
560 NEXT SPOTS

```

With the above change, the results shown in frame 16 would appear as follows:



Make the above change to DIE ROLLER AND COUNTER and roll a few dice. Try 100 or 200 rolls. Try 1000 rolls. Oops! The computer probably tried to paint a stripe right off the screen and stopped with an error message.

Then read on and we'll ask you some leading questions about PAINT THE RESULTS.

- (a) When SPOTS is 1, what is the value of DOWN in line 530? _____
 (b) When SPOTS is 1, what happens in lines 540 through 560? _____

 (c) When SPOTS is 1, what color is the stripe painted in lines 540 through 560? _____

We could also ask the above questions for SPOTS=2, SPOTS=3, and so on up to SPOTS=6. We won't. Instead, we ask you to ask yourself these questions and answer them.

- (a) 2
 (b) The computer paints a stripe from OVER=1 to OVER=D(1). Aha! The length of the stripe is the number of times the die came up 1.
 (c) green

When SPOTS is 2, the value of DOWN is 4. The computer draws a stripe from OVER=1 to OVER=D(2). This stripe is yellow. And so on.

21. Next, let's turn the computer into a sound and color organ. You can play up to eight notes in up to eight colors using the number keys 1 through 8. In lines 910 and 920 we have put the tone numbers for the scale of C and color codes for green, yellow, blue, and red. Change them to your choice of tones and colors.

```

100 REM ** SOUND AND COLOR ORGAN
200 REM ** FILL TONE AND COLOR ARRAYS
210 FOR K=1 TO 8
220   READ T(K)
230 NEXT K
240 FOR K=1 TO 8
250   READ C(K)
260 NEXT K

300 REM ** TELL WHAT TO DO
310 CLS
320 PRINT "USE ONLY KEYS 1 TO 8."
330 PRINT "PRESS A KEY (1 TO 8)."

```

Line 420 can also
be written as:
420 K = VAL(K\$)

Enter the program and RUN it. Press any number key, 1 to 8. Each key will cause a different tone to play.

- (a) What colors might appear? _____
 (b) If you press the 1 key, what is the value of K in line 420? _____
-

- (a) green, yellow, blue, or red. (See line 920.)
 (b) 1. Why? K\$ is "1". Therefore, ASC(K\$) is 49 and ASC(K\$) - 48 is 1.

Just to see what happens, press the 9 key or a letter key or the space bar. Remember, this program is designed to work only with the number keys 1 to 8. These keys are used to select tone from the array T() and colors from the array C().

22. Rewrite SOUND AND COLOR ORGAN so you can press letters of the alphabet to make tones and colors happen

Press A to get first tone and color
 Press B to get second tone and color
 Press C to get third tone and color
 .
 .
 .
 Press Z to get 26th tone and color

We made these changes:

```
110 DIM T(26), C(26)
210 FOR K=1 TO 26
240 FOR K=1 TO 26
420 K = ASC(K$) - 64
```



You must also put 26 tone numbers and 26 color numbers in DATA statements. If you wish, your favorite tone numbers or color numbers can appear more than once.

23. Remember the GUESS MY WORD game? If not, review Chapter 10, frames 6 to 8, and Self-Test question 4. Rewrite GUESS MY WORD so the computer first puts the words into the string array WORD\$, then selects words at random from the string array. Here is an outline of the program:

```

100 REM ** GUESS MY WORD
200 REM ** READ AND COUNT WORDS
300 REM ** TELL HOW TO PLAY
400 REM ** PICK RANDOM WORD FROM ARRAY
500 REM ** GET GUESS
600 REM ** IF NOT CORRECT, GIVE CLUE
700 REM ** WINNER!
800 REM ** TELL HOW TO PLAY AGAIN
1000 REM ** LIST OF WORDS & FLAG, ZZZ
1010 DATA ARK, BAH, CAT, DIG, ELF, FLY
1020 DATA FUN, GNU, HAW, HEX, JOY, LEU
1030 DATA MEW, NIX, OAF, PAL, RAM, RED
1040 DATA ROC, SHH, SKY, TOO, UGH, VIA
1050 DATA WAG, WOW, YAK, ZOO, ZZZ

```



Don't peek! Write your program and try it. Then read on.

24. We began like this:

```

100 REM ** GUESS MY WORD
110 CLEAR 1000
120 DIM WORD$(200)

200 REM ** READ AND COUNT WORDS
210 NW = 0: K = 1
220 READ WORD$(K)
230 IF WORD$(K) = "ZZZ" THEN 310
240 NW = NW + 1: K = K + 1
250 GOTO 220

1000 REM ** LIST OF WORDS & FLAG, ZZZ
1010 DATA ARK, BAH, CAT, DIG, ELF, FLY
1020 DATA FUN, GNU, HAW, HEX, JOY, LEU
1030 DATA MEW, NIX, OAF, PAL, RAM, RED
1040 DATA ROC, SHH, SKY, TOO, UGH, VIA
1050 DATA WAG, WOW, YAK, ZOO, ZZZ

```

Line 110 reserves memory space for 1000 string characters. Line 120 reserves memory space for a string array WORD\$ with up to 201 elements. However, WORD\$(0) is not used, so the array can store up to 200 words.

(a) Will 200 three-letter words fit into 1000 characters spaces? _____

Suppose the computer has just finished executing block 200. It has read and stored all the three-letter words in the DATA statements in frame 26.

(b) What is the value of WORD\$(3)? _____

(c) Where is PAL stored? _____

(d) Where is ZZZ stored? _____

(e) What is the value of K? _____

(f) What is the value of NW? _____

(a) Yes. 200 three-letter words can be stored in 600 memory positions plus a few required for "bookkeeping."

(b) CAT

(c) WORD\$(16)

(d) WORD\$(29)

(e) 29

(f) 28 Good! NW is the number of words, *not counting ZZZ*. See frame 12.

25. That was the hard part. Here's the rest:

```

300 REM ** TELL HOW TO PLAY
310 CLS
320 PRINT "I'LL THINK OF A 3-LETTER WORD."
330 PRINT "MY WORD IS BETWEEN AAA AND ZZZ."
340 PRINT
350 PRINT "MY LOWEST 'WORD' IS AAA."
360 PRINT "MY HIGHEST 'WORD' IS ZZZ."

400 REM ** PICK RANDOM WORD FROM ARRAY
410 RW = RND(NW)
420 W$ = WORD$(RW)

500 REM ** GET GUESS
510 PRINT: INPUT "YOUR GUESS": G$

600 REM ** IF NOT CORRECT, GIVE CLUE
610 IF G$ < W$ THEN PRINT "TRY HIGHER WORD": GOTO 510
620 IF G$ > W$ THEN PRINT "TRY LOWER WORD": GOTO 510

700 REM ** WINNER!
710 PRINT "THAT'S IT! YOU GUESSED MY WORD."
720 SOUND 89, 20: SOUND 108, 20: SOUND 125, 20
730 SOUND 176, 40

800 REM ** TELL HOW TO PLAY AGAIN
810 PRINT
820 PRINT "TO PLAY AGAIN, PRESS ANY KEY."
830 IF INKEY$ = "" THEN 830 ELSE 310

```



Enjoy playing. Put your word list in lines 1010 on. Try four-letter words or five-letter words. If you do, remember to change block 300, which tells how to play.

Self-Test

You are nearing the end of the adventure. You have acquired skill, knowledge, and experience points. So plunge into another Self-Test with confidence—nothing can go array.

1. Classify each variable as a numeric variable (NV), string variable (SV), subscripted numeric variable (SNV), subscripted string variable (SSV), or not a legal variable (OOPS).
 - (a) X_____
 - (b) X\$_____
 - (c) \$X_____
 - (d) TONE_____
 - (e) T(K)_____
 - (f) KOLOR(K\$)_____
 - (g) L5\$(3)_____
 - (h) T(ASC(L\$)-64)_____
 2. Write a single DIM statement to reserve space for a numeric array N with 100 elements and a string array SS with 50 elements.
 3. Fill in the missing gaps in the following program. This program is similar to TWINKLING STARS in frames 5–7 but uses blips of color instead of asterisks to represent stars.
-

```

100 REM ** TWINKLING STARS
110 DIM OVER(100), DOWN(100), KOLOR(100)
120 CLS 0

200 REM ** READ STAR DATA INTO ARRAYS
210 FOR STAR=1 TO 7
220   READ OVER(STAR), DOWN(STAR), KOLOR(STAR)
230 NEXT STAR

300 REM ** TURN ON THE STARS
310 _____
320 _____
330 _____

400 REM ** RANDOM STAR
410 RS = RND(7)

500 REM ** TWINKLE IT OFF
510 RESET(OVER(RS), DOWN(RS)) ← Turns star to black.
520 GOSUB 810

600 REM ** TWINKLE IT ON
610 _____
620 _____

700 REM ** DO IT AGAIN
710 GOTO 410

800 REM ** TIME DELAY SUBROUTINE
810 Z = 200
820 FOR K=1 TO Z: NEXT K
830 RETURN

900 REM ** STAR DATA: OVER, DOWN, COLOR
910 DATA 6, 12, 1
920 DATA 18, 10, 2
930 DATA 26, 12, 3
940 DATA 34, 14, 4
950 DATA 38, 20, 6
960 DATA 54, 20, 7
970 DATA 56, 14, 8

```

Seven stars.

4. Rewrite your program for question 3 so the computer reads the star data into arrays OVER, DOWN, KOLOR and also counts the number of stars. Use -1,-1,-1 as the end of data flags. Putting two stars per DATA statement, the star data looks like this:

```

900 REM ** STAR DATA: OVER, DOWN, KOLOR
910 DATA 6, 12, 1, 18, 10, 2
920 DATA 26, 12, 3, 34, 14, 4
930 DATA 38, 20, 6, 54, 20, 7
940 DATA 56, 14, 8, -1, -1, -1

```

These are end of data flags.

- Rewrite your program for question 4 so the star data is entered from the keyboard in response to an INPUT statement.

```

OVER, DOWN, COLOR: 6, 12, 1
OVER, DOWN, COLOR: 18, 10, 2
.
.
.
OVER DOWN, COLOR? 56, 14, 8
OVER, DOWN, COLOR? -1, -1, -1
    
```

- Flip two coins. Let Heads equal 0 and Tails equal 1. The outcome of a flip is the sum of these two numbers. Here are the possible outcomes.

FIRST COIN	SECOND COIN	OUTCOME
H	H	0
H	T	1
T	H	1
T	T	2

The possible outcomes are 0, 1, or 2.

Write a program similar to DIE ROLLER AND COUNTER (frames 18–21) to flip two coins and count the outcomes. A RUN might look like this:

```

HOW MANY FLIPS? 100

OUTCOME      FREQUENCY
0             29
1             47
2             24

FOR MORE ROLLS, PRESS ANY KEY
    
```

Looks like these outcomes are not equally likely.

- Modify your program of question 6 so the computer rolls two dice NR times and counts the possible outcomes.
Possible outcomes: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
- How about three dice? Aha! We use three dice to roll characteristics for fantasy adventure characters. The results of running your program for a large number of rolls (try 1000) should tell you something about the probability, or likelihood, of rolling the various values from 3 to 18.
- In your programs for questions 6 to 8, paint the results as in frame 20.
- Using arrays, again try *Scrabble Scores*, question 9 in the Chapter 10 Self-Test.

11. Make music from strings of letters. Assign number scores to the letters A through Z as follows:

A = 1	G = 7	M = 3	S = 19	Y = 25
B = 2	H = 8	N = 14	T = 20	Z = 26
C = 3	I = 9	O = 15	U = 21	
D = 4	J = 10	P = 16	V = 22	
E = 5	K = 11	Q = 17	W = 23	
F = 6	L = 12	R = 18	X = 24	

Write a program to:

- assign 26 tone numbers to T(1), T(2), T(3), . . . , T(26);
- ask for a word or phrase; when someone enters a word, phrase, or any string, put it in WORD\$;
- for each letter of WORD\$, play the tone that corresponds to the letter; for A, play T(1); for B, play T(2); for C, play T(3); and so on—for Z, play T(26);
- ignore spaces; in fact, ignore everything except letters;
- after playing a tone for each letter, go back and ask for another word or phrase; or, play the same stuff again and again and again until people yell, "Turn that BLEEP thing off!"

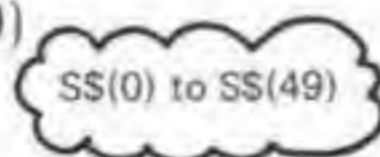
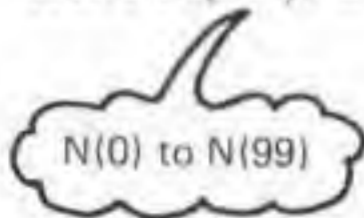
Is your name musical? Is MATHEMATICS melodious? Can you contrive some tuneful talk?

Answers to Self-Test

The numbers in parentheses after each answer refer to the frames in the chapter where the topic is discussed.

- X NV
 - X\$ SV
 - SX OOPS
 - TONE OOPS Begins with a reserved word (TO).
 - T(K) NV
 - KOLOR (K\$) OOPS Subscript must be numeric.
 - L5\$(3) SSV
 - T(ASC(L\$)-64) SSV (frames 1, 2)

- DIM N(99), SS(49)



(frame 11)

- ```
310 FOR STAR=1 TO 7
320 SET(OVER(STAR), DOWN(STAR), KOLOR(STAR))
330 NEXT STAR
```

```
610 SET(OVER(RS), DOWN(RS), KOLOR(RS))
```

```
620 GOSUB 810
```

(frames 5-7)

4. We made these changes to the program in question 3:

```

200 REM ** READ STAR DATA INTO ARRAYS
210 STAR = 0
220 STAR = STAR + 1
230 READ OVER(STAR, DOWN(STAR), KOLOR(STAR))
240 IF OVER(STAR) <> -1 THEN 220
250 NS = STAR - 1
310 FOR STAR = 1 TO NS
410 RS = RND(NS)

```

Remember: Use the  
DATA statements  
in question 4!

(frame 13)

5. Change only block 200.

```

200 REM ** ENTER STAR DATA INTO ARRAYS
210 STAR = 0
220 STAR = STAR + 1
230 INPUT "OVER, DOWN, KOLOR": OVER(STAR), DOWN(STAR),
 KOLOR(STAR)
240 IF OVER(STAR) <> -1 THEN 220
250 NS = STAR - 1

```

(frames 10, 14)

6. 100 REM \*\* COIN FLIPPER & COUNTER (2 COINS)

```

200 REM ** TALK TO PERSON
210 CLS
220 INPUT "HOW MANY FLIPS": NF
300 REM ** START FROM ZERO
310 N(0) = 0
320 N(1) = 0
330 N(2) = 0
400 REM ** FLIP COINS & COUNT OUTCOMES
410 FOR FLIP=1 TO NF
420 C1 = RND(2) - 1
430 C2 = RND(2) - 1
440 RESULT = C1 + C2
450 N(RESULT) = N(RESULT) + 1
460 NEXT FLIP
500 REM ** SHOW TOTALS
510 PRINT
520 PRINT "OUTCOME", "FREQUENCY"
530 PRINT
540 PRINT 0, N(0)
550 PRINT 1, N(1)
560 PRINT 2, N(2)
600 REM ** WAIT FOR KEY PRESS, DO AGAIN
610 PRINT
620 PRINT "FOR MORE FLIPS, PRESS ANY KEY"
630 IF INKEY$="" THEN 630 ELSE 210

```

Variation: Make one or both coins "unfair." (frames 15-19)

7, 8, 9, 10. No answers. Use your new skills to write your answers.

---

---

## CHAPTER TWELVE

# Computing Problems and Challenges

---

---

In the first 11 chapters, you learned how to use the following words in Color BASIC.

|       |         |         |        |         |      |
|-------|---------|---------|--------|---------|------|
| ABS   | ASC     | CHRS    | CLEAR  | CLS     | DATA |
| DIM   | END     | ELSE    | FOR    | GOSUB   | GOTO |
| IF    | INKEY\$ | INPUT   | LEFT\$ | LEN     | LIST |
| MID\$ | NEW     | NEXT    | PRINT  | PRINT@  | READ |
| REM   | RESET   | RESTORE | RETURN | RIGHT\$ | RND  |
| RUN   | SET     | SOUND   | STEP   | STOP    | THEN |
| TO    | VAL     |         |        |         |      |

Already, 38 tools in your BASIC Toolbox! In this chapter, we will pose problems, then solve them in several ways, using the above BASIC tools. You will also learn to use additional BASIC features, including the following:

INT ON POKE SGN STR\$

This chapter is not divided into frames, as were chapters 1 through 11. Instead, each problem begins with a heading and description, followed by one or more solutions. You will see that there are usually *many* different ways to solve a problem! Write your solution before you look at ours.

For more computing problems, puzzles, and challenges, try these:

“Computing Problems” in every issue of *Dymax Gazette*, P.O. Box 310, Menlo Park, CA 94025.

“Computing Problems” in every issue of *The Computing Teacher*, Dept. of Computer and Information Science, University of Oregon, Eugene, OR 97403.

“My Computer Likes Me.” in every issue of *Popular Computing*, P.O. Box 397, Hancock, NH 03449.



## POSITIVE, NEGATIVE, OR ZERO

An easy problem. Write a program to ask for a number, then tell whether the number is positive, negative, or zero. When you RUN the program, it might go like this:

```
ENTER A NUMBER AND I WILL TELL
YOU WHETHER YOUR NUMBER IS
POSITIVE, NEGATIVE, OR ZERO.

YOUR NUMBER? -7
YOUR NUMBER IS NEGATIVE

YOUR NUMBER? 3
YOUR NUMBER IS POSITIVE

YOUR NUMBER? 0
YOUR NUMBER IS ZERO

YOUR NUMBER? ■ and so on.
```

Solution #1 should be familiar. You first saw it in Chapter 9, frame 2. We made some very slight changes.

```
100 REM ** POSITIVE, NEGATIVE OR ZERO #1
200 REM ** TELL PERSON WHAT TO DO
210 CLS
220 PRINT "ENTER A NUMBER AND I WILL TELL"
230 PRINT "YOU WHETHER YOUR NUMBER IS"
240 PRINT "POSITIVE, NEGATIVE, OR ZERO."

300 REM ** ASK FOR A NUMBER
310 PRINT
320 INPUT "YOUR NUMBER": X

400 REM ** TELL PERSON ABOUT THE NUMBER
410 IF X>0 THEN PRINT "YOUR NUMBER IS POSITIVE"
420 IF X<0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
430 IF X=0 THEN PRINT "YOUR NUMBER IS ZERO"

500 REM ** LOOP BACK FOR NEW NUMBER
510 GOTO 310
```

Solution #2. Make the following changes to Solution #1.

---

```

100 REM ** POSITIVE, NEGATIVE, OR ZERO #2
110 N$ = "YOUR NUMBER IS NEGATIVE"
120 Z$ = "YOUR NUMBER IS ZERO"
130 P$ = "YOUR NUMBER IS POSITIVE"

400 REM ** TELL PERSON ABOUT THE NUMBER
410 IF X>0 THEN PRINT P$
420 IF X<0 THEN PRINT N$
430 IF X=0 THEN PRINT Z$

```

Remember, these are changes. You will also need blocks 200, 300, and 500 from Solution #1.

Solution #3 uses two new BASIC features called ON and SGN. Look for them in lines 410 and 420.

```

100 REM ** POSITIVE, NEGATIVE, OR ZERO #3
110 N$ = "YOUR NUMBER IS NEGATIVE"
120 Z$ = "YOUR NUMBER IS ZERO"
130 P$ = "YOUR NUMBER IS POSITIVE"

200 REM ** TELL PERSON WHAT TO DO
210 CLS
220 PRINT "ENTER A NUMBER AND I WILL TELL"
230 PRINT "YOU WHETHER YOUR NUMBER IS"
240 PRINT "POSITIVE, NEGATIVE, OR ZERO."

300 REM ** ASK FOR A NUMBER
310 PRINT
320 INPUT "YOUR NUMBER"; X

400 REM ** TELL PERSON ABOUT THE NUMBER
410 W = SGN(X) + 2
420 ON W GOTO 430, 440, 450
430 PRINT N$: GOTO 310
440 PRINT Z$: GOTO 310
450 PRINT P$: GOTO 310

```

Block 500 is not needed  
in this solution.

OK! OK! We'll explain lines 410 and 420. The SGN function has three possible values: -1, 0, or 1.

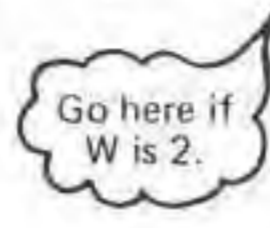
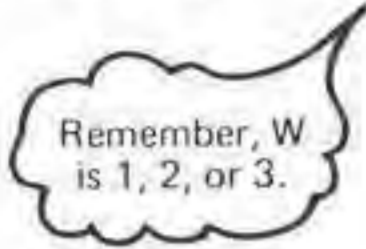
If X is negative, then SGN(X) is -1.  
If X is zero, then SGN(X) is 0.  
If X is positive, then SGN(X) is 1.

In line 410, the value of W is  $\text{SGN}(X) + 2$ .

If X is negative, then W is 1.  
If X is zero, then W is 2.  
If X is positive, then W is 3.

Line 420 sends the computer to line 430, or line 440, or line 450, depending on the value of W.

## ON W GOTO 430, 440, 450



If W is 1, the computer goes to line 430, prints the value of N\$, then goes to line 310. If W is 2, the computer goes to line 440, prints the value of Z\$, then goes to line 310. And so on.

Lines 410 and 420 can be combined into a single statement, as follows:

```
410 ON SGN(X) + 2 GOTO 430, 440, 450
```

If you try this, remember to delete the old line 420.

Solution #4 uses ON . . . GOSUB . . . instead of ON . . . GOTO . . . Here are the changes to make in Solution #3:

```
100 REM ** POSITIVE, NEGATIVE, OR ZERO #4
410 ON SGN(X)+2 GOSUB 430, 440, 450
420 GOTO 310
430 PRINT N$: RETURN
440 PRINT Z$: RETURN
450 PRINT P$: RETURN
```

It works like this:

If X is negative, then  $\text{SGN}(X)+2$  is 1. The computer GOSUBs to line 430, PRINTs the value of N\$, and RETURNs to line 420.

If X is zero, then  $\text{SGN}(X)+2$  is 2. The computer GOSUBs to line 440, PRINTs the value of Z\$, and RETURNs to line 420.

If X is positive, then  $\text{SGN}(X)+2$  is 3. The computer GOSUBs to line 450, PRINTs the value of P\$, and RETURNs to line 420.

Four solutions already. Can there be more? Of course . . . here comes Solution #5.



```

100 REM ** POSITIVE, NEGATIVE, OR ZERO #5
110 NZP$(0) = "YOUR NUMBER IS NEGATIVE"
120 NZP$(1) = "YOUR NUMBER IS ZERO"
130 NZP$(2) = "YOUR NUMBER IS POSITIVE"

200 REM ** TELL PERSON WHAT TO DO
210 CLS
220 PRINT "ENTER A NUMBER AND I WILL TELL"
230 PRINT "YOU WHETHER YOUR NUMBER IS"
240 PRINT "POSITIVE, NEGATIVE, OR ZERO."

300 REM ** ASK FOR A NUMBER
310 PRINT
320 INPUT "YOUR NUMBER": X

400 REM ** TELL PERSON ABOUT THE NUMBER
410 W = SGN(X) + 1
420 PRINT NZP$(W)

500 REM ** LOOP BACK FOR NEW NUMBER
510 GOTO 310

```

In line 410, W can be 0, 1, or 2. Thus, in line 420, the computer prints the appropriate message from the string array NZP\$.

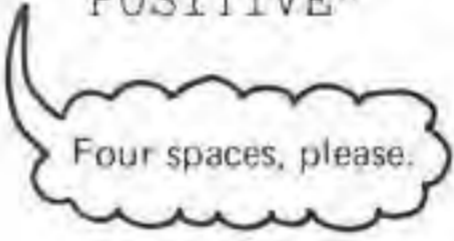
We like this solution!

Solution #6 packs information into a single string, then picks off the appropriate substring. Make these changes to Solution #5.

```

100 REM ** POSITIVE, NEGATIVE, OR ZERO #6
110 NZP$ = "NEGATIVEZERO POSITIVE"

```



```

410 W = SGN(X) + 1
420 PRINT "YOUR NUMBER IS" MID$(NZP$, 8*W+1, 8)


```

Or you can use READ and DATA to get information into the array NZP\$. Make these changes to Solution #5. Call this Solution #7:

```

100 REM ** POSITIVE, NEGATIVE, OR ZERO #7
110 FOR K=0 TO Z
120 READ NZP$(K)
130 NEXT K
140 DATA YOUR NUMBER IS NEGATIVE
150 DATA YOUR NUMBER IS ZERO
160 DATA YOUR NUMBER IS POSITIVE

```



Remember: Modify #5  
to get this solution.

Just one more solution, Solution #8. We threw this one in primarily as a review of READ, DATA and RESTORE.

```
100 REM ** POSITIVE, NEGATIVE, OR ZERO #8
```

```
200 REM ** TELL PERSON WHAT TO DO
```

Same as Solution #1.

```
300 REM ** ASK FOR A NUMBER
```

Same as Solution #1.

```
400 REM ** TELL PERSON ABOUT THE NUMBER
```

```
410 W = SGN(X) + 2
```

```
420 RESTORE
```

```
430 FOR K=1 TO W
```

```
440 READ NZP$
```

```
450 NEXT K
```

```
460 DATA NEGATIVE, ZERO, POSITIVE
```

```
470 PRINT "YOUR NUMBER IS" NZP$
```

```
500 REM ** LOOP BACK FOR NEW NUMBER
```

```
510 GOTO 310
```

Eight different ways to solve one simple problem. Are there yet more ways? Of course—*your* way, for one!

## TWO-DIGIT NUMBER SPLITTER

In this problem, a two-digit number is an integer\* from 10 to 99, inclusive. Write a program to ask for a two-digit number. When someone enters a two-digit number, split it into its two digits and print them separately.

Here is a sample RUN:

---

\*In case you have forgotten, an integer is a number that does not have a fractional part.

These are integers: 0, 1, 2, 3, and so on.

These are integers: -1, -2, -3, and so on.

These are *not* integers: 1.4, 2.3, 4.95, -.82, etc.

TWO-DIGIT NUMBER, PLEASE? 37  
 TENS DIGIT = 3 ONES DIGIT = 7

TWO-DIGIT NUMBER, PLEASE? 123  
 MUST BE AN INTEGER, 10 TO 99.

TWO-DIGIT NUMBER, PLEASE? 07  
 MUST BE AN INTEGER, 10 TO 99.

TWO-DIGIT NUMBER, PLEASE? 4.5  
 MUST BE AN INTEGER, 10 TO 99.

TWO-DIGIT NUMBER, PLEASE? -25  
 MUST BE AN INTEGER, 10 TO 99.

TWO-DIGIT NUMBER, PLEASE? 99  
 TENS DIGIT = 9 ONES DIGIT = 9

TWO-DIGIT NUMBER, PLEASE? 10  
 TENS DIGIT = 1 ONES DIGIT = 0

and so on.



Write *your* program, then look at ours.

Solution #1 uses arithmetic to compute the tens digit and the ones digit. It is similar to using long division, as follows:

$$\begin{array}{r}
 3 \leftarrow \text{Tens digit.} \\
 10 \overline{) 37} \\
 \underline{30} \\
 7 \leftarrow \text{Ones digit.}
 \end{array}$$

$$\begin{array}{r}
 1 \leftarrow \text{Tens digit.} \\
 10 \overline{) 10} \\
 \underline{10} \\
 0 \leftarrow \text{Ones digit.}
 \end{array}$$

First, divide the two-digit number by 10. The whole number quotient is the tens digit. Then multiply the tens digit by 10 and subtract it from the two-digit number. The result is the ones digit.



In the following program, we use T for the tens digit and U for the ones digit. We don't use O (Oh) because it looks too much like 0 (zero).

```
100 REM ** TWO-DIGIT NUMBER SPLITTER #1
110 OOPS$ = "MUST BE AN INTEGER, 10 TO 99."
200 REM ** ASK FOR A TWO-DIGIT NUMBER, X
210 CLS
220 PRINT
230 INPUT "TWO-DIGIT NUMBER, PLEASE": X
300 REM ** IF X IS OK, GO ON ELSE GO BACK
310 IF X<>INT(X) PRINT OOPS$: GOTO 220
320 IF X<10 PRINT OOPS$: GOTO 220
330 IF X>99 PRINT OOPS$: GOTO 220
400 REM ** SPLIT X INTO T (TENS) & U (ONES)
410 T = INT(X/10)
420 U = X - 10*T
500 REM ** PRINT RESULTS
510 PRINT "TENS DIGIT ="T, "ONES DIGIT ="U
600 REM ** GO BACK FOR MORE
610 GOTO 220
```

Something new: the INT function in lines 310 and 410. If X is an integer, then INT(X) is equal to X.

|               |               |
|---------------|---------------|
| INT(0) is 0   | INT(1) is 1   |
| INT(24) is 24 | INT(-3) is -3 |

If X is not an integer, then INT(X) is the greatest integer that is less than X.

|                |                  |
|----------------|------------------|
| INT(3.14) is 3 | INT(-3.14) is -4 |
| INT(.01) is 0  | INT(-.01) is -1  |

For any number X, INT(X) is the greatest integer that is less than or equal to X.

In line 310, if X is not an integer, the computer PRINTs the value of OOPS\$ and goes to line 220. If X is an integer, then the computer goes on to line 320.

Line 410 tells the computer to divide X by 10, then compute the integer part of the result and assign it as the value of T. Suppose X is 37.

$$T = \text{INT}(37/10) = \text{INT}(3.7) = 3$$

---

Aha! This is the tens digit of 37. Line 420 tells how to compute the ones digit.

$$U = X - 10 * T = 37 - 10 * 3 = 37 - 30 = 7$$

$$\begin{array}{r}
 3 \leftarrow T = \text{INT}(T/10) \\
 10 \overline{) 37} \leftarrow X \\
 \underline{30} \leftarrow 10 * T \\
 7 \leftarrow U = X - 10 * T
 \end{array}$$

It seems that a simple thing for a person is difficult for a computer! After all, anyone can look at 37 and tell you immediately that the tens digit is 3 and the ones digit is 7. The 3 is to the *left* of the 7, which, of course, is to the *right* of the 3.

Look at 88. The left 8 is the tens digit; the 8 on the right is the ones digit.

Let's teach the Color Computer to "think" that way. Here are changes for Solution #2.

```

100 REM ** TWO-DIGIT NUMBER SPLITTER #2
400 REM ** SPLIT X INTO T$ (TENS) & U$ (ONES)
410 X$ = STR$(X)
420 T$ = MID$(X$, 2, 1)
430 U$ = MID$(X$, 3, 1)

500 REM ** PRINT RESULTS
510 PRINT "TENS DIGIT = " T$, "ONES DIGIT = " U$

```

Recall that X is a two-digit number. It is positive and has a tens digit (1 to 9) and a ones digit (0 to 9).

The function STR\$(X) creates a three-character string consisting of a space (because X is positive), the tens digit of X, and the ones digit of X. For example, suppose X is 37.

STR\$(X) = STR\$(37) = " 37"

The space occurs because X is a positive number. For a negative number, the first character in the string would be a minus sign (-).

Line 420 picks out the second character of X\$ (tens digit) and assigns it as the value of T\$. Line 430 picks out the third character of X\$ (ones digit) and assigns it as the value of U\$. T\$ and U\$ are each one-character strings.

LEN(X\$) is 3      LEN(T\$) is 1      LEN(U\$) is 1

Solution #3 shows another way to split X\$ into T\$ and U\$. Here are the changes:

```
400 REM ** SPLIT X INTO T$ (TENS) & U$ (ONES)
410 X$ = STR$(X)
420 T$ = LEFT$(X$, 2)
430 U$ = RIGHT$(X$, 1)
```

In line 420, T\$ will be a string of length 2 consisting of a space and the tens digit. You may wish to adjust line 510 slightly to remove a space, or change block 400 to one of the following.

(1) Change only line 420, as follows:

```
420 T$ = RIGHT$(LEFT$(X$, 2), 1)
```

(2) Change lines 410 and 420, as follows:

```
410 X$ = RIGHT$(STR$(X), 2)
420 T$ = LEFT$(X$, 1)
```

How about one more solution? Here's #4.

```
100 REM ** TWO-DIGIT NUMBER SPLITTER #4
400 REM ** SPLIT X INTO T (TENS) & U (ONES)
410 X$ = STR$(X)
420 T$ = LEFT$(X$, 2): T = VAL(T$)
430 U$ = RIGHT$(X$, 1): U = VAL(U$)
500 REM ** PRINT RESULTS
510 PRINT "TENS DIGIT =" T, "ONES DIGIT=" U
```

Blocks 200, 300, and 600 are the same as in Solution #1.

## STRING SQUEEZERS

Write a subroutine to squeeze the blanks (spaces) out of a string.

Before Squeezing

After Squeezing

```
THE FORCE IS WITH YOU
3 D 6
NEVER ODD OR EVEN
SOUTH DAKOTA
```

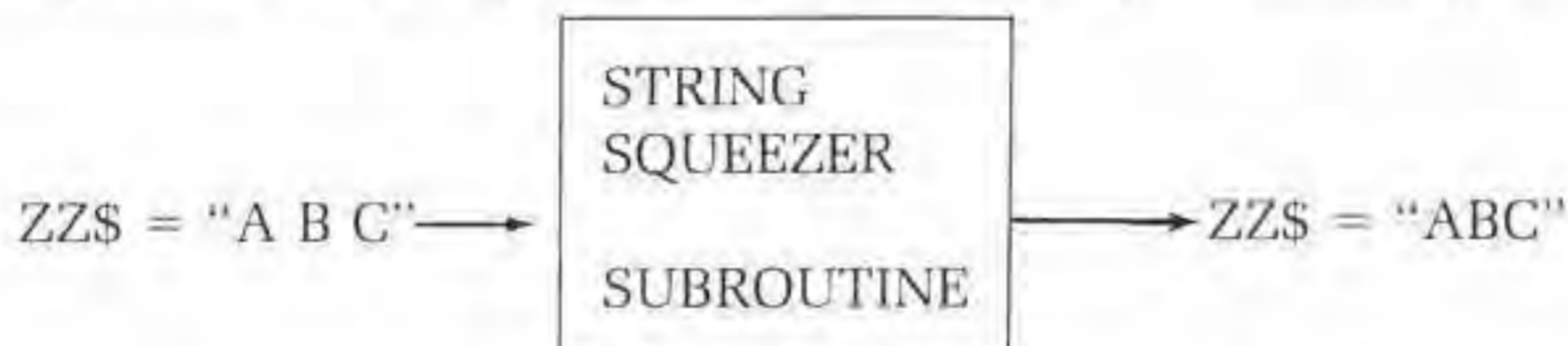
```
THEFORCEISWITHYOU
3D6
NEVERODDOREVEN
SOUTHDAKOTA
```

This might be useful  
in Gamemaster's Dice,  
at the end of this chapter.





Think of your subroutine as a “black box” that accepts a string ZZ\$, squeezes out the spaces, and returns the squeezed string ZZ\$.



Complete the following program by writing the STRING SQUEEZER SUBROUTINE.

```

100 REM ** STRING SQUEEZER #1
200 REM ** ASK FOR STRING
210 CLS
220 PRINT: INPUT "STRING": ZZ$

300 REM ** SQUEEZE OUT SPACES
310 GOSUB 10010

400 REM ** PRINT SQUEEZED STRING
410 PRINT ZZ$

500 REM ** DO IT AGAIN
510 GOTO 220

10000 REM ** STRING SQUEEZER SUBROUTINE

```

You write this.

A RUN might go like this:

```

STRING? A B C
ABC

STRING? THE FORCE IS WITH YOU
THEFORCEISWITHYOU

STRING? . . . and so on.

```

Try this solution:

```

10000 REM ** STRING SQUEEZER SUBROUTINE
10010 SS$ = ""
10020 FOR KK=1 TO LEN(ZZ$)
10030 CC$ = MID$(ZZ$, KK, 1)
10040 IF CC$ <> CHR$(32) THEN SS$ = SS$ + CC$
10050 NEXT KK
10060 ZZ$ = SS$: SS$ = ""
10070 RETURN

```

The subroutine begins (line 10010) by setting up an empty string, SS\$. The FOR-NEXT loop (lines 10020 through 10050) scans ZZ\$, character by character. If a character (CC\$) is not a space, it is added (joined) to SS\$.

When the computer finishes the FOR-NEXT loop, S\$\$ is the desired squeezed string. So, in line 10060, ZZ\$ is set equal to the squeezed string. Then S\$\$ is again "emptied." Why? Well, simply to save memory space. Empty strings don't take much room!

To see S\$\$ build up a character at a time, temporarily add this line:

```
10045 PRINT S$: SOUND 89, 10
```

Your turn again. Write a subroutine to squeeze out everything from a string, except letters A to Z.

| BEFORE SQUEEZING | AFTER SQUEEZING |
|------------------|-----------------|
| ISN'T            | ISNT            |
| FLIP-FLOP        | FLIPFLOP        |
| A, B, C          | ABC             |

Here's most of the program. All you have to do is complete the subroutine beginning at line 10100.

```
100 REM ** STRING SQUEEZER #2
200 REM ** ASK FOR STRING
210 CLS
220 PRINT: INPUT "STRING": ZZ$
300 REM ** SQUEEZE OUT STUFF
310 GOSUB 10110
400 REM ** PRINT SQUEEZED STRING
410 PRINT ZZ$
500 REM ** DO IT AGAIN
510 GOTO 220
10100 REM ** STRING SQUEEZER SUBROUTINE
```

Finished already? OK, now you may look at our solution.

```
10100 REM ** STRING SQUEEZER SUBROUTINE
10110 LO = 65
10120 HI = 90
10130 S$$ = ""
10140 FOR KK=1 TO LEN(ZZ$)
10150 CC$ = MID$(ZZ$, KK, 1)
10160 CC = ASC(CC$)
10170 IF CC < LO OR CC > HI THEN 10190
10180 S$$ = S$$ + CC$
10190 NEXT KK
10200 ZZ$ = S$: S$$ = ""
10210 RETURN
```

We could have combined lines 10170 and 10180 into a single line, as follows:

```
10170 IF CC>=LO AND CC<=HI THEN SS$ = SS$ + CC$
```

If you do this, remember to delete line 10180.

How would you change the subroutine to squeeze out everything except the digits 0 to 9?

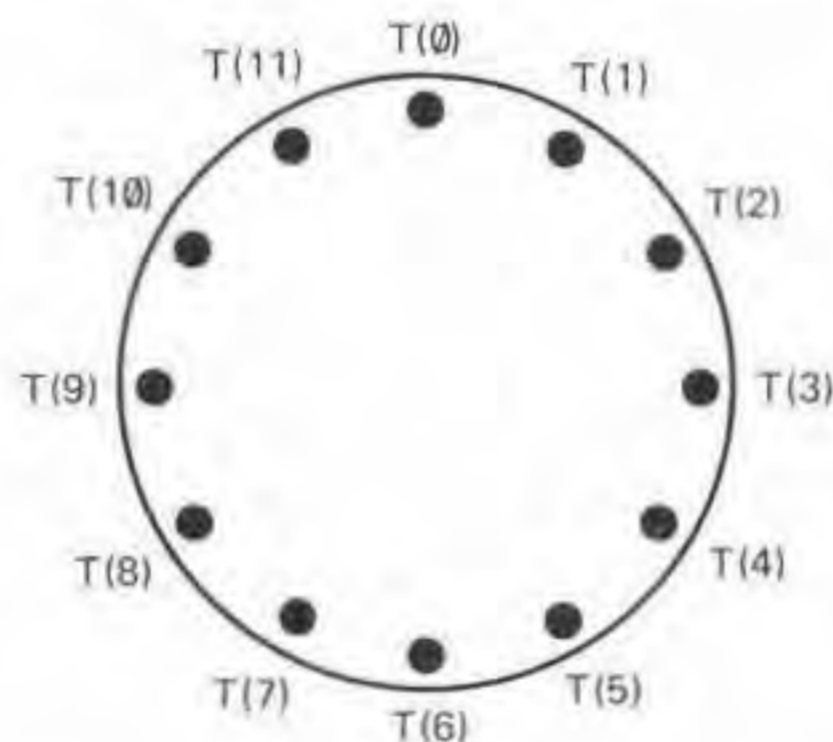
-----

Make these changes:

```
10110 LO = 48
10120 HI = 57
```

## CLOCK MUSIC

Imagine an array of tone numbers arranged in a circle like a clockface.



Play some monotonous random music. First, pick a tone at random and play it. Then flip a coin. If it comes up heads, go clockwise one tone number and play it. If tails, go one tone counterclockwise and play it.

Beware! One tone clockwise from T(11) is T(0), not T(12). One tone counterclockwise from T(0) is T(11), not T(-1).

In the following program, you write block 400 so the computer plays music as described above.



```

100 REM ** CLOCK MUSIC #1
110 DIM T(11)
120 CLS

200 REM ** READ TONE NUMBERS
210 FOR K=0 TO 11
220 READ T(K)
230 NEXT K
240 DATA 89, 99, 108, 117, 125, 133
250 DATA 140, 147, 153, 159, 165, 170

300 REM ** CHOOSE & PLAY FIRST TONE, NT
310 NT = RND(12) - 1
320 SOUND T(NT), 1

400 REM ** CHOOSE NEXT TONE, NT

```

You write this.

```

500 REM ** PLAY NEXT TONE
510 D = RND(3): IF D=3 THEN D=4
520 SOUND T(NT), D

600 REM ** GO CHOOSE AGAIN
610 GOTO 410

```

First, we did it like this:

```

400 REM ** CHOOSE NEXT TONE #1
410 C = RND(2)
420 IF C=1 THEN NT = NT + 1
430 IF C=2 THEN NT = NT - 1
440 IF NT = 12 THEN NT = 0
450 IF NT = -1 THEN NT = 11

```

Remember:  
NT must be an  
integer, 0 to 11.

Then we did it this way.

```

400 REM ** CHOOSE NEXT TONE #2
410 C = 2*(RND(2) - 1) - 1
420 NT = NT + C
430 IF NT = 12 THEN NT = 0
440 IF NT = -1 THEN NT = 11

```

Are you a bit perplexed? In line 410, C will be -1 or 1. Why? Follow along.

|                    |             |
|--------------------|-------------|
| RND(2)             | is 1 or 2.  |
| RND(2) - 1         | is 0 or 1.  |
| 2*(RND(2) - 1)     | is 0 or 2.  |
| 2*(RND(2) - 1) - 1 | is -1 or 1. |

Therefore, in line 420, NT will be increased by 1 (if C is 1) or decreased by 1 (if C is -1). The resulting value of NT will be an integer from -1 to 12, inclusive. But -1 and 12 are not allowed. Lines 430 and 440 take care of this problem.

Here is still another way:

```
400 REM ** CHOOSE NEXT TONE #3
410 C = 2*(RND(2) - 1) - 1
420 NT = NT + C
430 NT = NT + 12
440 NT = NT = 12*INT(NT/12)
```

Oh oh, did we lose anyone? Following line 420, NT will be an integer from -1 to 12. Here is what happens in line 430:

| <u>IF NT WAS</u> | <u>IT BECOMES</u> |     |
|------------------|-------------------|-----|
| -1               | 11                | OK! |
| 0                | 12                |     |
| 1                | 13                |     |
| 2                | 14                |     |
| .                | .                 |     |
| .                | .                 |     |
| .                | .                 |     |
| 12               | 24                |     |

Line 440 computes the remainder on dividing NT by 12.

| <u>NT (Line 420)</u> | <u>NT (Line 430)</u> | <u>NT (Line 440)</u> |
|----------------------|----------------------|----------------------|
| -1                   | 11                   | 11                   |
| 0                    | 12                   | 0                    |
| 1                    | 13                   | 1                    |
| 2                    | 14                   | 2                    |
| 3                    | 15                   | 3                    |
| 4                    | 16                   | 4                    |
| 5                    | 17                   | 5                    |
| 6                    | 18                   | 6                    |
| 7                    | 19                   | 7                    |
| 8                    | 20                   | 8                    |
| 9                    | 21                   | 9                    |
| 10                   | 22                   | 10                   |
| 11                   | 23                   | 11                   |
| 12                   | 24                   | 0                    |

Remember this method. You will see it again as we try to make more interesting clock music.

Flip a coin and roll a die. If the coin shows heads, go clockwise the number of places shown on the die. If the coin shows tails, go counterclockwise the number of places shown on the die.

For example, suppose the computer has just played T(5). Flip the coin and roll the die. The coin shows heads and the die shows 4. Play T(9) next. Flip the coin and roll the die. You get tails and 6. Play T(3). Flip and roll, getting tails and 5. Play T(-2). Oops! That should be T(10). So, play T(10). Flip and roll, getting heads and 4. Play T(14). Oops, again—that should be T(2). Keep on playing!

We will number solutions from where we left off before. So here is Solution #4:

```
400 REM ** CHOOSE NEXT TONE #4
410 C = RND(2)
420 D = RND(6)
430 IF C=1 THEN NT = NT + D
440 IF C=2 THEN NT = NT - D
450 NT = NT + 12
460 NT = NT - 12*INT(NT/12)
```

You could, of course, combine lines 430 and 440 into the following single line.

```
430 IF C=1 THEN NT = NT + D ELSE NT = NT -D
```

Another solution? Coming right up!

```
400 REM ** CHOOSE NEXT TONE #5
410 C = 2*RND(2) - 1) - 1
420 D = RND(6)
430 NT = NT + C*D
440 NT = NT + 12
450 NT = NT - 12*INT(NT/12)
```

Yes, you can combine lines 430 and 440.

```
430 NT = NT + C*D + 12
```

Variations: Roll two six-sided dice and subtract 1.

```
420 D = RND(6) + RND(6) - 1
```

Or roll two four-sided dice (tetrahedrons)

```
420 D = RND(4) + RND(4) - 1
```

Each way of selecting the next note gives a different kind of random music. Try these variations to choose the next tone:

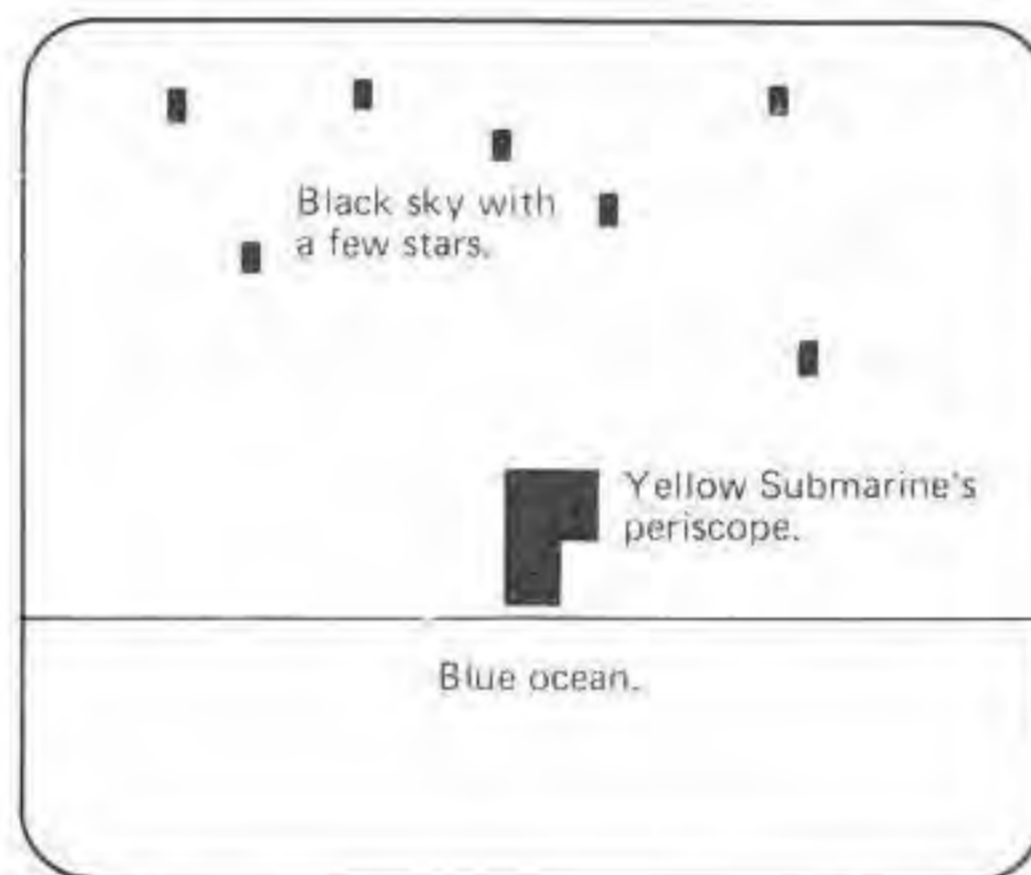
---



Go clockwise or counterclockwise 0, 1, 2, or 4 places.  
 Go clockwise or counterclockwise 2, 5, or 8 places.  
 Fibonacci clock music: go clockwise or counterclockwise 0, 1, 2, 3, 5, or 8 places.  
 Prime time music: 2, 3, 5, 7, or 11 places.  
 Use a bigger clock. 18 tones? 24 tones? As many tones as you want.  
 And so on. Invent your kind of clock music.

## LOST SUBMARINE

It is night, a few stars are out, the ocean is blue. Alas, the Yellow Submarine is lost. With periscope up, it wanders back and forth, looking for ???



Write a program to make it happen. The yellow periscope should move at random right, left, right, left, and so on. When it changes from right to left, of course, it should point to the left.



We recommend graphics codes 157 and 158 for Yellow Submarine's periscope. Use the bottom third of the screen for the blue ocean and the top two-thirds for a black sky. SET a few stars in the sky.

Now, quickly close this book and write your program. Later, after your program is working, look at ours, if you want to.

We first wrote an outline of the program, using REM statements.

```

100 REM ** LOST SUBMARINE
200 REM ** BLACK SKY WITH STARS
300 REM ** BLUE OCEAN
400 REM ** YELLOW PERISCOPE
500 REM ** PERISCOPE MOVES RIGHT
600 REM ** PERISCOPE MOVES LEFT
700 REM ** GO MOVE AGAIN
800 REM ** TIME DELAY SUBROUTINE

```

Blocks 100 and 200 look easy.

```

100 REM ** LOST SUBMARINE
200 REM ** BLACK SKY WITH STARS
210 CLS 0
220 FOR STAR = 1 TO 7
230 OVER = RND(62)
240 DOWN = RND(15)
250 KOLOR = RND(8)
260 SET(OVER, DOWN, KOLOR)
270 NEXT STAR

```

We put the stars in the top half of the screen so there is no danger that Yellow Submarine's periscope will bump into a star. You might prefer to read star positions and color into arrays so that you can blink them on and off during the program.

Next, let's paint the ocean blue from print position 352 to print position 511. Oops! Almost forgot. If we PRINT @ 511, the screen will scroll. We show three ways to paint print positions 352 to 511 blue without scrolling when position 511 goes blue.

```

300 REM ** BLUE OCEAN
310 FOR OCEAN=352 TO 510
320 PRINT @OCEAN, CHR$(175);
330 NEXT OCEAN
340 SET(62, 30, 3); SET(63, 30, 3)
350 SET(62, 31, 3); SET(63, 31, 3)

```

Graphics character, solid blue

The FOR-NEXT loop (lines 310 to 330) puts the solid blue graphics character (ASCII code 175) in print positions 352 through 510. Lines 340 and 350 set print position 511 to all blue.

Here is another way.

```

300 REM ** BLUE OCEAN
310 FOR OCEAN=352 TO 510
320 PRINT @OCEAN, CHR$(175);
330 NEXT OCEAN
340 POKE 1535, 175

```



Line 340 POKES ASCII code 175 into memory location 1535. This location controls screen position 511. So that position becomes blue.

Hmmm . . . Why not just POKE the solid blue graphics character into all the video memory locations? Like this:

```
300 REM ** BLUE OCEAN
310 FOR OCEAN=352 TO 511
320 POKE@ OCEAN+1024, 175
330 NEXT OCEAN
```

Memory locations 1024 to 1535 correspond to screen print positions 0 to 511. To get the memory location that corresponds to a screen print position, add 1024 to the screen position.

$$\text{Memory Location} = \text{Screen Position} + 1024$$

This method is faster than the other two methods and the program block is shorter. Serendipity!

And now, here comes Yellow Submarine, represented by a nervous Yellow Periscope.

```
400 REM ** YELLOW PERISCOPE
410 SP = 320 + 7 + RND(16)
420 FOR K=1 TO 10
430 PRINT @SP, CHR$(158);
440 TT = RND(500); GOSUB 810
450 PRINT @SP, CHR$(157);
460 T = RND(500); GOSUB 810
470 NEXT K
```

Yellow Periscope appears at a random screen position between 328 and 343, inclusive. The screen position is selected in line 410. Yellow Periscope looks nervously right and left several times. Also try the following version of block 400.

```
400 REM ** YELLOW PERISCOPE
410 SP = 320 + 7 + RND(16)
420 FOR K=1 TO 10
430 PRINT @SP, CHR$(158);
440 TT = RND(500); GOSUB 810
450 PRINT @SP, CHR$(128);
460 PRINT @SP-1, CHR$(157);
470 TT = RND(500); GOSUB 810
480 PRINT @SP-1, CHR$(128);
490 NEXT K
```





Now on to block 500.

```
500 REM ** PERISCOPE MOVES RIGHT
510 YP$ = CHR$(158)
520 HF = RND(350 - SP)
530 FOR RT=1 TO HF
540 PRINT (@SP, CHR$(128));
550 SP = SP + 1
560 PRINT (@SP, YP$;
570 TT = 50: GOSUB 810
580 NEXT RT
```

In line 510, YP\$ is set equal to the yellow graphics character **█**. In line 520, HF means How Far. Its value will be random and will be less than the distance to the right edge of the screen. Remember, SP is the current screen position of Yellow Periscope (see line 410).

Lines 530 through 580 cause Yellow Periscope to move to the right HF spaces. Line 540 erases Yellow Periscope from its present position. Line 550 increases the screen position by one. Line 560 prints Yellow Periscope in its new position.

```
600 REM ** PERISCOPE MOVES LEFT
610 YP$ = CHR$(157)
620 HF = RND(SP - 321)
630 FOR LT=1 TO HF
640 PRINT (@SP, CHR$(128));
650 SP = SP - 1
660 PRINT (@SP, YP$;
670 TT = 50: GOSUB 810
680 NEXT LT

700 REM ** GO MOVE AGAIN
710 GOTO 510

800 REM ** TIME DELAY SUBROUTINE
810 FOR KK = 1 TO TT: NEXT KK
820 RETURN
```

Block 600 is similar to block 500, except that Yellow Submarine is moving left instead of right. Compare each line of block 600 with the corresponding line of block 500 and puzzle out what is happening.

Your turn. Modify the program so that, while Yellow Submarine scurries to and fro, the stars above twinkle merrily. Hmmm . . . instead of seven random stars, you might want to put some constellations in the sky. The Big Dipper? The North Star? Orion? Draco? The choice is yours.

---

## PROBLEMS WITHOUT SOLUTIONS

For your continued amusement, here are a few more problems, offered without solutions. They range from easy to hard to awful.

### PROBLEM 1. THREE-DIGIT NUMBER SPLITTER

We define a three-digit number as an integer from 100 to 999, inclusive. Write a program to ask for a three-digit number. When someone enters a three-digit number, split it into its three digits and print them separately.

Here is a sample RUN:

```
THREE-DIGIT NUMBER, PLEASE? 123
HUNDREDS DIGIT: 1
TENS DIGIT: 2
ONES DIGIT: 3

THREE-DIGIT NUMBER, PLEASE? 007
MUST BE AN INTEGER, 100 to 999.

THREE-DIGIT NUMBER, PLEASE? 12.3
MUST BE AN INTEGER, 100 to 999.

THREE-DIGIT NUMBER, PLEASE? █
```

And so on. Accept only integers in the range 100 to 999, inclusive.

### PROBLEM 2. SUM AND PRODUCT OF DIGITS

Write a program to compute the sum and product of the digits of a positive integer. Your program should work for positive integers up to 999999999. A RUN might look like this:

```
POSITIVE INTEGER? 123456
SUM OF DIGITS IS 21
PRODUCT OF DIGITS IS 720

POSITIVE INTEGER? 999999999
SUM OF DIGITS IS 81
PRODUCT OF DIGITS IS 387420489

POSITIVE INTEGER? 1.23
THAT'S NOT A POSITIVE INTEGER!

POSITIVE INTEGER? -123
THAT'S NOT A POSITIVE INTEGER!

POSITIVE INTEGER? 1000000000
TOO BIG. I CAN'T DO THAT ONE.

POSITIVE INTEGER? █
```

### PROBLEM 3. RUNNING DICE TOTALS

A six-sided die is thrown repeatedly until the running total is more than 12.

| Throw No. | Result | Total                   |
|-----------|--------|-------------------------|
| 1         | 4      | 4                       |
| 2         | 1      | 5                       |
| 3         | 2      | 7                       |
| 4         | 1      | 8                       |
| 5         | 4      | 12                      |
| 6         | 3      | 15 ←Stop, more than 12. |

This time, we got 15. Possible final totals range from 13 to 18. Write a program to simulate this process N times. Count the number of times each possible final total (13 through 18) occurred. After N times, print or show the results in a table such as the ones below. Two RUNs of our program are shown.

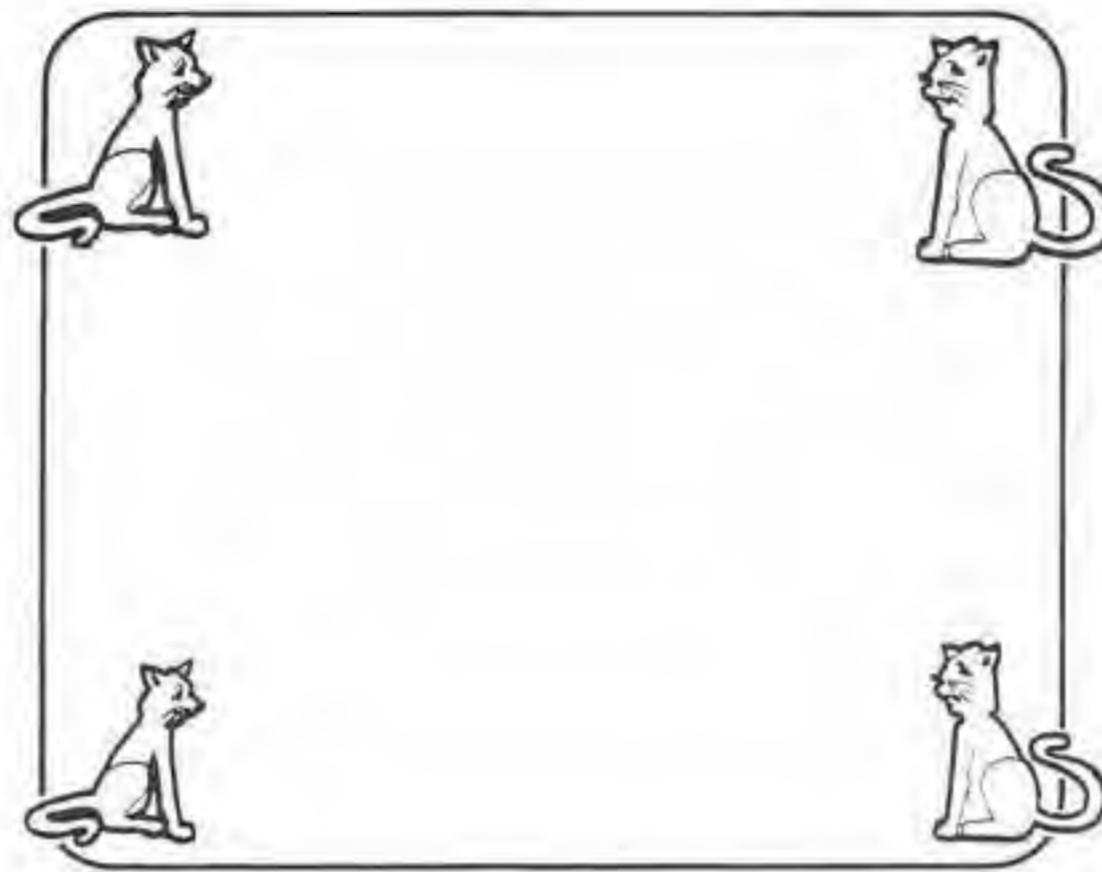
| HOW MANY TIMES? 1000 |                 | HOW MANY TIMES? 10000 |                 |
|----------------------|-----------------|-----------------------|-----------------|
| FINAL TOTAL          | NUMBER OF TIMES | FINAL TOTAL           | NUMBER OF TIMES |
| 13                   | 310             | 13                    | 2728            |
| 14                   | 261             | 14                    | 2448            |
| 15                   | 162             | 15                    | 1879            |
| 16                   | 139             | 16                    | 1499            |
| 17                   | 87              | 17                    | 960             |
| 18                   | 41              | 18                    | 486             |

How would we do this problem without a computer? For each possible final total, what odds might you give in a betting situation?

### PROBLEM 4. FOUR CATS CHASING

Imagine there are four cats in the four corners of the TV screen, one cat in each corner. Suddenly, the cats begin to chase one another. Each cat chases the cat in the nearest corner, clockwise from its corner.



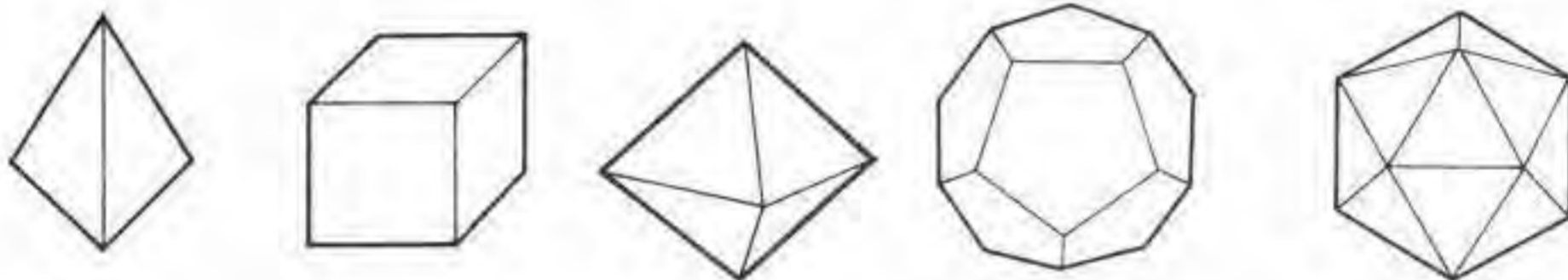


Write a program to make it happen on your TV. Will the cats ever meet? If so, where? Will their paths cross? Will there be a cat fight?

You might want to leave a trail of cat-tracks (paw pads?) behind each cat. We suggest you do this with SET commands, using four rectangular cats in four different colors. Use a variable time delay so you can easily speed up or slow down the action on the screen.

### PROBLEM 5. GAMEMASTER'S DICE

Gamemaster's dice are tetrahedra, hexahedra (cubes), octahedra, dodecahedra, and icosahedra.



Fantasy role-playing games use the following dice to simulate events in the nonreal world.

Abbreviation Description

|      |                                                                                                                                                                                                 |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| D4   | Tetrahedron, numbered 1 to 4.                                                                                                                                                                   |
| D6   | Hexahedron (cube), numbered 1 to 6.                                                                                                                                                             |
| D8   | Octahedron, numbered 1 to 8.                                                                                                                                                                    |
| D10  | Icosahedron, numbered 1 to 10, with each number appearing twice.                                                                                                                                |
| D12  | Dodecahedron, numbered 1 to 12.                                                                                                                                                                 |
| D20  | Icosahedron, numbered 1 to 20.                                                                                                                                                                  |
| D100 | Two icosahedra. Each die is numbered 0 to 9. One die gives the tens digit; the other gives the ones digit. A roll of 00 means 100. This pair of dice is used to roll percentages from 1 to 100. |
| D3   | A roll of a D6 with results as follows: 1 or 2 = 1, 3 or 4 = 2, 5 or 6 = 3.                                                                                                                     |

Events are sometimes determined by rolling two or more dice.

3D6        Roll three six-sided dice. The result is the sum of the individual dice (3 to 18).  
5D8        Roll five eight-sided dice. The possible results are 5 to 40.

Write a program to simulate gamemaster's dice. Here is a sample RUN of the program we have in mind.

|                    |                                    |
|--------------------|------------------------------------|
| DICE? D6           | We ask for one six-sided die.      |
| 3                  | the computer rolls a 3.            |
| DICE? 3D6          | We ask for three six-side dice.    |
| 14                 | Good roll!                         |
| DICE? 2D12         | We ask for two twelve-sided dice.  |
| 19                 |                                    |
| DICE? D100         | We ask for a percentage roll.      |
| 37                 | 37%.                               |
| DICE? 3 D 6        | The computer should ignore spaces. |
| 11                 |                                    |
| DICE? 3%6          | Must be getting tired . . .        |
| I DON'T UNDERSTAND |                                    |
| DICE? 2D7          | Not on our list of GM dice.        |
| I DON'T HAVE A D7  |                                    |
| DICE?              | And so on.                         |

If you are a gamemaster and lose your dice, just crank up your faithful Color computer to roll for you.

---

---

---

## CHAPTER THIRTEEN

# The Color BASIC Toolbox

---

---

This chapter is your quick-reference to the tools of Color BASIC, from ABS to VAL, for the Radio Shack Color Computer, Catalog Number 26-3001.

In this chapter, you will find a brief description of each BASIC word covered in this book. You will also find brief descriptions of BASIC words not covered elsewhere in this book. Look in Appendix K for sources of information and inspiration as you continue learning about Color BASIC and move on to learn about Extended Color BASIC.

### ABS

This numeric function computes the ABSolute value of a number.

$$\text{ABS}(X) = \begin{cases} X & \text{if } X \text{ is a positive number,} \\ 0 & \text{if } X \text{ is zero,} \\ -X & \text{if } X \text{ is a negative number.} \end{cases}$$

For example:

ABS(-3) is 3.    ABS(0) is 0.    ABS(7) is 7.

In parentheses following ABS, you may put a number, numeric variable, or numeric expression.

ABS(-2)    ABS(X)    ABS(X - G)

EXPERIMENT:

```
10 CLS
20 PRINT: INPUT X
30 PRINT ABS(X)
40 GOTO 20
```



## ASC

Computes the ASCII code of the *first character* in a string. The result is a number in the range 0 to 255, inclusive. See Appendix I for a list of ASCII codes.

ASC("D") is 68.                      ASC("DRAGON") is 68.  
 ASC("DOG") is 68.                  ASC("D6") is 68.  
 If N\$ = "DOROTHY" then ASC(N\$) is 68.

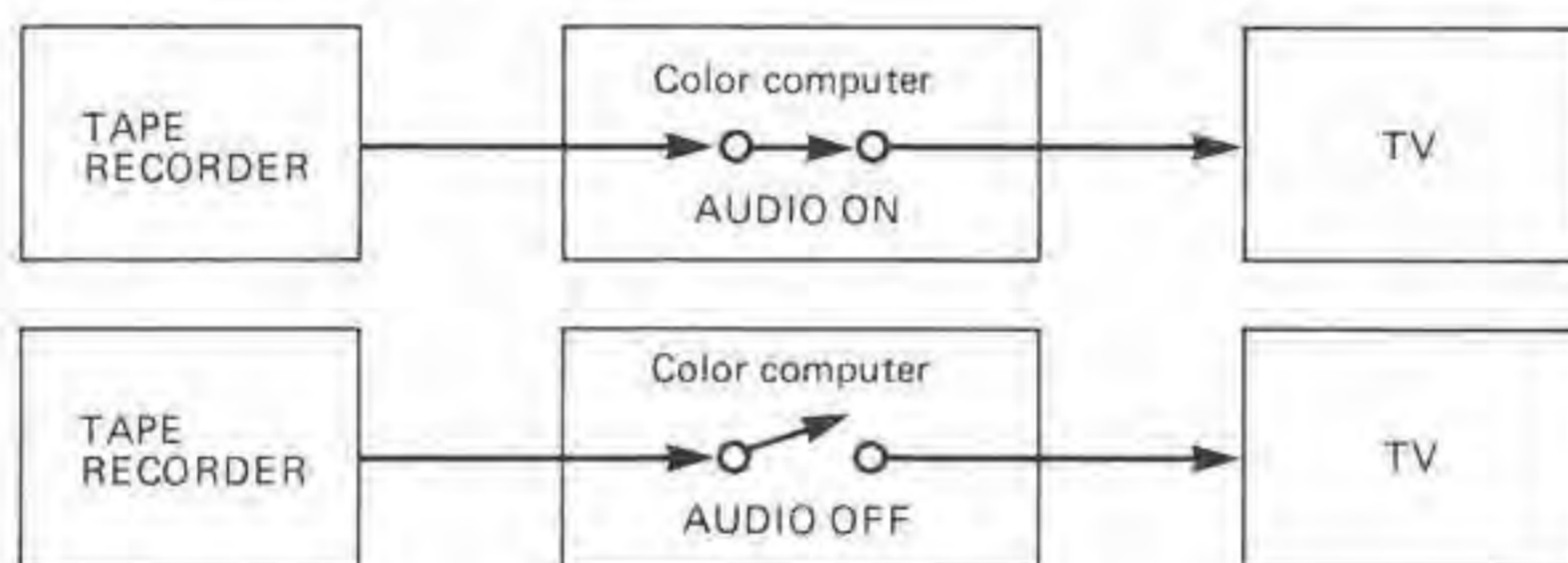
Remember, inside parentheses put only strings, string variables, or string expressions.

ASC("7") is 55, but ASC(7) causes a TM ERROR.  
 ASC(CHR\$(65)) is 65. Of course!  
 ASC(" ") is 32, but ASC("") gives an FC ERROR.  
           ↑                                    ↑  
       space                               empty  
                                               string

## AUDIO

If you have a cassette tape recorder connected to your Color Computer, which is connected to a color TV, you can use AUDIO to connect or disconnect the audio output of the tape recorder to the audio system of the TV.

Think of AUDIO as a switch.



Also see MOTOR, used to turn the tape recorder motor on or off.

## CHR\$

Note the dollar sign. CHR\$ is a *string function*. When you use CHR\$, the result is a string. CHR\$ computes a *one-character* string, the character that corresponds to an ASCII code.

CHR\$( )

Any whole number, 0 to 255.

CHR\$ computes the ASCII character that corresponds to the numeric ASCII code in parentheses. Yes, you can also put a numeric variable or expression in parentheses, as long as the value is 0 to 255, inclusive.

OK: CHR\$(0) CHR\$(255) CHR\$(X) CHR\$(X + 128)  
 NOT OK: CHR\$(-1) CHR\$(256) CHR\$(N\$)

CHR\$ and ASC are inverses of each other.

CHR\$(65) is A, and ASC("A") is 65.  
 CHR\$(ASC("A")) is A, and ASC(CHR\$(65)) is 65.

ASCII codes 128 to 255 are graphics characters. See Appendix H.

## CLEAR

Use CLEAR to tell the Color Computer to reserve memory space for strings. If you don't, the computer automatically reserves room for 200 characters. Then, if you try to use 201 characters, you will get an OS ERROR.

CLEAR 500 Save memory space for 500 characters.  
 CLEAR 100 Save memory space for 100 characters.

## CLOAD

CLOAD tells the computer to find and load a BASIC program from a cassette tape.

CLOAD Find and load the first program.  
 CLOAD "MANDALA" Find and load the program called MANDALA. If there are programs on the tape before MANDALA, they will be skipped over as the computer searches for MANDALA.

## CLOADM

Tells the computer to find and load a *machine-language* program from a cassette tape. Not discussed in this book. See Appendix K for sources of additional information.

## CLOSE

Closes a file by shutting off communication to a file device.

CLOSE #-1      Close file—cassette recorder.  
CLOSE #-2      Close file—printer.

This book does not cover files. See Appendix K for sources of information.

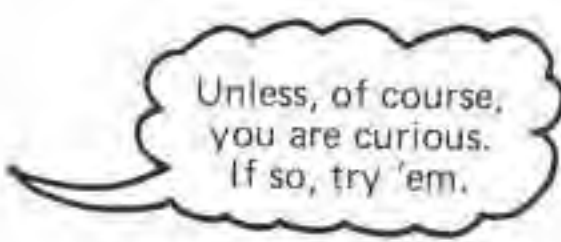
## CLS

Clears the screen to one of 9 colors, including black, as follows:

|       |        |       |         |
|-------|--------|-------|---------|
| CLS   | Green  | CLS 4 | Red     |
| CLS 0 | Black  | CLS 5 | Buff    |
| CLS 1 | Green  | CLS 6 | Cyan    |
| CLS 2 | Yellow | CLS 7 | Magenta |
| CLS 3 | Blue   | CLS 8 | Orange  |

Following CLS, you may put nothing at all, a whole number from 0 to 8, a numeric variable, or a numeric expression. The color code may be enclosed in parentheses.

CLS 3    or    CLS(3)  
CLS C    or    CLS(C)  
CLS T/32    or    CLS(T/32)  
Try these:    CLS (7.9)    CLS (2.3)  
But not these:    CLS(-1)    CLS(9)



Unless, of course,  
you are curious.  
If so, try 'em.

## CONT

If you stop the computer by pressing the **BREAK** key, type CONT and press **ENTER**. Presto! The computer CONTinues from where it stopped. This also works if the computer stopped because of a STOP statement in the program.

## CSAVE

Use CSAVE to put your BASIC programs on cassettes. Having done so, you can later use CLOAD to quickly load them into memory, thus saving your aching fingers for more interesting work.

---



|                 |                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------|
| CSAVE           | Copy a program from the computer's memory to a cassette tape.                                         |
| CSAVE "MANDALA" | Copy a program to cassette tape and call it MANDALA. Aha! Now the computer can search for it by name. |

## CSAVEM

Advanced stuff, not covered in this beginner's book. When you learn to write *machine language* programs, you will use CSAVEM to put them on cassettes so you can later load them back into the computer with CLOADM. See Appendix K for ideas on where to go next to advance from intermediate to advanced to expert to . . . ???

## DATA

DATA and READ are companion statements. Each is nothing without the other. Use DATA to store information to be read by READ.

DATA 89, 108, 125, 133, 147, 159, 170, 999

No comma here.    Commas between items.    No comma here.

Since commas are used *between* items, they can't be used *within* a single item. Unless . . .

DATA "COMPUTERTOWN, USA!"

OK here because string information is enclosed in quotation marks.

DATA "FIREDRAKE, GEORGE", "HARRIS, BOB"

This comma is part of a string. It is *inside* quotes.

This comma is part of a string. It is *inside* quotes.

This comma separates two strings. It is *outside* quotes.

Also see READ and RESTORE.

## DIM

Reserves space for arrays of subscripted variables.

|                 |                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------|
| DIM NAME\$(100) | Reserves space for the string array NAME\$. This array may have up to 101 elements from NAME\$(0) to NAME\$(100). |
| DIM T(24)       | Reserves space for the numeric array called T. This array may have up to 25 elements from T(0) to T(24).          |

You can reserve space for two or more arrays in a single DIM statement.

|                              |                       |
|------------------------------|-----------------------|
| DIM NAME\$(100), PHONES(100) | Two string arrays.    |
| DIM T(24), D(24), C(24)      | Three numeric arrays. |
| DIM CHAR\$(7), CHAR(7)       | One of each.          |

And so on. Remember, you can DIM an array once. Put your DIMs early in the program, before you use the arrays. If you don't DIM an array, the computer does it for you, allowing subscripts from 0 to 10.

If you use a large string array, remember to also use CLEAR to reserve memory space for the strings.

## END

When the computer sees this, it quits. Put END in a program to stop the computer. The statement STOP also stops the computer. Try both of these to see the slight difference.

|                          |                           |
|--------------------------|---------------------------|
| 10 CLS                   | 10 CLS                    |
| 20 PRINT "END DOES THIS" | 20 PRINT "STOP DOES THIS" |
| 30 END                   | 30 STOP                   |

If you don't use STOP or END, the computer . . .

```
10 CLS
20 PRINT "NEITHER END NOR STOP"
```

*Experiment!*

## EOF

EOF means *End Of File*. We don't cover *files* in this book—after all, as a beginner, do you want a book that has 1,000 pages, is 3 inches thick, and costs \$29.95? See Appendix K to find out how to find out about files.

---

## EXEC

More advanced stuff. EXEC transfers control to a machine language program (whatever *that is!*) at the address (???) you specify.

EXEC 29999 Start execution at 29999.

Or, if you have just loaded a program, using CLOADM:

EXEC Start at the address specified the last time you used CLOADM.

Sorry, in this beginner's book, we don't talk about this. If you buy a program that requires CLOADM and EXEC, read and save the written information that comes with it. This information will tell you how to get the program into your Color Computer (CLOADM) and then use it. If it doesn't, ask for your money back!

## FOR-NEXT LOOP (FOR, NEXT, AND STEP)

A FOR-NEXT loop usually consists of three things:

1. A FOR statement.
2. A NEXT statement.
3. One or more statements between the FOR statement and the NEXT statement.

For example:

```
10 CLS
20 FOR X=1 TO 7
30 PRINT X
40 NEXT X
```



A FOR-NEXT loop begins with a FOR statement and ends with a NEXT statement. The FOR statement defines a *sequence of values* for the variable immediately following the word FOR.

| FOR Statement  | Variable | Sequence of Values        |
|----------------|----------|---------------------------|
| FOR X=1 TO 7   | X        | 1, 2, 3, 4, 5, 6, 7       |
| FOR K=1 TO 100 | K        | 1, 2, 3, . . . , 100      |
| FOR C=0 TO 8   | C        | 0, 1, 2, 3, 4, 5, 6, 7, 8 |
| FOR F=.5 TO 3  | F        | .5, 1.5, 2.5              |



The value of the variable increases by one each time, unless you add a STEP clause, as follows:

| FOR Statement         | Variable | Sequence of Values     |
|-----------------------|----------|------------------------|
| For X=1 TO 7 STEP 2   | X        | 1, 3, 5, 7             |
| FOR Z=10 TO 0 STEP -1 | Z        | 10, 9, 8, ..., 2, 1, 0 |

**EXPERIMENT!** Use this program to gather information about FOR-NEXT loops.

```
10 CLS
20 INPUT "FROM"; A
30 INPUT "TO"; B
40 INPUT "STEPS OF"; H
50 PRINT
60 FOR X=A TO B STEP H
70 PRINT X
80 NEXT X
90 PRINT: GOTO 20
```

Variations:  
70 PRINT X;  
70 PRINT X;

## GOSUB

Tells the computer to use a subroutine. The subroutine must include a RETURN command to send the computer back to the statement following GOSUB.

**GOSUB 810**    Go use a subroutine, beginning at line 810. When a RETURN is executed, return to the command following this GOSUB.

## GOTO

Tells the computer to go to a line and continue executing from that line.

**GOTO 210**    Go to line 210 and continue from there.  
(Do not collect \$200.)  
(Do not pass Go)

---

## IF . . . THEN . . . ELSE

This statement has a number of forms.

—IF *condition* THEN *statement*

The condition can be *true* or *false*. If it is true, the computer executes the statement following THEN. If the condition is false, the computer does not execute the statement. Examples:

IF X < 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"

IF GS > XS THEN PRINT "TRY A LOWER WORD"

IF OVER > 63 THEN OVER = 63

IF N > INT(N) THEN GOTO 310

—IF *condition* THEN *line number*

This is the same as IF *condition* THEN GOTO *line number*.

IF N <> INT(N) THEN 310

—IF *condition* THEN *statement* ELSE *statement*

If condition is TRUE, do this.

If condition is FALSE, do this.

IF R=1 THEN CS = "H" ELSE CS = "T"

IF INKEY\$="" THEN GOTO 720 ELSE GOTO 210

—IF *condition* THEN *line number* ELSE *line number*

This is the same as: IF *condition* THEN GOTO *line number* ELSE GOTO *line number*.

IF INKEY\$="" THEN 720 ELSE 210

## INKEY\$

Scans the keyboard. If a key has been pressed since the last time INKEY\$ was used, INKEY\$ is set equal to that keyboard character. If no key has been pressed, the value of INKEY\$ will be the empty string (""). Examples:

KEY\$ = INKEY\$

IF INKEY\$="S" THEN 210

IF INKEY\$="" THEN 720 ELSE 310

## INPUT

Tells the computer to print a question mark, turn on the cursor, and wait for someone to enter information.

INPUT T

INPUT "YOUR NAME"; N\$

INPUT "TONE, DURATION"; T, D

INPUT can also be used to acquire information from a file device.

INPUT #-1, NAMES, PHONES      Obtain information from cassette tape.

We do not cover *files* in this book. See Appendix K for pointers to information about files.

## INT

This numeric function computes the greatest integer that is less than or equal to a given number.

|                 |                  |
|-----------------|------------------|
| INT(3.7) is 3.  | INT(-3.7) is -4. |
| INT(8) is 8.    | INT(-8) is -8.   |
| INT(.001) is 0. | INT(-.001) is 0. |
| INT(0) is 0.    | INT(.99) is 0.   |

If X is positive or zero, INT can be used to round X to the nearest integer, tenth, hundredth, and so on.

|                           |                                   |
|---------------------------|-----------------------------------|
| Round to nearest integer: | $XR = \text{INT}(X + .5)$         |
| nearest tenth:            | $XR = \text{INT}(10*X + .5)/10$   |
| nearest hundredth:        | $XR = \text{INT}(100*X + .5)/100$ |

INT is also handy for computing the remainder R on dividing a positive integer N by a positive integer D.

$$R = N - D * \text{INT}(N/D)$$

Remember the good old days (?) when you were learning long division?

|                       |                                                                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| $Q = \text{INT}(N/D)$ |                                                                                                                                                  |
| $R = N - D * Q$       |                                                                                                                                                  |
|                       | $\begin{array}{r} 19 \leftarrow Q \\ 12 \overline{) 235} \leftarrow N \\ \underline{12} \\ 115 \\ \underline{108} \\ 7 \leftarrow R \end{array}$ |



## JOYSTK

Tells the computer to read the horizontal or vertical coordinate of the right joystick or the left joystick.

```
JOYSTK(0) horizontal coordinate, right joystick
JOYSTK(1) vertical coordinate, right joystick
JOYSTK(2) horizontal coordinate, left joystick
JOYSTK(3) vertical coordinate, left joystick
```

A joystick value is a whole number from 0 to 63.



## LEFT\$

This string function picks off the left part of a string.

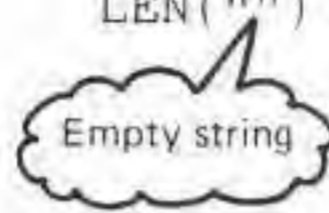
```
LEFT$("ABC", 1) is "A"
LEFT$("ABC", 2) is "AB"
LEFT$("ABC", 3) is "ABC"
LEFT$("ABC", 4) is "ABC"
```

LEFT\$(X\$, N) is a string consisting the leftmost N characters of X\$.

## LEN

LEN is a numeric function that computes the length of a string. Its value is always a non-negative integer.

LEN("A") is 1.      LEN("AB") is 2.  
LEN("ABC") is 3.    LEN("ABCD") is 4.  
LEN(" ") is 1.      LEN("") is 0.



IF X\$ is "ABC", then LEN(X\$) is 3.

## LIST

Lists a program on the screen.

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| LIST         | Lists the entire program.                                               |
| LIST 100     | Lists only line 100.                                                    |
| LIST 100-250 | Lists lines 100 through 250.                                            |
| LIST 100-    | Lists from line 100 to the end of the program.                          |
| LIST -250    | Lists from the beginning of the program up to, and including, line 250. |

## LLIST

Same as LIST, except that the program, or parts of it, is listed on a printer. Of course, you must have a printer connected to the computer for this to work.

## MEM

This is a function whose value is the number of memory locations still unused in the computer's memory. As you use up memory space, the value of MEM becomes less, and less, and . . .

PRINT MEM      Prints the amount of unused memory space.

---

## MID\$

This string function picks off a substring from anywhere within a string.

Let X\$ be "ABCDE"

MID\$(X\$, 1, 1) is "A"  
MID\$(X\$, 1, 2) is "AB"  
MID\$(X\$, 1, 5) is "ABCDE"  
MID\$(X\$, 2, 3) is "BCD"

In parentheses, you specify the string, which character to begin at, and how many characters to take.

MID\$(X\$, 2, 3)  
From this string      Start here.      take this many characters.

## MOTOR

Turns the motor of the cassette recorder ON or OFF.

MOTOR ON      Turns the cassette recorder motor ON.  
MOTOR OFF     Turns the cassette recorder motor OFF.

## NEW

Erases everything in memory (RAM, that is).

## ON . . . GOSUB

This is a multiple GOSUB. The computer will GOSUB to one or more line numbers depending on the value of a numeric variable or expression.

ON X GOSUB 710, 810

If X is 1, the computer GOSUBs to line 710. If X is 2, the computer GOSUBs to line 810.

ON SGN(X)+2 GOSUB 520, 530, 540  
SGN(X)+2=1      SGN(X)+2=2      SGN(X)+2=3

ON expression GOSUB line number, line number, . . .  
expression=1.      expression=2, . . .



## ON . GOTO . . .

A multiple GOTO. The computer goes to the 1st line number, 2nd line number, and so on, depending on the value (1, 2, and so on) of a numeric variable or expression.

```
ON X GOTO 710, 810
If X is 1, GOTO 710. If X is 2, GOTO 810.
ON SGN(X)+2 GOTO 520, 530, 540
 SGN(X)+2=1 SGN(X)+2=2 SGN(X)+2=3
```

```
ON expression GOTO line number, line number, . . .
 expression=1. expression=2, . . .
```

## OPEN

Opens communication to a file device.

```
OPEN "I", #-1, "GAMES" Opens communication to cassette re-
 recorder for INPUT of information.
OPEN "O", #-1, "GAMES" Opens communication to cassette re-
 recorder for Output of information.
OPEN "O", #-2, "GAMES" Opens communication to printer for
 Output of information.
```

File name may be up to 8 characters.

We do not cover file programming in this book. See Appendix K for sources of information.

## PEEK

PEEK is a numeric function that lets you peek into the computer's memory. The result of a PEEK is a whole number from 0 to 255, inclusive. Let's PEEK.

```
PRINT PEEK(0) Prints the number in memory location 0.
SW = PEEK(65280) Assigns the number in memory location
 65280 to the numeric variable SW.
```

The number in parentheses following PEEK is called a *memory address*, or just *address*. It identifies a particular location in the computer's memory. An address may be any whole number from 0 to 65535.

In a program, you can use PEEK to find out what is being shown at any screen PRINT position. Each screen position has a corresponding memory location.

| Screen Position | Memory Location |
|-----------------|-----------------|
| 0               | 1024            |
| 1               | 1025            |
| 2               | 1026            |
| .               | .               |
| .               | .               |
| .               | .               |
| 511             | 1535            |

Let ML stand for Memory Location and SP for Screen Position.

$$ML = SP + 1024$$

So, PEEK(1024) tells you what is being shown at screen position 0, PEEK(1025) at screen position 1, and so on. The result of a PEEK is the ASCII code of the character on the screen.

## POINT

This numeric function tests a single small point on the screen. You specify the point on the screen, as follows:

POINT(OVER, DOWN)  
 0 to 63      0 to 31

The point on the screen might be a small rectangular blip set by a SET command or a CLS KOLOR command. It might be one of eight colors, or it might be black.

It might also be none of the above. It might be part of the letter A, or B, or %, or any other PRINTed character.

POINT tells all, as follows:

| VALUE OF POINT<br>(OVER,DOWN) | MEANING OF POINT<br>(OVER,DOWN)                                                                                                          |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| -1                            | This screen location is part of an ASCII character.                                                                                      |
| 0                             | Location is OFF (black)                                                                                                                  |
| 1 to 8                        | Location is ON. In this case, the value is a color code. POINT tells you what color is turned on at that screen location (SET location). |

## POKE

What PEEK is to looking, POKE is to putting something there to look at. You can use POKE to put a number from 0 to 255 into any memory location, provided the location is RAM. You can't POKE numbers into ROM!

We don't do much with PEEK and POKE in this book. If we covered everything in enough detail so you really understood it, you could hardly carry the book! (Much less afford it.)

For novices, perhaps the most interesting places to POKE into are memory locations that correspond to screen PRINT positions.

$$\begin{aligned}\text{Memory Location} &= \text{Screen Position} + 1024 \\ \text{ML} &= \text{SP} + 1024\end{aligned}$$

So, POKE some stuff into the screen.

|                |                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------|
| POKE 1535, 42  | POKE an asterisk (ASCII code 42) into memory location 1535. Aha! That's screen position 511. And—it didn't scroll! |
| POKE 1535, 65  | POKE an "A" into ML = 1535 (SP=511).                                                                               |
| POKE 1535, 163 | POKE a yellow graphics character into ML = 1535 (SP=511).                                                          |

And so on. But, POKE carefully. You can clobber BASIC by indiscriminate POKEing. If you want to POKE around in some other books that might teach you about POKE, please POKE into Appendix K.

## PRINT

Tells the computer to print something on the screen or on a specified output device.

|                |                                            |
|----------------|--------------------------------------------|
| PRINT X        | Print the value of X on the screen.        |
| PRINT #-1, N\$ | Print the value of N\$ on a cassette tape. |
| PRINT #-2, N\$ | Print the value of N\$ on a printer.       |

When PRINTing on the screen, you can use punctuation symbols to tell the computer how to print things. Try these:

```
PRINT 1, 2, 3
PRINT 1: 2: 3
PRINT "1", "2", "3"
PRINT "1"; "2"; "3"
PRINT "1" "2" "3"
PRINT "1" + "2" + "3"
```

**EXPERIMENT!**

And remember what happened.



## PRINT @

Tells the computer to print something on the screen at a specified screen location, 0 to 511.

```
PRINT @100, X Print the value of X at screen location 100.
PRINT @235, "*" Print an asterisk (*) at screen location 235.
```

Alas, if you PRINT @511, the entire screen scrolls up one line. If you want to put something at screen location 511 without scrolling, you can POKE it.

```
POKE 1535, 42 Puts an asterisk at screen location 511,
 without scrolling.
POKE 1535, ASC("A") Puts an A at screen location 511, without
 scrolling.
```

See Appendix F for a screen map showing all screen locations. And see POKE for more about POKE.

## READ

Reads information from a DATA statement and assigns it to a variable or variables in the READ statement.

```
READ X Read one number from a DATA statement and
 assign it as the value of X.
READ N$ Read one string from a DATA statement and assign
 it as the value of N$.
READ X, N$ Read one number and one string from a DATA
 statement. Assign the number to X and the string to
 N$.
```

Also see DATA. Of course! Without DATA, READ can do nothing.

## REMARK or REM

This REMarkable statement is used to REMind people what the program does. The computer ignores everything in a REM statement, but people find them very useful when they try to read and understand a program.

```
REM ** TIME DELAY SUBROUTINE
REM ** A BLUE TONE
REM ** DRAW A RED HORIZONTAL LINE
```

## RESET

Turns off a blip on the screen that was turned on using a SET statement. Actually, RESET simply makes that blip go black.

RESET(OVER, DOWN)  
          ↑          ↑  
          0 to 63   0 to 31

Examples? Of course.

RESET(7, 22) Turn off the blip (make it black) at OVER = 7,  
                  DOWN = 22.

Hmmm . . . suppose a tiny screen location contains part of the letter A. What happens if you try to RESET it?

```
10 CLS
20 PRINT 92, "A";
30 FOR K=1 TO 920: NEXT K ← Two-second delay.
40 RESET(0, 12) ← This is the upper left corner of
50 GOTO 50 PRINT position 192.
```

EXPERIMENT!

## RESTORE

Tells the computer to begin at the first item in the first DATA statement. Use this when you want the computer to reuse information in DATA statements.

## RETURN

Use this to terminate a subroutine and return to the statement following the GOSUB that sent the computer to the subroutine. GOSUB and RETURN always work together.

---

## RIGHT\$

This string function picks off the right part of a string.

```
RIGHT$("ABC", 1) is "C"
RIGHT$("ABC", 2) is "BC"
RIGHT$("ABC", 3) is "ABC"
RIGHT$("ABC", 4) is "ABC"
```

RIGHT\$(X\$, N) is a string consisting of the rightmost N characters of X\$.

## RND

The mysterious and unpredictable RND function computes a random integer from 1 to the maximum integer you specify.

```
RND(2) is a random integer, 1 or 2.
RND(3) is a random integer, 1, 2, or 3.
RND(100) is a random integer from 1 to 100.
```

If N is a positive integer, then RND(N) is a random integer from 1 to N.

This is one of the nice features of Radio Shack Computers that make them easy to use for people who don't have degrees in mathematics. Thanks, Steve Leininger!

You may also use RND to compute random decimal fractions between 0 and 1.

```
RND(0) is a random decimal fraction between 0 and 1. Other
 computers do it this way. Why? Because the original
 BASIC did it this way. Thanks again, Radio Shack.
```

To get integers from 1 to 6 using RND(0), do this:

```
INT(6*RND(0)) + 1
```





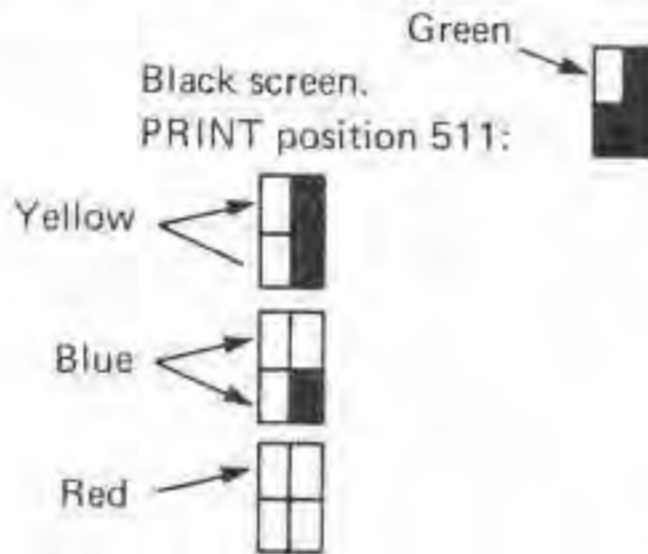
## SET

Turns on a tiny rectangular color blip on the screen.

|                        |                                                 |
|------------------------|-------------------------------------------------|
| SET(0, 0, 1)           | Turns on a green blip at<br>OVER=0, DOWN=0.     |
| SET(63, 31, 8)         | Turns on an orange blip at<br>OVER=63, DOWN=31, |
| SET(OVER, DOWN, KOLOR) |                                                 |
|                        |                                                 |

Within a PRINT position (0 to 511), all SET rectangles will be the same color or black. They will be the *last* color SET. For example, try this while watching the lower right corner of the screen:

```
CLS 0
SET(62, 30, 1)
SET(62, 31, 2)
SET(63, 30, 3)
SET(63, 31, 4)
```



Yes, within a single PRINT position, each SET changes all previously SET colors to the most recent color.

## SGN

This numeric function has three possible values (-1, 0, 1) that depend on whether a number is negative, zero, or positive.

SGN(X) is -1 if X is negative.  
 SGN(X) is 0 if X is zero.  
 SGN(X) is 1 if X is positive.

Well, that covers all the possibilities. A number must be exactly one of these things: negative, positive, or zero.

## SIN

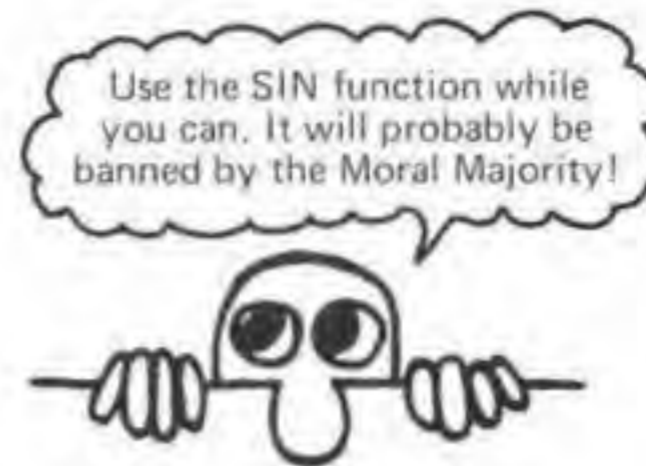
No, it is not wrongdoing of the sort that leads to perdition. It is simply the *sine* function of trigonometry.

---

`SIN(X)` is the sine function of  $X$ , where  $X$  is assumed given in radians. The value of `SIN(X)` will be a number from  $-1$  to  $1$ , inclusive.

Try this program and variations:

```
10 CLS
20 X = 0
30 T = 15 + 15*SIN(X)
40 PRINT TAB(T) "*"
50 X = X + 125
60 GOTO 30
```



VARIATIONS (any or all)

```
50 X = X + .25 or 50 X = X + .0625
30 T = 15 + 8*SIN(X) + 7*SIN(2*X)
```

## SKIPF

Tells the computer to skip to the end of the next program on cassette tape, or to the end of the program you name.

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>SKIPF</code>           | Skip to the end of the next program on cassette tape. |
| <code>SKIPF "MANDALA"</code> | Skip to the end of the program called MANDALA.        |

Why skip? Well, you may want to `CLOAD` the program that begins immediately after `MANDALA`. By skipping to the end of `MANDALA`, the computer is ready to `CLOAD` the next program.

Or, you may wish to skip to the end of the *last* program on tape so you can then `CSAVE` a new program, *without* erasing a previously recorded program.

## SOUND

Tells the computer to play a tone. You specify a tone number and duration number.

```
SOUND 89, 20
 ↑ ↙
Play this tone for this long (duration).
```

A tone number may be 1 to 255. A duration number may be 1 to 255. See Appendix G for more information.

## STOP

Stops the computer. See also CONT.

## STR\$

This string function computes the string equivalent to a number.

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| STR\$(123)  | is a four-character string consisting of space, 1, 2, and 3.               |
| STR\$(-123) | is a four-character string consisting of a minus sign, 1, 2, and 3.        |
| STR\$(3.7)  | is a four-character string consisting of a space, 3, decimal point, and 7. |

The string equivalent of a positive number begins with a space (ASCII code 32). The string equivalent of a negative number begins with a minus sign (ASCII code 45).

You may wonder why anyone would want to have a STR\$ equivalent to a number. One reason might be to get easy access to the digits of the number. Do you have a credit card? One of the digits of your credit card is a check digit, computed from the other digits. See TWO-DIGIT NUMBER SPLIT-TER in Chapter 12 for an example of how to use STR\$.

## TAB

Use TAB in a PRINT statement to tell the computer to TAB (move to the right) to a specified PRINT position on the line. TAB(0) is the left edge of the screen; TAB(31) is the right edge of the screen.

|                        |                                                                                                     |
|------------------------|-----------------------------------------------------------------------------------------------------|
| PRINT TAB(15) "*"      | Move to the 15th position on the print line and print an asterisk.                                  |
| PRINT @96, TAB(15) "*" | Move to the 15th position on the print line beginning at PRINT position 96, then print an asterisk. |

For any line, TAB positions begin at the left edge of the screen with TAB(0). TAB(31) is the right edge of the screen. Hmmm . . . what if you TAB(32) or TAB(46) or TAB(123)? Try it.

## USR

Calls a machine language subroutine. Not covered in this book. See Appendix K for sources of information.

---



## VAL

Computes the numeric equivalent of a string number.

```
VAL("123") is 123.
VAL("-123") is -123.
VAL("123ABC") is 123.
VAL("ABC") is 0.
If X$ = "37", then VAL(X$) is 37.
```

That's it. Color BASIC tools, ABS to VAL. Use them to build BASIC programs, or to refurbish BASIC programs written by others, or simply to understand what a BASIC program does.

---

---

---

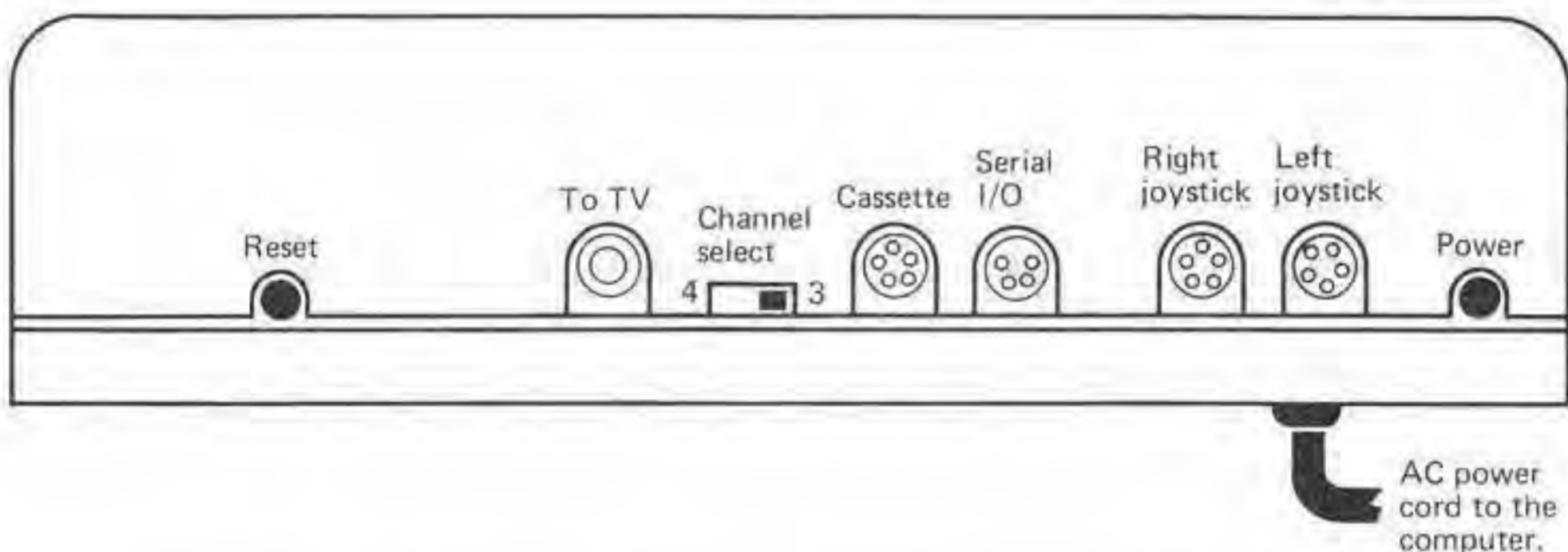
## APPENDIX A

# Hooking Up Your Color Computer

---

---

Your Color Computer has a built-in interface that sends signals to a TV through a cable attached to both the Color Computer and the TV. Turn your computer around and look at the back. It looks like this:

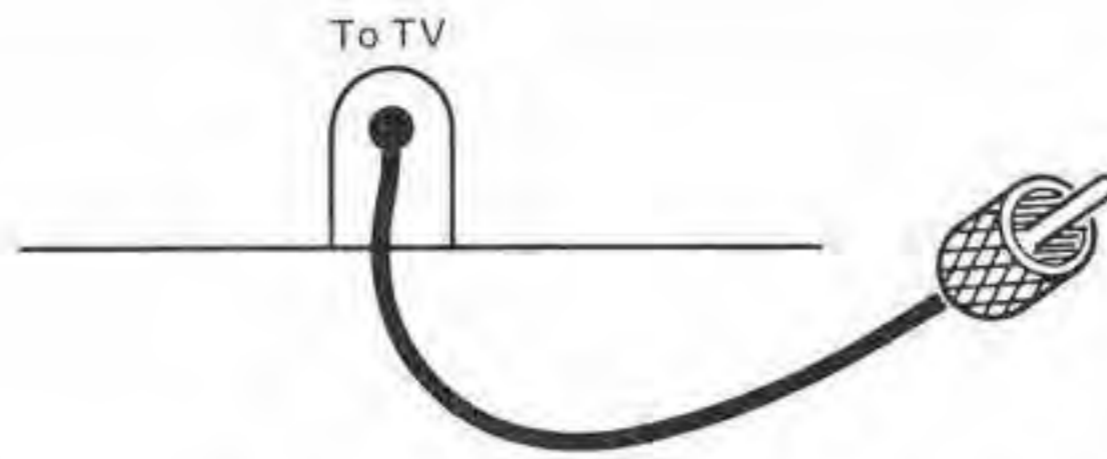


The AC power cord is permanently attached to the computer. Plug it into a 110-volt outlet. Press the power button to turn on the Color Computer. Press it again to turn off the computer. When the button is farthest out, the computer is off.

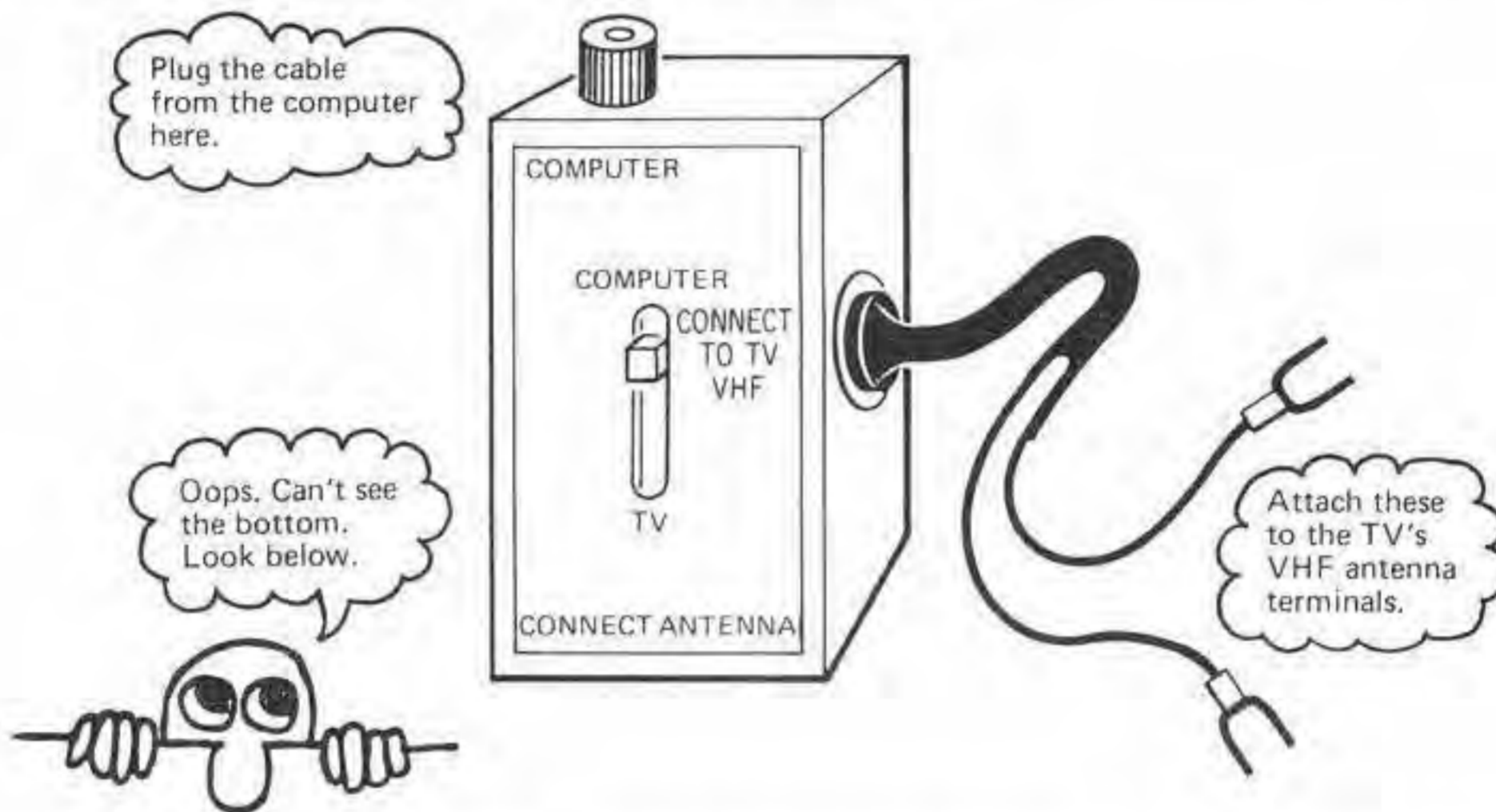
Before hooking up the Color Computer to the TV, turn the computer OFF. The cable that connects the computer to the TV looks like this:



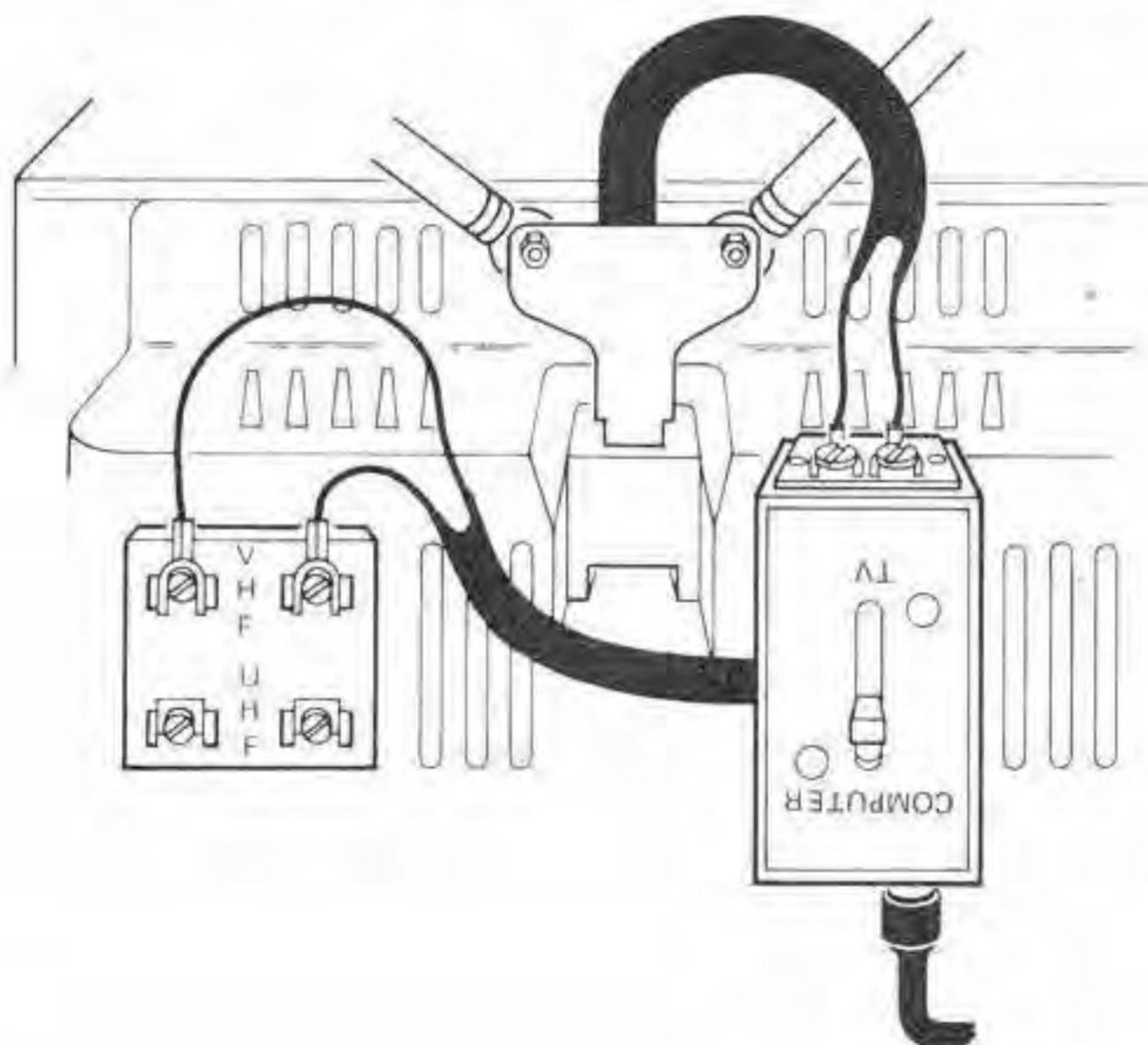
Plug either end into the computer output labelled TO TV.



As you may suspect, the other end of the cable should be attached to the TV. Not quite. Instead, it plugs into a switchbox that comes with the Color Computer. The switchbox looks like this:



The bottom of the switchbox has two screw terminals. Connect the TV's VHF antenna to these. A complete hook-up might look like this:

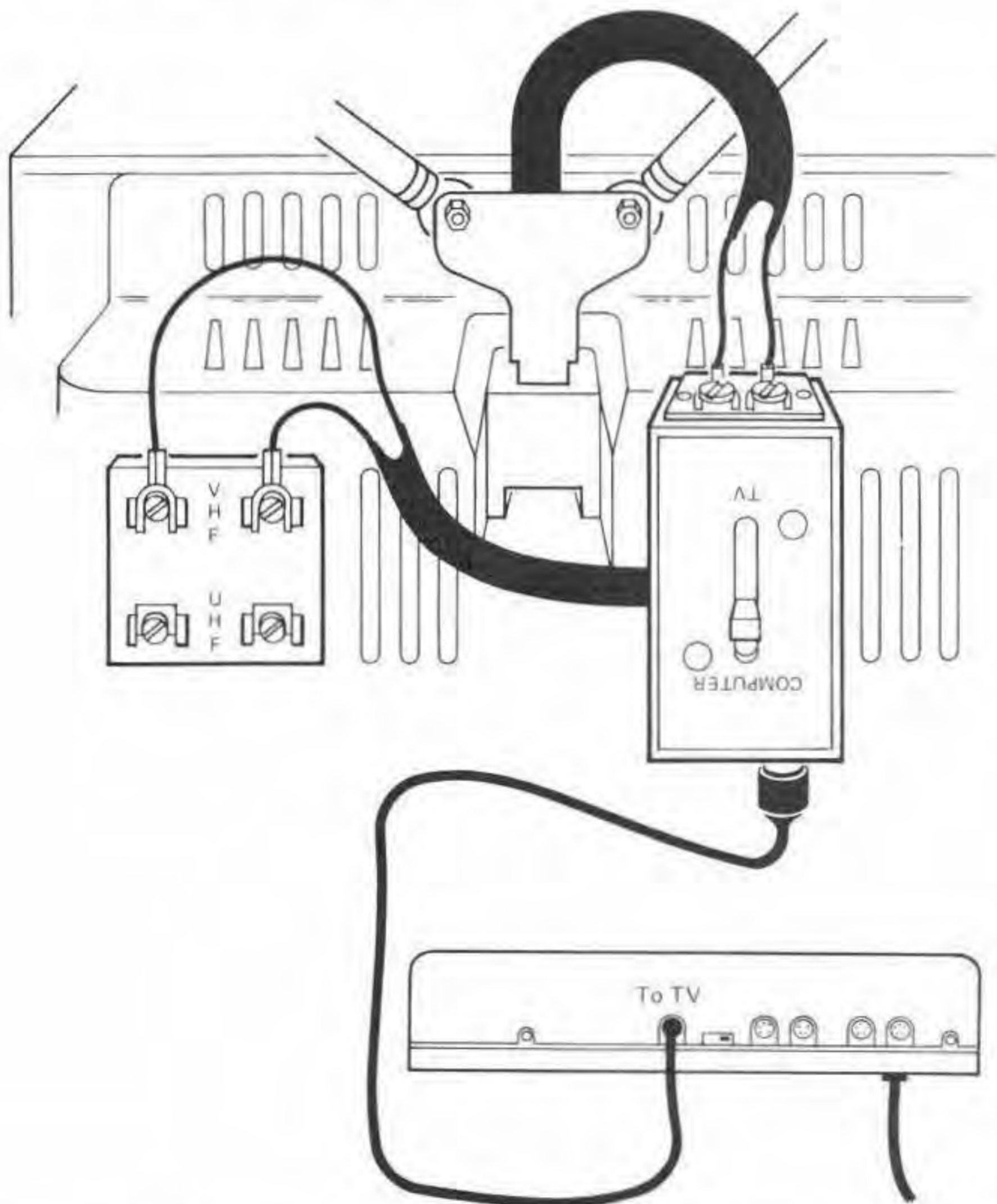




One side of the switchbox has sticky tape attached. Peel off the backing and stick the switchbox to your TV. Then, make all the above connections.

- Plug the cable into the computer (TO TV) and into the switchbox.
- Attach the switch box to the VHF (NOT UHF) antenna terminals on the TV.
- Attach the VHF (NOT UHF) antenna to the switchbox.

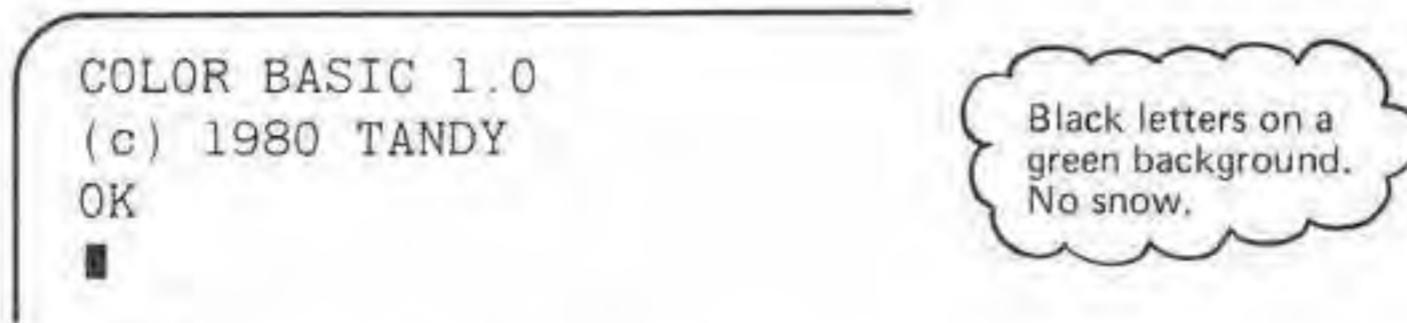
Done? Yes? Good. It should look something like this, if you are standing behind the computer and TV:



Does your TV have a 75/300 ohm switch? It might be labeled 75  $\Omega$  at one setting and 300  $\Omega$  at the other setting. Set it to 300 ohms (300  $\Omega$ ).

Depending on where you live, use either channel 3 or channel 4 on your TV. Try both.

- On the back of the computer, set the CHANNEL SELECT switch to 3. On the TV, set the channel to 3.
- Turn on the computer and turn on the TV. You should see something like this:



If you see this, or something similar or other channels, and the picture is good, go with it! Otherwise, try channel 4.

- On the computer, set the CHANNEL SELECT switch to 4.
- On the TV, set the channel to 4.

Hopefully, you are now up and computing. If not,

- carefully check all connections.
- call the nearest Radio Shack store.
- yell for help! There are so many Color Computers in homes, schools, and elsewhere, that someone might hear you and come to the rescue.

For more information, and for other ways of hooking up, read the *TRS-80™ Color Computer Operation Manual*, available from Radio Shack.

---

---

## APPENDIX B

# Using the Tape Cassette Recorder

---

---

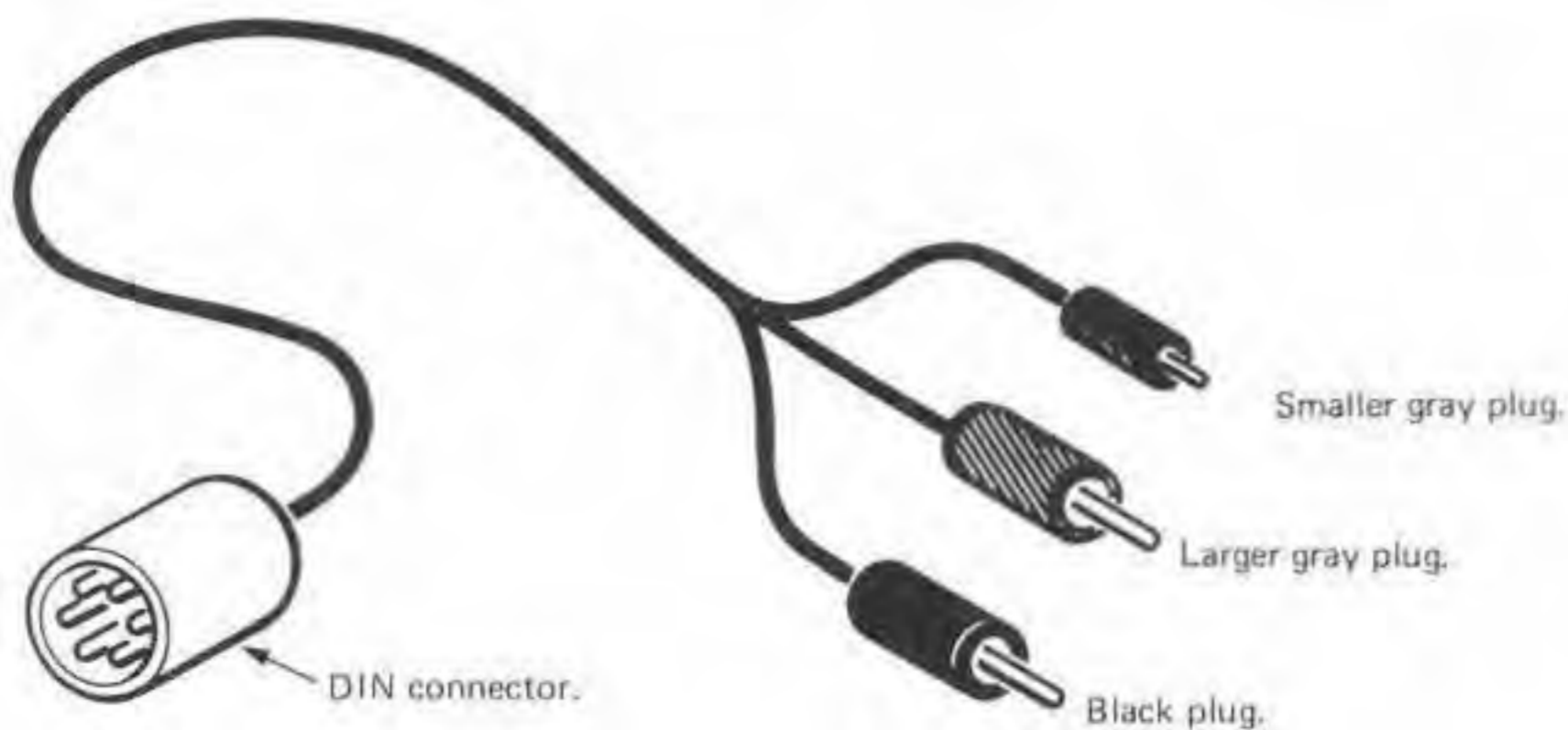
Color Computer programs can be permanently stored on tape cassettes. You can buy professionally written programs or you can record *your* programs on cassettes. You can also find program listings in magazines, type them into your Color Computer, and save them on cassettes.

We will describe how to use the Radio Shack CTR-80A cassette recorder (Radio Shack Catalog Number 26-1206) to load cassette programs into the Color Computer's memory or to save a program already in memory on a blank cassette.

The CTR-80A looks like this:



To connect the CTR-80A to the Color Computer, you need a cable that looks like this:

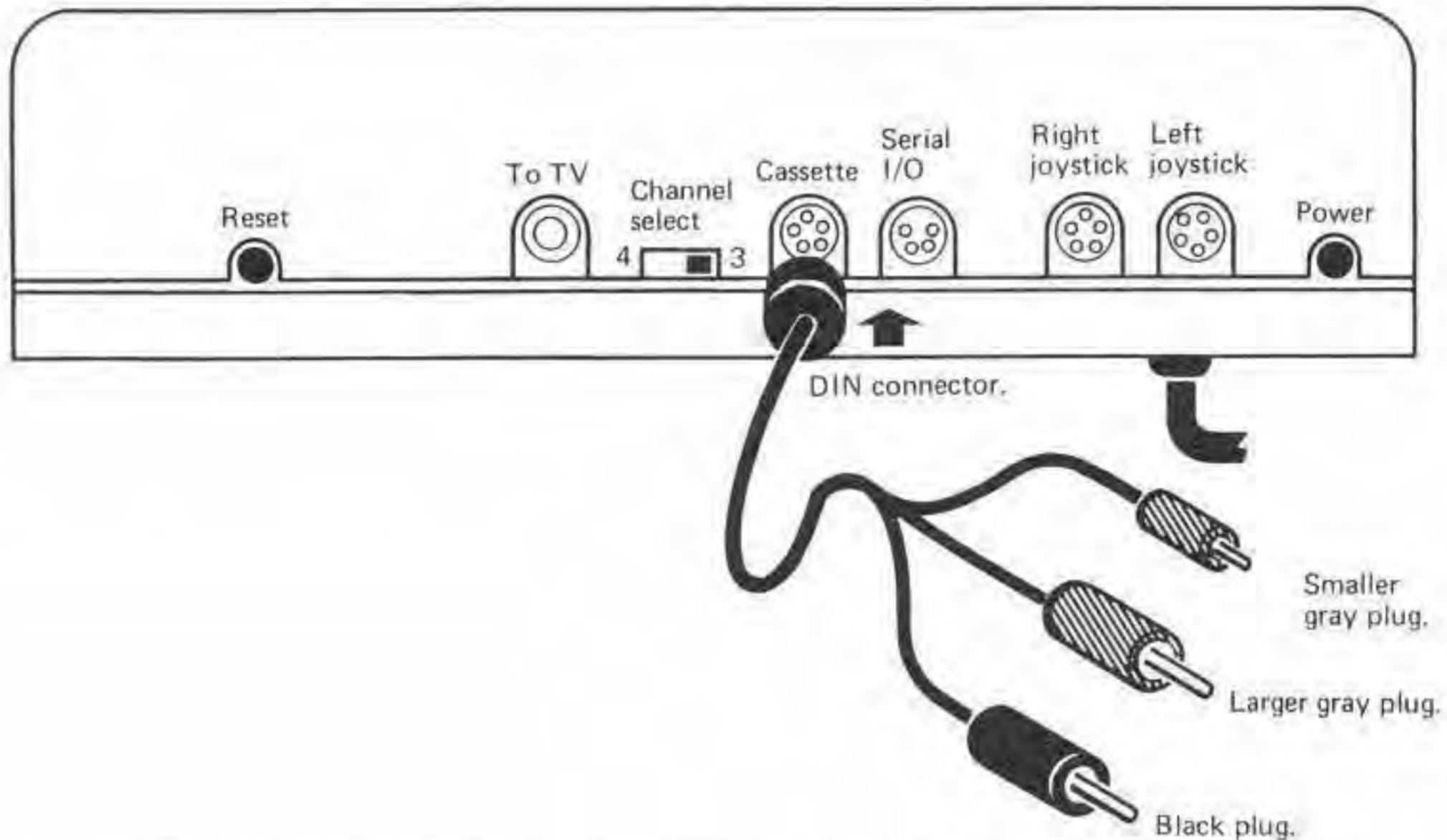


This cable comes with the CTR-80A. You can also buy it separately from Radio Shack (Catalog Number 26-3020).



Connect the cassette recorder to the Color Computer, as follows:

1. Gently and carefully plug the DIN connector into the CASSETTE jack on the rear of the Color Computer. Don't force it! Rotate the DIN connector until it lines up with the holes in the jack.



2. Plug the *black plug* into the EAR jack on the tape recorder. This connection is used to load a program from tape cassette to memory.
3. Plug the *larger gray plug* into the AUX jack on the tape recorder. This connection is used to record a program from memory to tape cassette.
4. Plug the *smaller gray plug* into the smaller MIC jack on the tape recorder. This connection is used by the computer to turn the tape recorder's motor on or off.
5. Of course, also plug the recorder's power cord into the recorder and into a 110-volt AC outlet.

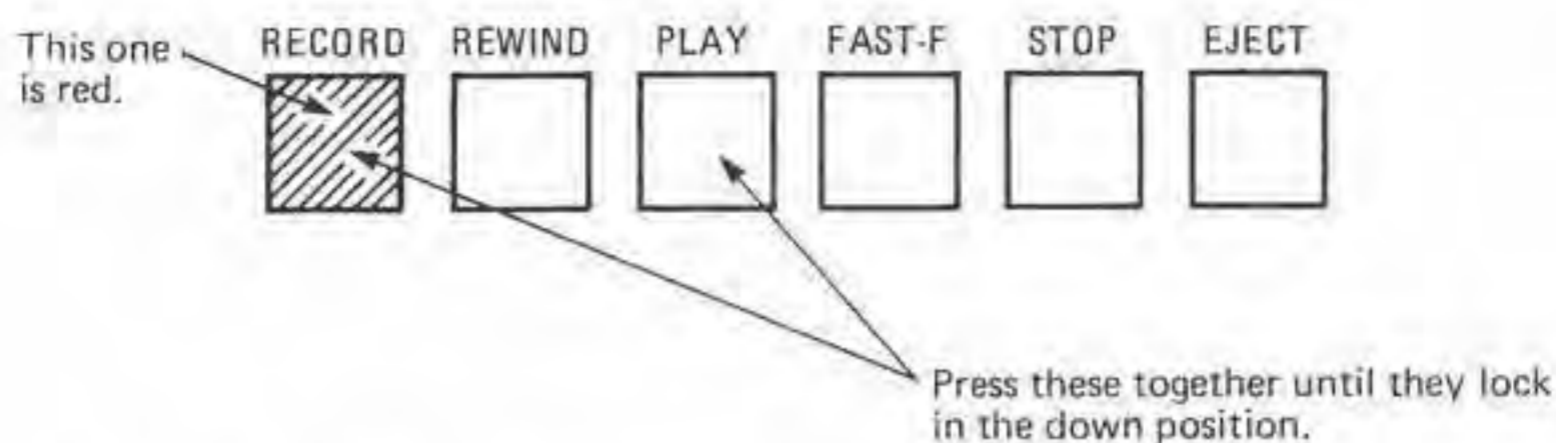
Everything connected? Yes? OK, now you are ready to save a program or load a program.

### **Saving a Program on Tape (CSAVE)**

Suppose there is a program in the Color Computer's memory and you want to record it on a cassette. Here's how:

1. Put a blank cassette in the cassette recorder.
2. Set the volume control on the recorder somewhere between 3 and 10. Try 5, or EXPERIMENT!

3. Press the recorder's **RECORD** and **PLAY** buttons at the same time until they lock.



4. Type **CSAVE** and press **ENTER**.
5. The computer will turn on the recorder's motor (you will see the tape move) and record the program.
6. When the program has been recorded, the computer stops with **OK** on the screen. The tape recorder also stops.

You may wish to make two or more copies on the same tape or on separate tapes, just in case you lose one or accidentally erase a tape.

Instead of typing **CSAVE** and pressing **ENTER**, you can give a program a name, as follows:

Type **CSAVE "name"** and press **ENTER**.

Any name you choose, up to eight characters.

For example:

```
CSAVE "MANDALA"
CSAVE "GAME #1"
CSAVE "MUSIC"
```

If a program has a name, you can tell the computer to search for it by name. This is especially useful if you record two or more programs on the same cassette.

## Loading a Program from Tape (**CLOAD**)

Here's how you load a previously recorded program from a cassette into memory:

1. Put the program cassette in the recorder and rewind it. When the tape is rewound, press the **STOP** button on the recorder.
2. Press the **PLAY** button until it locks.
3. Erase any program in memory. Type **NEW** and press the **ENTER** key.
4. Type **CLOAD** and press **ENTER**. The computer will load the first program it finds on the cassette.



5. When the program has been loaded, the computer stops with OK on the screen. The tape recorder also stops.

You can also tell the computer to search for and load a program by name.

4. Type CLOAD "name" and press **ENTER**.

Name of the program you want to load.

The computer will search for the program you named. While searching, it displays the letter "S" in the upper left part of the screen.

When the computer finds the program, it displays the letter "F" and the name of the program while it loads the program.

### **Skipping Past a Program (SKIPF)**

The SKIPF command tells the computer to search for a named program. When the computer finds the program, it skips to the end of the program *without* loading it into the computer.

Use SKIPF when you want to store a program on a tape that already contains a program or programs. Skip to the end of the *last* program on the tape. This positions the tape at the beginning of unused blank tape that can be used to store another program. For example:

1. Rewind the cassette.
2. Press the **PLAY** button until it locks.
3. Type SKIPF "name" and press **ENTER**.

The computer will search for the program you named. If it finds other programs along the way, it displays their names on the screen. If it finds the named program, it moves to the end of it, stops the recorder, and puts OK on the screen.

If the named program is not on the tape, the computer will read to the end of tape and then put an I/O ERROR on the screen. The screen will also contain the names of all programs the computer encountered while searching. Aha! Now you know what's on the tape. Their names are on the screen.

If you skip to the end of the last program on tape, and if there is enough unused tape remaining, you can save another program on the tape.

---



---

---

## APPENDIX C

# Error Messages

---

---

- /0 Division by zero. The computer can't divide by zero. Try this example: PRINT 2/0.
- A0 Attempt to open a data file that is already open.
- BS Bad subscript. A subscript is too large. If you do not DIMension an array, any subscript larger than 10 causes a BS error. For example: A(11). Use DIM to specify the largest subscript allowed in an array.
- CN Can't continue. This happens if a program has ended and you type CONT and press **ENTER**. A program that has ended can't be continued.
- DD Attempt to dimension an array more than once. An array can be dimensioned only once. For example, you cannot have DIM T(20) and DIM T(30) in the same program. Also, be careful that a GOTO does not cause the computer to try to execute the same DIM a second time. DIM should not be *inside* a loop.
- DN Device number error. The allowable device numbers are 0, -1, and -2. Any other number used in an OPEN, CLOSE, PRINT, or INPUT statement will cause a DN error.
- DS Direct statement. A data file contains a direct statement. This might happen if you load a program with no line numbers.
- FC Illegal Function Call. There are many ways to get this one! An FC error occurs when a number is too small or too large in a BASIC statement or function. Here are some examples:

|               |                                |
|---------------|--------------------------------|
| SOUND 89, 256 | SET(64,15)                     |
| SOUND 0, 100  | SET(15,32)                     |
| CHR\$(256)    | CHR\$(-1)                      |
| CLS -1        | CLS 9 ←This one is a surprise! |

A negative subscript also causes an FC error. For example, try  $A(-1) = 2$ .

- 
- FD Bad file data. This happens if you PRINT data to a file, or INPUT data from a file, and the type of data (numeric or string) does not match the type of variable in the PRINT or INPUT statement. For example, suppose the file contains *numeric* data. Then INPUT #-1, N\$ would cause an FD error because N\$ is a *string* variable.
- FM Bad file mode. This happens if you OPEN a file for OUTPUT(O), then try to INPUT data from that file. It also happens if you try to PRINT to a file that is OPEN for INPUT(I).
- ID Illegal direct statement. This happens when you use INPUT as a direct statement, without a line number. Example: Type INPUT A and press **ENTER**. You will get an ID error.
- IE Attempt to INPUT past end of file. Before using INPUT, use EOF to test for end of file. If you have reached end of file, CLOSE the file.
- IO Input/Output error. Often caused by trying to load a program or read a file from a bad cassette tape.
- LS String too long. A string can have up to 255 characters.
- NF NEXT without FOR. FOR and NEXT statements must occur in matching pairs. If the computer tries to execute a NEXT statement without a matching FOR statement, an NF error occurs. What about FOR without NEXT? Experiment—try this program:

```
10 CLS
20 FOR X=1 TO 5
30 PRINT X
40 END
```

- NO File not open. You tried to PRINT to, or INPUT from, a data file without first OPENing the file.
- OD Out of data. The computer tried to READ data from a DATA statement, but all the data had already been read. Perhaps someone failed to supply a DATA statement or left out some of the data.
- OM Out of memory. All of memory has been used or reserved for program and data. Reduce the demand on memory or buy more memory!
- OS Out of string space. The computer automatically reserves 200 characters for string space. If you use more, you will get an OS error. Use CLEAR to reserve more or less string space.
- OV Overflow. A number is too large for the computer to handle.
- RG RETURN without GOSUB. The computer tried to execute a RETURN statement without having previously executed a GOSUB statement.
-

- SN Syntax error. Probably the most popular error! It means that the computer does not understand what you want or that what you want is not allowed. Check for a misspelled word, incorrect punctuation, unmatched parenthesis, illegal character, or use of a reserved word as a variable.
  - ST String expression too complex. Break it into two or more simpler expressions.
  - TM Type mismatch. You will get this if you try to assign a string to a numeric variable or a number to a string variable. For example, `X = "KARL"` or `N$ = 3`. A TM error also occurs when you enter a string in response to an `INPUT` statement with a numeric variable or if you use the wrong type of variable or value in a function. For example: `ASC(65)` or `CHR$("A")`.
  - UL Undefined line. If you try to `GOTO` or `GOSUB` to a nonexistent line number, you will get a UL error. It can also occur with `ON . . . GOTO . . .`, `ON . . . GOSUB . . .`, and `IF` statements.
-



---

---

## APPENDIX D

# Arithmetic

---

---

Use the following symbols to tell the Color Computer to do arithmetic.

| <u>Operation</u> | <u>Symbol</u> | <u>Example</u> | <u>Result</u> |
|------------------|---------------|----------------|---------------|
| Addition         | +             | $3 + 4$        | 7             |
| Subtraction      | -             | $3 - 4$        | -1            |
| Multiplication   | *             | $3 * 4$        | 12            |
| Division         | /             | $3 / 4$        | .75           |

If an expression has two or more operations, they are done in the same order used in paper-and-pencil arithmetic or algebra. Example:

$$2 + 3 + 4 = 5 + 4 = 9$$

$$2 + 3 - 4 = 5 - 4 = 1$$

$$2 - 3 + 4 = -1 + 4 = 3$$

$$2 - 3 - 4 = -1 - 4 = -5$$

$$2 * 3 * 4 = 6 * 4 = 24$$

$$2 * 3 / 4 = 6 / 4 = 1.5$$

$$2 / 3 * 4 = .666666667 * 4 = 2.666666667$$

$$2 / 3 / 4 = .666666667 / 4 = .166666667$$

Additions and subtractions  
are done in left-to-right order.

Multiplications and  
divisions are done in  
left-to-right order.

Results are rounded to nine digits.

Multiplications or divisions are done before additions or subtractions.

$$2 * 3 + 4 = 6 + 4 = 10$$

$$2 + 3 * 4 = 2 + 12 = 14$$

$$2 + 3 / 4 = 2 + .75 = 2.75$$

$$2 * 3 + 4 * 5 = 6 + 20 = 26$$

$$2 * 3 + 4 / 5 = 6 + .8 = 6.8$$

$$2 / 3 + 4 * 5 = .666666667 + 20 = 20.66666667$$

$$2 / 3 + 4 / 5 = .666666667 + .8 = 1.466666667$$

Operations within parentheses are performed first. If parentheses are nested (one pair within another pair), operations within the innermost pair are performed first, then evaluation proceeds to the next pair out.

$$2*(3 + 4) = 2*7 = 14$$

$$2/(3*4) = 2/12 = .166666667$$

$$2 - (3 + 4) = 2 - 7 = -5$$

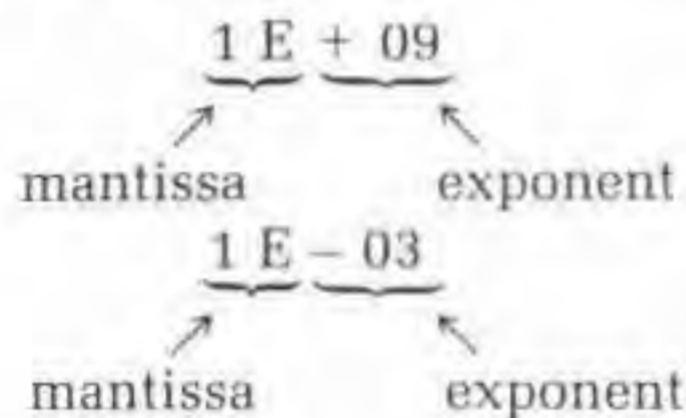
$$2*(3 + 4*(5 + 6)) = 2*(3 + 4*11) = 2*(3 + 44) = 2*47 = 94$$

$$(2 + 3)*(4 + 5) = 5*9 = 45$$

Numbers larger than 999999999 or smaller than .01 are shown in *floating point notation*. Floating point notation is similar to *scientific notation* used in math, science, and engineering books.

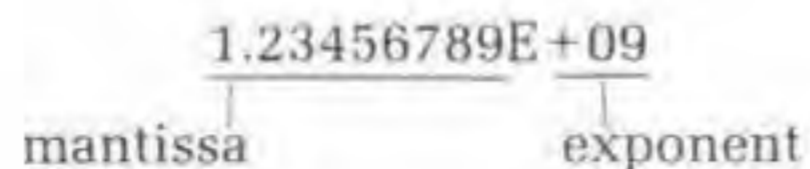
| Number       | Scientific Notation          | Floating Point Notation |
|--------------|------------------------------|-------------------------|
| 1000000000   | $1 \times 10^9$              | 1E+09                   |
| 2000000000   | $2 \times 10^9$              | 2E+09                   |
| 10000000000  | $1 \times 10^{10}$           | 1E+10                   |
| 20000000000  | $2 \times 10^{10}$           | 2E+10                   |
| .001         | $1 \times 10^{-3}$           | 1E-03                   |
| .002         | $2 \times 10^{-3}$           | 2E-03                   |
| .0001        | $1 \times 10^{-4}$           | 1E-04                   |
| .0002        | $2 \times 10^{-4}$           | 2E-04                   |
| 1234567890   | $1.23456789 \times 10^9$     | 1.23456789E+09          |
| .00123456789 | $1.23456789 \times 10^{-11}$ | 1.23456789E-11          |

Floating point is simply a shorthand way of expressing very large or quite small numbers. In floating point, a number is represented by a *mantissa* and an *exponent*.



The mantissa and exponent are separated by the letter E.

Numbers in the Color Computer can have a mantissa with up to nine digits.



EXPERIMENT! Use this program:

```

10 CLS
20 INPUT "NUMBER, PLEASE"; N
30 PRINT N
40 PRINT
50 GOTO 20

```

Try these numbers:

Volume of the Earth in bushels: 31 708 000 000 000 000 000 000

Speed of a snail in kilometers per second: .000015

Mass of the hydrogen atom in kilograms:

.000 000 000 000 000 000 001 67

Mass of the Earth in kilograms:

5 980 000 000 000 000 000 000 000

---



---

---

## APPENDIX E

# Reserved Words

---

---

The following words are reserved words in *Extended Color Computer BASIC*. A reserved word may not be used as a variable or as the beginning of a variable in *Extended Color BASIC*. For example, you may not use TO, TON, TOE, or TONE as a variable, because TO is a reserved word.

|        |         |        |          |        |
|--------|---------|--------|----------|--------|
| ABS    | END     | MEM    | PUT      | THEN   |
| AND    | EOF     | MID\$  | READ     | TIMER  |
| ASC    | EXEC    | MOTOR  | REM      | TO     |
| ATN    | EXP     | NEW    | RENUM    | TROFF  |
| AUDIO  | FIX     | NEXT   | RESET    | TRON   |
| CHR\$  | FN      | NOT    | RESTORE  | USING  |
| CIRCLE | FOR     | OFF    | RETURN   | USR    |
| CLEAR  | GET     | ON     | RIGHT\$  | VAL    |
| CLOAD  | GOSUB   | OPEN   | RND      | VARPTR |
| CLOADM | GOTO    | OR     | RUN      |        |
| CLOSE  | HEX\$   | PAINT  | SCREEN   |        |
| CLS    | IF      | PCLEAR | SET      |        |
| COLOR  | INKEY\$ | PCLS   | SGN      |        |
| CONT   | INPUT   | PCOPY  | SIN      |        |
| COS    | INSTR   | PEEK   | SKIPF    |        |
| CSAVE  | INT     | PLAY   | SOUND    |        |
| DATA   | JOYSTK  | PMODE  | SQR      |        |
| DEF    | LEFT\$  | POINT  | STEP     |        |
| DEL    | LEN     | POKE   | STOP     |        |
| DIM    | LET     | POS    | STR\$    |        |
| DLOAD  | LINE    | PPOINT | STRING\$ |        |
| DRAW*  | LIST    | PRESET | SUB      |        |
| EDIT   | LLIST   | PRINT  | TAB      |        |
| ELSE   | LOG     | PSET   | TAN      |        |

Remember, the above words are reserved in *Extended Color BASIC*. Many of them will work as variables in minimum *Color BASIC* on the unextended *Color Computer*. We suggest you *don't* use them. *Color BASIC* programs, as described in this book, will work on the *Extended Color BASIC* computer if you *don't* use the above words as variables.

---



---

## APPENDIX F

# Screen Maps

---



---

The screen is your palette; the screen is your page. Here are worksheets for designing stuff to appear on the screen, using PRINT@ or SET and RESET.

First, a screen map for people using PRINT@

|      |    | COLUMNS |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|------|----|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--|--|--|
|      |    | 0       | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |  |  |  |
| ROWS | 0  |         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 1  | 32      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 2  | 64      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 3  | 96      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 4  | 128     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 5  | 160     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 6  | 192     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 7  | 224     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 8  | 256     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 9  | 288     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 10 | 320     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 11 | 352     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 12 | 384     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 13 | 416     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 14 | 448     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |
|      | 15 | 480     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |  |  |  |

Our PRINT@ screen map shows both left-of-screen PRINT@ positions and PRINT rows. For purposes of PRINTing, the screen is divided into 16 rows, numbered 0 to 15. Each row has 32 columns, numbered 0 to 31.





Each screen PRINT position contains four (4) SET positions, as shown by the following screen map.

| PRINT | 0     | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |  |
|-------|-------|----|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
|       | SET 0 | 2  | 4 | 6 | 8 | 0 | 1 | 1 | 1 | 1 | 2  | 2  | 2  | 2  | 2  | 3  | 3  | 3  | 3  | 3  | 4  | 4  | 4  | 4  | 4  | 5  | 5  | 5  | 5  | 6  | 6  |    |  |  |
| 0     | 0     | 0  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 1     | 32    | 2  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 2     | 64    | 4  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 3     | 96    | 6  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 4     | 128   | 8  |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 5     | 160   | 10 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 6     | 192   | 12 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 7     | 224   | 14 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 8     | 256   | 16 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 9     | 288   | 18 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 10    | 320   | 20 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 11    | 352   | 22 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 12    | 384   | 24 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 13    | 416   | 26 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 14    | 448   | 28 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| 15    | 480   | 30 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
|       |       | 31 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |

You may copy the screen maps in this appendix for your personal use. Also, ask at your Radio Shack store for larger screen maps.

---

---

## APPENDIX G

# Sound and Music

---

---

Using the SOUND statement, you can tell the Color Computer to make sounds that are close to (but sometimes not exactly) tones on a piano keyboard. Of course, the *quality* of the sound will be different. Middle C on the Color Computer won't sound like Middle C on the piano. However, the *frequency*, or *pitch*, of the tone will be close to the same tone on a well-tuned piano.

SOUND           ,            
                  ↑                  ↑  
          tone (1 to 255)    duration (1 to 255)  
Example: SOUND 89, 10

The duration number determines the length of time the tone is played, as follows:

| <u>Duration Number</u> | <u>Time(Second)</u> |                  |
|------------------------|---------------------|------------------|
| 1                      | 6/100               |                  |
| 2                      | 12/100              |                  |
| 3                      | 18/100              |                  |
| .                      | .                   |                  |
| .                      | .                   |                  |
| .                      | .                   |                  |
| 16                     | 96/100              | (about 1 second) |
| .                      | .                   |                  |
| .                      | .                   |                  |
| .                      | .                   |                  |
| 255                    | 1530/100            | (15.30 seconds)  |





---

---

## APPENDIX H

# Color Codes and Graphics Characters

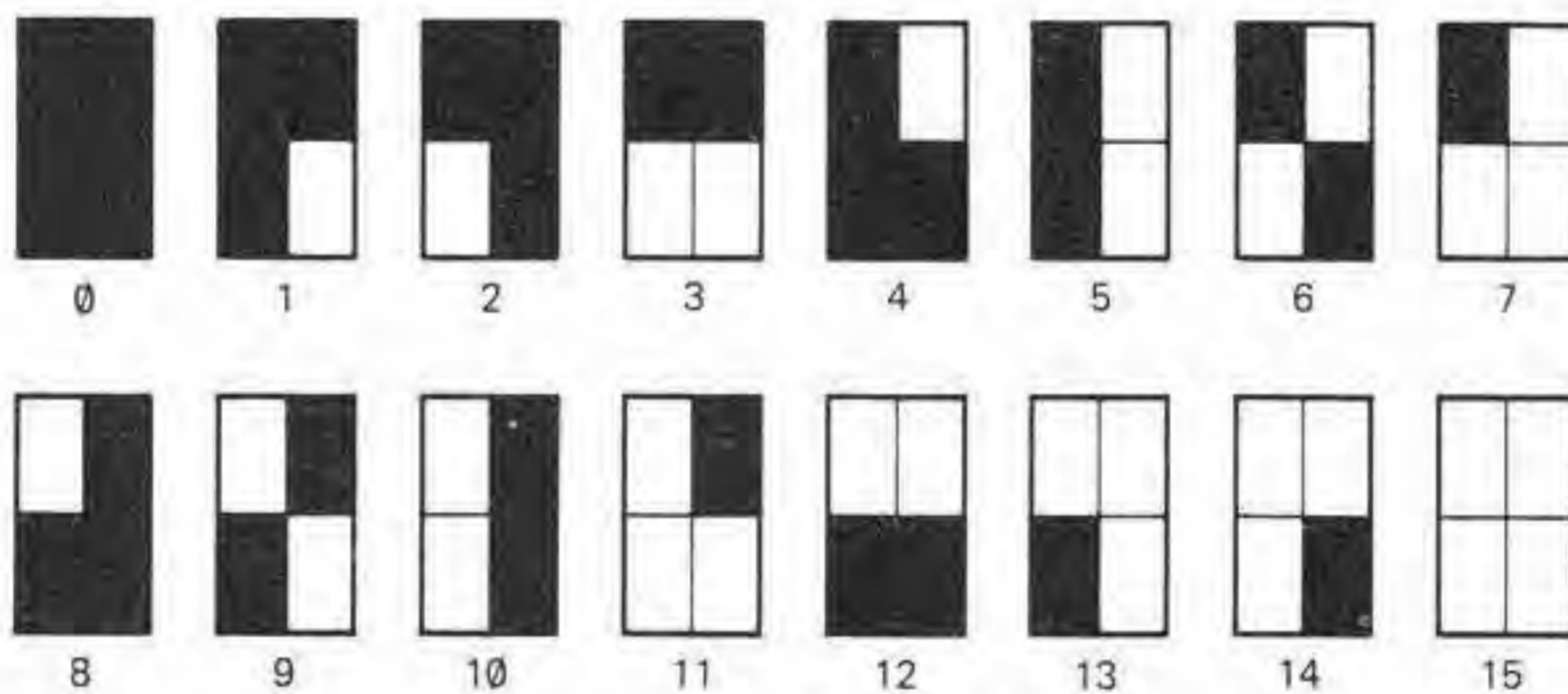
---

---

You can put nine colors (including black) on the screen. Colors are numbered 0 to 8, as follows:

|   |        |   |         |
|---|--------|---|---------|
| 0 | Black  |   |         |
| 1 | Green  |   |         |
| 2 | Yellow |   |         |
| 3 | Blue   |   |         |
| 4 | Red    |   |         |
|   |        | 5 | Buff    |
|   |        | 6 | Cyan    |
|   |        | 7 | Magenta |
|   |        | 8 | Orange  |

ASCII codes 128 to 255 are graphics characters. Each graphics character occupies one print position. A graphics character is a simple shape in black plus one color. There are 16 distinct shapes, numbered 0 to 15, shown below in black and white. Substitute any color (1 to 8) for white.



Color the above 16 characters green. These are ASCII graphics characters 128 to 143. Color the above characters yellow. You now have graphics characters 144 to 159—and so on, as shown by the following table.

---

| Color | ASCII Codes |
|-------|-------------|
|-------|-------------|

|         |            |
|---------|------------|
| Green   | 128 to 143 |
| Yellow  | 144 to 159 |
| Blue    | 160 to 175 |
| Red     | 176 to 191 |
| Buff    | 192 to 207 |
| Cyan    | 208 to 223 |
| Magenta | 224 to 239 |
| Orange  | 240 to 255 |

Or, compute the ASCII code, as follows:

1. Pick a SHAPE, 0 to 15.
2. Pick a KOLOR, 1 to 8.

Compute the ASCII Graphics Code (GC), as follows:

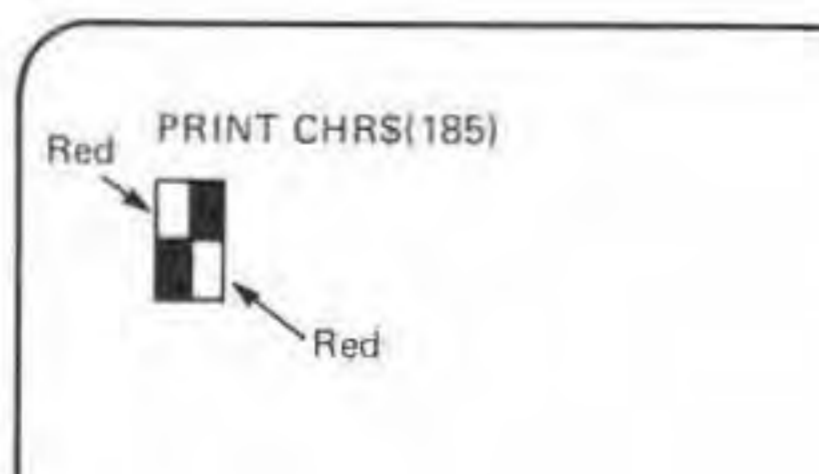
$$GS = 128 + \text{SHAPE} + 16 * (\text{KOLOR} - 1)$$

For example,

Pick a SHAPE.  This is SHAPE = 9.

Pick a KOLOR. Choose red. KOLOR = 4.  
 $GC = 128 + 9 + 16 * (4 - 1) = 185$

Try it. Type PRINT CHR\$(185) and press ENTER.



---



---

# APPENDIX I

## ASCII Codes

---



---

ASCII means "American Standard Code for Information Interchange." An ASCII code is a whole number, 0 to 255.

Each Color Computer keyboard character has a corresponding ASCII code, as follows:

| KEYBOARD CHARACTER | ASCII CODE | KEYBOARD CHARACTER | ASCII CODE |
|--------------------|------------|--------------------|------------|
| SPACE BAR          | 32         | A                  | 65         |
| !                  | 33         | B                  | 66         |
| "                  | 34         | C                  | 67         |
| #                  | 35         | D                  | 68         |
| \$                 | 36         | E                  | 69         |
| %                  | 37         | F                  | 70         |
| &                  | 38         | G                  | 71         |
| '                  | 39         | H                  | 72         |
| (                  | 40         | I                  | 73         |
| )                  | 41         | J                  | 74         |
| *                  | 42         | K                  | 75         |
| +                  | 43         | L                  | 76         |
| ,                  | 44         | M                  | 77         |
| -                  | 45         | N                  | 78         |
| .                  | 46         | O                  | 79         |
| /                  | 47         | P                  | 80         |
| 0                  | 48         | Q                  | 81         |
| 1                  | 49         | R                  | 82         |
| 2                  | 50         | S                  | 83         |
| 3                  | 51         | T                  | 84         |
| 4                  | 52         | U                  | 85         |
| 5                  | 53         | V                  | 86         |
| 6                  | 54         | W                  | 87         |
| 7                  | 55         | X                  | 88         |
| 8                  | 56         | Y                  | 89         |
| 9                  | 57         | Z                  | 90         |
| :                  | 58         | ␣                  | 94         |
| ;                  | 59         | ␣                  | 10         |
| <                  | 60         | ␣                  | 8          |
| =                  | 61         | ␣                  | 9          |
| >                  | 62         | <b>BREAK</b>       | 03         |
| ?                  | 63         | <b>CLEAR</b>       | 12         |
| @                  | 64         | <b>ENTER</b>       | 13         |



You cannot display lower-case letters on the screen, but you can print them on some printers.\* To get to lower-case mode, press the **SHIFT** and **0** keys together. On the screen, "lower-case" letters appear as upper-case letters in reverse color, green on black.

To get back to normal mode, again press **SHIFT** and **0** together.

| Lower-Case Letter | ASCII Code | Lower-Case Letter | ASCII Code |
|-------------------|------------|-------------------|------------|
| a                 | 97         | n                 | 110        |
| b                 | 98         | o                 | 111        |
| c                 | 99         | p                 | 112        |
| d                 | 100        | q                 | 113        |
| e                 | 101        | r                 | 114        |
| f                 | 102        | s                 | 115        |
| g                 | 103        | t                 | 116        |
| h                 | 104        | u                 | 117        |
| i                 | 105        | v                 | 118        |
| j                 | 106        | w                 | 119        |
| k                 | 107        | x                 | 120        |
| l                 | 108        | y                 | 121        |
| m                 | 109        | z                 | 122        |

Try this: Type `PRINT CHR$(97)` and press **ENTER**.

You will see a green A on a black background on the screen. However, if you are printing on a printer that allows lower-case, use `PRINT#-2,` (instead of `PRINT`) and you will see a lower-case *a* printed.

ASCII codes 0 to 31 either are not used in Level I Color BASIC or have very special meanings. For example, use `PRINT CHR$(30)` to erase to the end of a line.

ASCII codes 128 to 255 are graphics characters. See Appendix H, Color Codes and Graphics Characters.

EXPERIMENT! Try `PRINT CHR$(     )`

↑  
ASCII code—you choose it.

\*See the Radio Shack catalog for information on printers.

---

---

## APPENDIX J

# Joysticks

---

---

If you have joysticks (Radio Shack Catalog Number 26-3008), plug them into the joystick connections in the back of the Color Computer. Then put the RIGHT JOYSTICK on your right and the LEFT JOYSTICK on your left as you face the computer.

Move the RIGHT JOYSTICK left, right, forward, and backward. Nothing happens. Aha! You need a program to detect the movement of the joysticks. Enter this little program:

```
10 CLS
20 PRINT @16, JOYSTK(0);
30 GOTO 20
```

Type RUN, watch the center of the top row of the screen, and move the RIGHT JOYSTICK.

Move the RIGHT JOYSTICK all the way to the left, then all the way to the right. You will see joystick numbers 0 (left) to 63 (right). Center the RIGHT JOYSTICK, then move it straight forward and backward. The number doesn't change (unless you lean a little left or right).

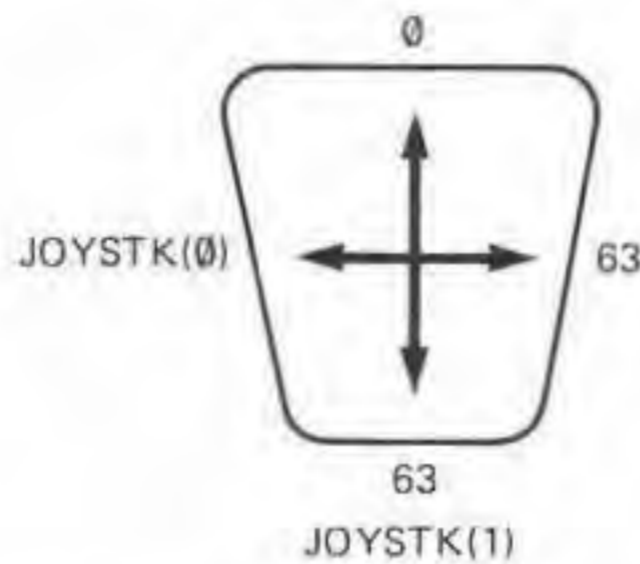
Now try this program:

```
10 CLS
20 PRINT @16, JOYSTK(0);
30 PRINT @24, JOYSTK(1);
40 GOTO 20
```

RUN the program. You will see that left-right movement changes the number at screen position 16, and forward-backward movement changes the number at screen position 24. Diagonal movement changes both numbers.

- JOYSTK(0) reads the left-right position of the RIGHT JOYSTICK. Possible values are 0 (left) to 63 (right).
- JOYSTK(1) reads the forward-backward position of the RIGHT JOYSTICK. Possible values are 0 (forward) to 63 (backward).

Like this:



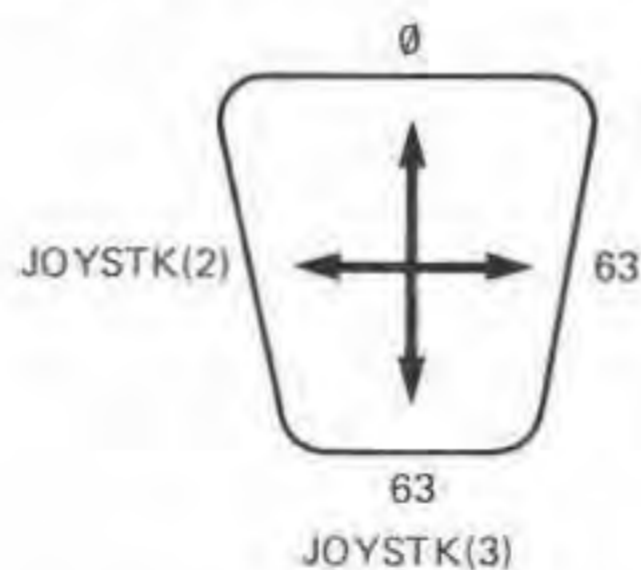
Now use the following program to try both joysticks.

```

10 CLS
20 PRINT @0, JOYSTK(2); Left
30 PRINT @8, JOYSTK(3); joystick.
40 PRINT @16, JOYSTK(0); Right
50 PRINT @24, JOYSTK(1); joystick.
60 GOTO 20

```

Yes, the LEFT JOYSTICK works like this:



**IMPORTANT NOTICE!** If you use joysticks, you *must* have the computer read JOYSTK(0), even if you aren't going to do anything with it. For example, suppose you are using *only* the LEFT JOYSTICK, using the following program.

```

10 CLS
30 PRINT @0, JOYSTK(2);
40 PRINT @8, JOYSTK(3);
50 GOTO 30

```

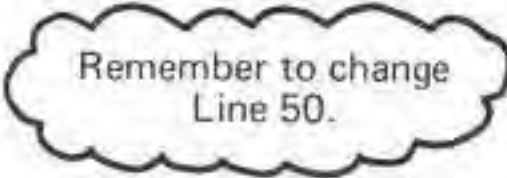


Try it. Wiggle the LEFT JOYSTICK. Nothing happens. So change the program as follows:

```

10 CLS
20 JS = JOYSTK(0)
30 PRINT @0, JOYSTK(2);
40 PRINT @8, JOYSTK(3);
50 GOTO 20

```



Remember to change  
Line 50.

Line 20 tells the computer to read JOYSTK(0) and assign the value of JS. This enables the computer to go on and read the values of JOYSTK(2) and JOYSTK(3).

Experiment. Try these programs:

```

10 CLS
20 SP = JOYSTK(0)
40 PRINT @SP, "*";
50 Z = 10
60 FOR K=1 TO Z: NEXT K
70 GOTO 10

```

---

```

10 CLS 0
20 OVER = JOYSTK(0)
40 SET (OVER, 15, 8)
50 Z = 10
60 FOR K=1 TO Z: NEXT K
70 GOTO 10

```

---

```

10 CLS 0
20 OVER = JOYSTK(0)
30 DOWN = INT(JOYSTK(1)/2)
40 SET(OVER, DOWN, 8)
50 Z = 10
60 FOR K=1 TO Z: NEXT K
70 GOTO 10

```

---

```

10 CLS
20 COL = INT(JOYSTK(0)/2)
30 ROW = INT(JOYSTK(1)/14)
40 PRINT @32*ROW + COL, "*";
50 Z = 10
60 FOR K=1 TO Z: NEXT K
70 GOTO 10

```

---

In any of the above programs, change line 70 as follows:

```
70 GOTO 20
```

Are you curious about the buttons on the joysticks?



Button settings (on or off) are detected by PEEKing into the computer's memory at location 65280. Use the following program.

```
10 CLS
20 BUTTON = PEEK(65280)
30 PRINT @237, BUTTON
40 GOTO 20
```

Move the RIGHT JOYSTICK to its *rightmost* position and RUN the program. You will see:

```
255
```

```
Number in memory
location 65280.
```

Move the RIGHT JOYSTICK to its *leftmost* position. The number will change to 127.

Press the button on the RIGHT JOYSTICK. The number changes from 127 to 126. Release the button.

Press the button on the LEFT JOYSTICK. The number changes to 125. Release the button.

Press *both* buttons at the same time. The number is now 124.

Move the RIGHT JOYSTICK to the right and repeat all the above. The results are summarized in the following table.

| PEEK(65280)                        | RIGHT JOYSTICK<br>IN LEFT POSITION | RIGHT JOYSTICK<br>IN RIGHT POSITION |
|------------------------------------|------------------------------------|-------------------------------------|
| BOTH BUTTONS OFF                   | 127                                | 255                                 |
| LEFT BUTTON OFF<br>RIGHT BUTTON ON | 126                                | 254                                 |
| LEFT BUTTON ON<br>RIGHT BUTTON OFF | 125                                | 253                                 |
| BOTH BUTTONS ON                    | 124                                | 252                                 |

Experiment. Scribble on the screen with the following program, using the RIGHT JOYSTICK. Press the RIGHT BUTTON to see what happens:

```
10 CLS 0
20 OVER = JOYSTK(0)
30 DOWN = INT(JOYSTK(1)/2)
40 SET(OVER, DOWN, 8)
50 Z = 10
60 FOR K=1 TO Z: NEXT K
70 BUTTON = PEEK(65280)
80 IF BUTTON=126 THEN 10
90 IF BUTTON=254 THEN 10 ELSE 20
```



---

---

## APPENDIX K

# Look Here First

---

---

We hope you looked here soon after you first cracked this book. You can teach yourself how to use, program, and enjoy computers better, and have more fun, if you use more than one source of information and inspiration.

As you progress from beginner to novice to intermediate to advanced to ???, try some of these.

- *Getting Started with Color BASIC*, by Donna Greaves Smith; designed and illustrated by Linda Yakel. Radio Shack Catalog Number 26-3191.

*Beginner's Guide to the Color Computer*. It comes with the Color Computer or can be bought separately from Radio Shack. If you are a beginner, we recommend that you use both our book and Radio Shack's. They complement each other. By using both, you will learn much more than if you used only one.

- *Going Ahead with Extended Color BASIC*, by Jonathan Erickson; designed and illustrated by Linda Yakel. Radio Shack Catalog Number 26-3192.

An intermediate-to-advanced-level book on Extended Color BASIC. Continue here after you finish our *TRS-80 Color BASIC* or Radio Shack's *Getting Started with Color BASIC*.

- *TRS-80 Color Computer Graphics*, by Don Inman. (Reston Publishing Company, 11480 Sunset Hills Road, Reston, VA 22090.) Intermediate to advanced stuff. Don Inman is your guide into the mysteries of high resolution color graphics and animation in Extended Color BASIC. We suggest you read *Color Computer Graphics* and Radio Shack's *Going Ahead with Extended Color BASIC* together.

- *Color Computer/6809 Machine Language*, by Don Inman and Kurt Inman. (Reston Publishing Company, 11480 Sunset Hills Road, Reston, VA 22090.)

If you know a little Color Computer BASIC and are curious about what *really* happens inside the computer, try this book. When you finish, the *full power* of the Color Computer is yours. Beware! It's not as easy as BASIC. We suggest you complete *TRS-80 Color Computer BASIC* and spend some time with Don's *Color Computer Graphics* before you try this one.

- *Dymax Gazette*, P.O. Box 310, Menlo Park, CA 94025. \$12/year (6 issues).

A periodical for and by authors and readers of books (such as this one) by Dymax authors. The *Gazette* contains corrections, letters from readers, problems and solutions, tutorials, news, and stuff that will appear in future books.

- *Color Computer News*, P.O. Box 1192, Muskegon, MI 49443. \$9/year (6 issues).

A periodical devoted entirely to the Color Computer! Tutorials, articles, ads, reviews, and a place to write to when you want help.

- *Popular Computing*, P.O. Box 397, Hancock, NH 03449. \$15/year (12 issues).

The number-one best computer magazine for beginners. Start here and grow. If you like this book, watch for "DragonSmoke" and "My Computer Likes Me" in *Popular Computing*. The regular series "My Computer Likes Me" is especially for people who want to help kids learn to use, program, and enjoy computers.

We have a much longer list, but it changes too rapidly to put into a book. If you want the current list, send a self-addressed stamped envelope to:

DragonSmoke  
*Dymax Gazette*  
P.O. Box 310  
Menlo Park, CA 94025

---



# INDEX

- ABS.** 219, 323  
Absolute value, 219  
Addition (+), 357  
Array, 273, 275  
Arithmetic (+, -, \*, /), 357  
ARROWS program, 129  
**ASC.** 239, 324  
ASCII, 239, 248, 254, 368  
ASCII CODES FOR  
KEYBOARD CHARACTERS program, 249  
**AUDIO.** 324
- Backspace key (←), 19, 39  
**BASIC.** 1, 3  
BASIC program, 1, 4, 35  
Basic Role Playing, 193  
Big Dipper, 151  
BLINK GRAPHICS CHARACTER program, 176  
BLINK VALUE OF AS program, 136  
BLIP BLINKER program, 154  
BLIPS ON THE QUARTER-SCREEN program, 202  
Boxes, number, 59, 60  
Boxes, string, 85, 86  
BREAK key, 41, 42, 49  
BS ERROR, 281
- Cassette recorder, 1, 7, 350  
Cassette, tape, 7, 351  
Catenate strings, 262  
Chip, 2  
Chip, memory, 5  
Chip, microcomputer, 3  
**CHRS.** 166, 239, 250, 324  
**CLEAR.** 239, 324  
CLEAR key, 13, 15, 20  
**CLOAD.** 325, 352  
**CLOADM.** 325  
CLOCK MUSIC programs, 312  
**CLOSE.** 326  
**CLS.** 20, 35, 326  
COIN FLIP GAME program, 211  
COIN FLIPPER AND COUNTER program, 285, 298  
Colon (:), 23, 24  
Color BASIC, 1, 3, 4  
Color Computer, 1, 2  
Color Computer News, 376  
Colors, on screen, 20, 21, 367  
COLOR FROM STRINGS program, 255  
COLOR STRIPES, THE EASY WAY program, 156  
Color TV, 2  
Comma [, 78, 107, 125  
Compare strings, 239  
COMPUTER AS MAD ARTIST program, 217  
Computerese, 1, 8  
Computer language, 3  
Computer program, 4  
The Computing Teacher, 33, 299  
Condition, in IF statement, 206, 208  
CONSTELLATION PROGRAM, 104
- CONT.** 326  
CREATE A CHARACTER FOR D&D, RUNEQUEST, or T&T program, 257  
CREATE A CHARACTER program, 195, 198  
CREATE A RUNEQUEST CHARACTER program, 243  
CREATE A T&T CHARACTER program, 240  
**CSAVE.** 326, 351  
**CSAVEM.** 327  
Cursor, 12, 13
- DATA.** 59, 77, 105, 327  
D&D (Dungeons & Dragons), 177, 193, 256  
Delete a statement, 11, 35, 50  
DIE ROLLER AND COUNTER program, 286  
**DIM.** 273, 281, 327  
Direct statement, 11, 16, 20, 36  
Division (/), 357  
Dollar sign (\$), 88  
DO, RE, MI PROGRAM, 53, 107, 109  
DRACO ON A BLACK SCREEN program, 153  
Dungeons & Dragons, 177, 193, 256  
Duration, 25, 27, 364  
Dymax Gazette, 33, 176, 299, 376
- Edit a statement, 35  
**ELSE.** 205, 331  
Empty string, 225  
**END.** 328  
End of data flag, 240  
End of File (EOF), 328  
Entering a program, 35  
ENTER key, 11, 14  
**EOF.** 328  
Error messages, 11, 14, 354  
Errors, typing, 11, 18  
**EXEC.** 329  
EXPERIMENT WITH INKEYS program, 224  
EXPERIMENT: ROWS, COLUMNS, program, 134  
EXPERIMENT WITH SET program, 148  
EXOTIC NAMES IN FANTASY ADVENTURELAND, 269  
Exponents, 358  
Extended BASIC, 1  
Fantasy role playing games, 193  
1ST GRAND FINALE, CH. 5 program, 101  
1ST NUMBER GAME program, 209  
Flag, end of data, 240  
Floating point numbers, 358  
**FOR.** 85, 96, 102, 329  
**FOR-NEXT** loop, 96, 99, 329  
**FOR-NEXT-STEP** loop, 102, 103, 329  
FOUR CATS CHASING problem, 320
- GAMEMASTER'S DICE problem, 321  
**GOSUB.** 119, 330  
GO TEAM GO! program, 118  
**GOTO (GO TO).** 35, 36, 330  
Graphics character, 165, 168, 170, 366  
GRAPHICS CHARACTERS EVERYWHERE program, 203  
GRAPHICS CHARACTERS programs, 167, 169  
GUESS MY LETTER program, 246  
GUESS MY TONE program, 214  
GUESS MY WORD program, 244, 292  
**IF.** 205, 331  
**IF . . . THEN . . .** 206, 331  
**IF . . . THEN . . . ELSE.** 227, 331  
Immediate statement, 11, 36  
INDEFATIGABLE ARROW program, 129  
**INKEYS.** 205, 223, 224, 228, 231  
**INPUT.** 59, 64, 92, 94, 331  
**INPUT "string";** 69, 70, 331  
**INT.** 299, 332  
INTERGALACTIC BROADCASTING COMPANY program, 47
- Join strings, 262  
Joystick, 1, 8, 370  
**JOYSTK.** 333, 371
- Keyboard, 1, 6, 13  
KEYBOARD COLOR TONES program, 76  
KEYBOARD TONES program, 74
- LEFTS.** 239, 261, 333  
**LEN.** 239, 252, 334  
Line number, 36, 44  
**LIST.** 35, 40, 43, 334  
LISTEN AND GUESS MY NUMBER program, 219  
**LLIST.** 334  
LOST SUBMARINE program, 316
- MAD ARTIST, 217, 221  
MAD ARTIST MEANDERS program, 221  
MAKE A SENTENCE program, 262  
MANDALA, EVER CHANGING program, 187, 223  
mantissa, 358  
MEANDERING NAME program, 183  
**MEM.** 334  
Memory, 1, 5, 6  
Memory, random access, (RAM), 1, 6  
Memory, read only (ROM), 1, 5, 6  
Memory read/write (RAM), 6



- MESSAGE BLINKER program, 128
- Microcomputer, 1, 2
- Microcomputer chip, 1, 3
- Microprocessor, 2
- MIDS**, 239, 247, 259, 335
- MOTOR**, 335
- Multiple statements per line, 24, 28, 98
- Multiplication (\*), 357
- Musical accompaniment, 24
- MUSIC FROM ASCII CHARACTERS program, 253
- "My Computer Likes Me," 299
- MYSTERY PROGRAM**, 83, 111, 135, 136, 137, 201
- NAME BLINKER** program, 115
- NAME, TONE, AND COLOR** program, 110
- NEW**, 35, 37, 335
- NEXT**, 85, 95, 96, 329
- Number boxes, 59, 60
- Numeric variables, 59, 61, 88, 273
- Numeric variable, subscripted, 273, 274
- OD ERROR**, 77
- OK**, 12, 13
- ON**, 299, 301, 335, 336
- ON . . . GOSUB . . .**, 392, 335
- ON . . . GOTO . . .**, 301, 302, 336
- OPEN**, 336
- Order of arithmetical operations, 357
- PAINT RECTANGLES** program, 173
- PAINT STRIPES** program, 154
- PEEK**, 336, 373
- POINT**, 337
- POKE**, 299, 316, 317, 338
- Popular Computing*, 33, 299, 376
- POSITIVE, NEGATIVE, or ZERO** program, 207, 300
- PRINT**, 11, 15, 16, 35, 45, 48, 72, 90, 338
- PRINT#71, 115, 125, 339**
- Print position, screen, 115, 122, 146
- PRINT WHAT WHERE** program, 126
- Program, 1, 4, 7
- Program, BASIC, 1, 4, 35
- Program, computer, 4
- Program PAK, 1, 5
- Question mark (?), 64, 75, 87, 93, 107
- Quotation marks ("), 15, 16, 17
- RAM**, 1, 6
- Random Access Memory (RAM), 1, 6
- RANDOM COLOR BLIPS** program, 186
- RANDOM GRAPHICS CHARACTERS** program, 192
- RANDOM MUSIC FROM AN ARRAY** program, 284
- RANDOM MUSIC** program, 181, 233
- random numbers, 177, 179, 180
- RANDOM NUMBERS, 1 to N** program, 180
- RANDOM STARS** program, 185
- RANDOM STRIPES** program, 189, 191
- REACTION TIME PROGRAM**, 228
- READ**, 59, 77, 105, 339
- READ AND DATA COLOR TONES** program, 80
- READ AND DATA MUSIC** program, 78
- READ-DATA PAINTBRUSH** program, 172
- Read Only Memory (ROM), 1, 5, 6
- Read/Write Memory (RAM), 6
- REM**, 339
- REMARK**, 35, 52, 101, 339
- Remove a line, 50
- Reserved word, 74, 360
- Reserved words, list of, 360
- Reserve string space, 239, 324
- RESET**, 153, 340
- RESTORE**, 242, 340
- RETURN**, 119, 340
- Reverse color, 17, 18, 22
- REVERSE 3-CHARACTER STRING** program, 271
- RIGHTS**, 239, 261, 341
- RND**, 177, 179, 180, 342
- ROM**, 1, 5
- RQ (RuneQuest)**, 177, 193, 256
- RUN**, 35, 36, 40
- RuneQuest**, 177, 193, 256
- RUNNING DICE TOTALS** problem, 320
- Scale of C, 53
- Scientific notation for numbers, 358
- SCRABBLE SCORES**, 269
- Screen colors, 20, 21, 167
- Screen map, 124, 131, 132, 147, 151, 361-363
- Screen position, 115, 122, 146
- Screen **PRINT** position, 115, 122, 146
- Screen **SET** position, 146
- SCRIBBLE ON THE SCREEN** program, 230
- Semicolon (;), 45
- SET**, 143, 144, 342
- SGN**, 299, 301, 342
- SHIFT**key, 15, 16
- SIN**, 342
- SKIPF**, 343, 353
- SN ERROR**, 14, 42
- SOMETHING IN THE SKY** program, 150
- SOUND**, 24, 25, 27, 35, 343, 364
- SOUND AND COLOR ORGAN** program, 290
- STARS—A GUESSING GAME** program, 232
- Statement, 11, 35
- Statement, direct, 11, 16, 20, 36
- Statement, immediate, 11, 36
- Statements, multiple per line, 24, 28, 98
- STEP**, 102, 103, 329
- STEPing backwards**, 102, 130
- STOP**, 344
- STR\$**, 299, 307, 344
- String, 16, 85, 86, 239
- String boxes, 85, 86
- String, empty, 225
- String functions, 239
- String, in **INPUT**, 69, 70
- STRING SQUEEZER** program, 309
- String variable, 85, 88, 273
- String variable, subscripted, 273, 274
- STRIPE 'PAINTBRUSH'** program, 161
- Subroutine, 119
- Subroutine, time delay, 119
- Subscripted numeric variable, 272, 274
- Subscripted string variable, 273, 274
- Subscripted variable, 273, 274
- Substring, 239
- Subtraction (-), 357
- SUM AND PRODUCT OF DIGITS** problem, 319
- Switchbox, 347
- Syntax, 3
- TAB**, 344
- Tape Cassette, 7
- Television, 1, 2
- THEN**, 205
- THREE DIGIT NUMBER SPLITTER** problem, 319
- Time delay, 85, 95, 100
- Time delay subroutine, 119
- TIRED ARROW** program, 129
- Tone, 25, 27, 364
- TRS-80 Color BASIC**, 1, 3
- TRS-80 Color Computer**, 1, 2
- TRS-80 Color Computer Operation Manual*, 2, 349
- T&T (Tunnels & Trolls)**, 177, 193, 256
- TV**, 1, 2
- TV screen, 6, 11
- TWINKLING STARS** program, 276, 280, 295
- TWO-DIGIT NUMBER SPLITTER** program, 306
- TWO GRAPHICS CHARACTERS** program, 193
- Two statements on a line, 23, 24, 54
- UNFAIR COIN FLIPPER** program, 212
- UNTIRING LEFT ARROW** program, 130
- USR**, 344
- VAL**, 239, 255, 345
- Value of a numeric variable, 59, 61
- Value of a string variable, 85, 88
- Variable, numeric, 59, 61, 73, 88, 273
- Variable, string, 85, 88, 273
- Variable, subscripted, 273, 274
- VERTICAL COLOR STRIPES** program, 159
- VHF antenna connection, 347
- WANDERING CHARACTER** program, 199
- WORD MAKER** program, 263
- WORD'S WORTH #1**, 267
- WORD'S WORTH #2**, 268



Computers

\$9.95

# TRS-80<sup>TM</sup> COLOR BASIC

The new, immensely popular TRS-80<sup>TM</sup> Color Computer is friendly, fun, and easy to use by kids and adults, at home or in school. Now Bob Albrecht, the "dragon" of personal computing, shows you how to use the unique color, sound, and graphic capabilities of this low-cost machine for educational and recreational activities. And, as you learn and play your way through this crystal-clear guide, you can teach yourself BASIC, the language of the TRS-80 and many other computers. Packed with games, experiments, programming problems and solutions, this entertaining self-instructional book is *the* ideal introductory aid for kids, parents, and teachers using the Color Computer.

TRS-80<sup>TM</sup> COLOR BASIC leads beginners step by step into good programming practices. The programs in this book are exceptionally readable. No previous computer experience is required. You'll start with "easy stuff" . . . move on to recreations with words, numbers, music, and color graphics . . . and master games and simulations. You'll find activities related to popular fantasy role-playing games. There are plenty of exercises, scores of intriguing challenges (with solutions), and ample opportunities to experiment with your own ideas.

There's an entire chapter devoted to programming problems. Many different solutions are offered, showing how various features of BASIC apply and offering brand-new methods that readers can also adapt to Microsoft BASIC languages on other popular personal computers.

**"So, explore, enjoy, and tell us about your discoveries!"**  
—Bob Albrecht, "The Dragon"

**Bob Albrecht** is a founder of People's Computer Company, Computer Town, USA! and Computer Kid, USA! He writes regularly for *Popular Computing* and *The Computing Teacher* and is the author of five other best-selling **Wiley Self-Teaching Guides**.

More than a million people have learned to program, use, and enjoy microcomputers with Wiley Self-Teaching Guides. Look for them all at your favorite bookshop or computer store!

**JOHN WILEY & SONS, Inc.**  
605 Third Avenue, New York, N.Y. 10158  
New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0 471-09644-X



TRS-80<sup>TM</sup> is a trademark of Tandy Corp.  
Dragon Illustration: Yuri Salzman

ALBRECHT

TRS-80<sup>TM</sup> COLOR BASIC

A Self-Teaching Guide



Wiley