

Cat. No. 62-2061

Nine Dollars and Ninety-Five Cents

Radio Shack

TRS-80™

Level II BASIC

TRS-80 Level II BASIC a self-teaching guide



By
Bob Albrecht,
Don Inman and
Ramon Zamora

Albrecht
Inman
Zamora

a self-teaching guide

TRS-80 BASIC

TRS-80 BASIC

**BOB ALBRECHT
DON INMAN
and
RAMON ZAMORA**

*Dymax Corporation
Menlo Park, California*

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

Copyright © 1980, 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 Sections 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, Robert L

TRS-80 BASIC.

(Wiley self-teaching guides)

Includes index.

1. TRS-80 (Computer)—Programming—Programmed instruction. 2. Basic (Computer program language)—Programmed instruction. I. Inman, Don, joint author. II. Zamora, Ramon, joint author. III. Title.

QA76.8.T18A4 2 001.64'2 80-10268

ISBN: 0-471-06466-1

Printed in the United States of America

81 80 10 9 8 7 6 5 4 3 2 1

To the Reader

Microcomputers are beginning to appear everywhere. This is especially true of the Radio Shack TRS-80 computers, with thousands of these machines now being used in homes, schools, and businesses. Why?

The size of electronic computer parts has been decreasing rapidly in the past few years. Computers—complete systems including printing and storage devices—are now small enough to fit easily into small spaces. This new breed of computers can easily be carried and used where needed. We no longer have to use big computers locked away in big buildings to perform tasks about the home or office. We can now take the computer wherever we go.

With the reduction in size of computers also comes a reduction in cost. For as little as \$600, we can now own a machine that would have cost \$20,000 several years ago.

In fact, most of you, the readers, probably already own some kind of TRS-80. This book is a joyful exploration of that inexpensive and powerful machine that many of you now have—the TRS-80 Model I with Level II BASIC.

This book begins by showing you what your TRS-80 can do for you, even if you don't know how to program. The TRS-80's internal language, BASIC, is introduced step by step. (BASIC stands for Beginners All-purpose Symbolic Instruction Code, and was developed by John Kemeny and Thomas Kurtz at Dartmouth College.) In the early parts of the book, you learn how to get BASIC to work for you on your TRS-80.

We also show you many short programs that introduce and illustrate features of the BASIC language. As you progress through the book, you learn to understand and read BASIC programs and, ultimately, how to build *your own* BASIC programs.

The authors of this book are editors of *Recreational Computing* magazine.* We believe that learning to program should, and can, be an enjoyable experience. We feel that computer terminology and concepts can be introduced within a

*We encourage you to write to us at *Recreational Computing*, P.O. Box E, Menlo Park, CA 94025.

framework of fun and exploration, and that when we do this, *true* learning takes place. You will find small games and recreations sprinkled throughout the book. We also present application programs and the elements of developing simulation routines. Everything is directed toward teaching you how to get the most out of your TRS-80—the most use, the most enjoyment, the most satisfaction.

The TRS-80 computer is a tool that you can use to aid you in whatever you are doing. This book is designed to help you fully explore that tool and learn some general information about programming that applies to many other computers.

The next section tells you How to Use This Book. Why don't you first browse through that material, then begin your adventures with your TRS-80 and BASIC.

How to Use This Book

This book is a Self-Teaching Guide. This means that *you* can use the book to teach *yourself*. Each chapter of the book is composed of short, numbered sections called *frames*. Each frame presents a single idea or topic on the BASIC language, the TRS-80, or a program that is being developed. At the end of most frames are questions for *you* to answer or exercises for *you* to do. You can use a piece of paper to cover the answers that are given below the dashed line within the frame, if you like. You can also use the paper to write down your answers and exercise results before you look at our versions.

We encourage you to use this book while you are in front of a TRS-80 computer. Try the programs and exercises that are discussed on the machine. Let the TRS-80 be your “teacher.”

The first page of each chapter briefly lists what the chapter is to cover. Scan that list. If you feel you already know the material to be covered, skip to the back of the chapter and take the Self-Test. The answers to the Self-Test questions list the frame numbers within the chapter from which the question was taken. You can review that frame if you have trouble answering the question.

The material in the book gets more challenging as you move through the chapters in order. If you are a beginner, start with the first chapters and don't try to skip around. If you already know some programming, look at the table of Contents and feel free to explore some of the later chapters.

It might be a good idea to glance at the Contents anyway, or to look at the Appendixes in the back of the book. These are designed to help you get started understanding what the components do and connecting the parts of the TRS-80. The appendixes will become useful to you as you get to specific sections. For now, just note what is available and where things are within the book.

As a final note before you start, be aware that as you use the book, enter lines into your computer, answer Self-Test questions, and do the frame exercises:

YOU CANNOT DO ANYTHING WRONG!

There is no way you can “hurt” or “harm” the computer by what you type into it. You may make *mistakes*, but that is a natural part of learning and exploration. In fact, we introduce deliberate errors in several places as part of the learning process.

So, explore, enjoy, and tell us about your discoveries as you use your Radio Shack TRS-80 microcomputer.

Contents

| | | |
|------------|---|------------|
| Chapter 1 | The Radio Shack TRS-80 Computer | 1 |
| Chapter 2 | Getting Started | 11 |
| Chapter 3 | Basic Programs | 39 |
| Chapter 4 | FOR-NEXT Loops | 75 |
| Chapter 5 | Meandering | 105 |
| Chapter 6 | Patterns and Games | 133 |
| Chapter 7 | Entering and Displaying Data | 161 |
| Chapter 8 | Strings | 185 |
| Chapter 9 | One-Dimensional Arrays | 213 |
| Chapter 10 | Multidimensional Arrays | 233 |
| Chapter 11 | Editing and Debugging Programs | 259 |
| Chapter 12 | Graphics, Games, and Programs for the Home | 295 |
| Appendix A | Setting Up Your TRS-80 | 324 |
| Appendix B | The Cassette Recorder | 329 |
| Appendix C | Arithmetic | 333 |
| Appendix D | Error Messages | 339 |
| Appendix E | Print and Graphics Layout Sheets | 342 |
| Appendix F | Reserved Words | 345 |
| Appendix G | ASCII Codes | 347 |
| Index | | 349 |



CHAPTER ONE

The Radio Shack TRS-80 Computer

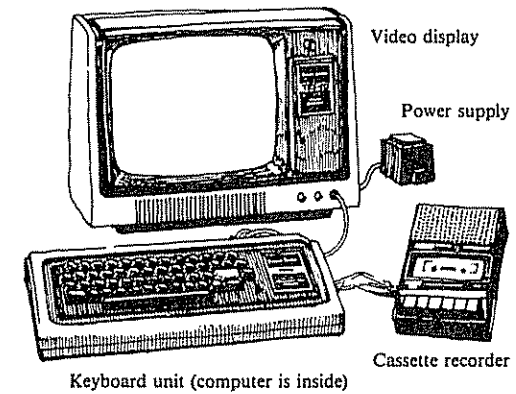
Here your fun begins. In this chapter, we will introduce you to the Radio Shack TRS-80, a powerful, friendly computer used in hundreds of thousands of homes, schools, and businesses. You will also begin to learn *computerese*, the jargon or terminology of computers, so you will be able to talk about the things you do on the TRS-80.

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

- Radio Shack TRS-80 Microcomputer System
- Keyboard
- Video display
- Cassette recorder
- Power supply
- Computer language
- BASIC (specifically, TRS-80 Level II Basic)
- Read Only Memory (ROM)
- Random Access Memory (RAM)
- BASIC program

And, at the end of this chapter, you will be ready to continue with Chapter 2, in which you begin “talking” to the TRS-80!

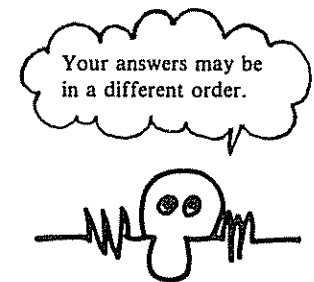
1. The TRS-80 Microcomputer System has four parts, shown in the diagram below.



What are the four parts of the TRS-80 computer?

- (a) _____
- (b) _____
- (c) _____
- (d) _____

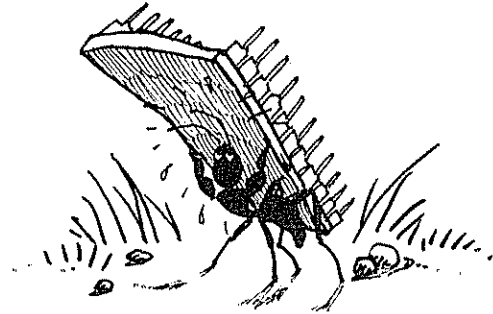
-
- (a) keyboard unit
 - (b) video display
 - (c) power supply
 - (d) cassette recorder



As you can see in the illustration, the video display, power supply, and cassette recorder each has a cable that plugs into the keyboard unit. If you need information on how to set up your TRS-80, see Appendix A, "Setting Up Your TRS-80," or consult the *LEVEL II BASIC Reference Manual* that accompanies the TRS-80. You can obtain this reference manual from most Radio Shack stores.

2. The computer is inside the keyboard unit. It consists of a set of "chips." A chip is a tiny wafer containing thousands of *microscopic* circuits; that's why the TRS-80 is called a *microcomputer*.

Perhaps you have seen pictures of a microcomputer chip resting on the end of someone's finger. We choose a different way to emphasize the small size of a microcomputer chip.



Check all appropriate responses below.

A microcomputer chip is:

- (a) about the size of a microscope
- (b) smaller than a breadbox
- (c) too small to be seen by human eyes
- (d) about the size of an ant
- (e) delicious with avocado dip

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

3. Oops! Maybe we should be more serious. Well, we will try; but it's difficult for us to remember that computers should be used for serious stuff, when they are so much fun to use.

As you work through this book (adventuring as you go!), 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 "converse" with the TRS-80.

What is BASIC? _____

a computer language or a simple language for communicating with a computer

There are many variations, or *dialects*, of BASIC. In this book, we will describe Level II BASIC for the Radio Shack TRS-80 computer. In Chapter 2, you will begin to learn BASIC.

IMPORTANT NOTICE! Most TRS-80s use Level II BASIC. However, some TRS-80s use a different version of BASIC called Level I BASIC, which is completely described in the excellent *User's Manual for Level I* by Dr. David Lien. This manual is available from most Radio Shack stores.

4. As you learn BASIC, you will also learn to read and understand *computer programs* written in BASIC and start writing original, never-before-seen-on-earth programs—*your* programs.

A program is a set of instructions. You may have already used or written “programs” in English (or another noncomputer language). For example:

- A recipe for baking a cake
- Instructions for assembling a model from a kit
- Directions on how to get to someone's house
- And, of course, 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. The vocabulary and syntax (rules) are stored in the *memory* of the TRS-80, located in the keyboard unit. BASIC is actually stored (or “remembered”) in a set of chips called ROM, which means *Read Only Memory*.

Information in ROM (or in the memory of the computer) is permanently stored, much like 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 of the calculator are permanently stored in ROM.

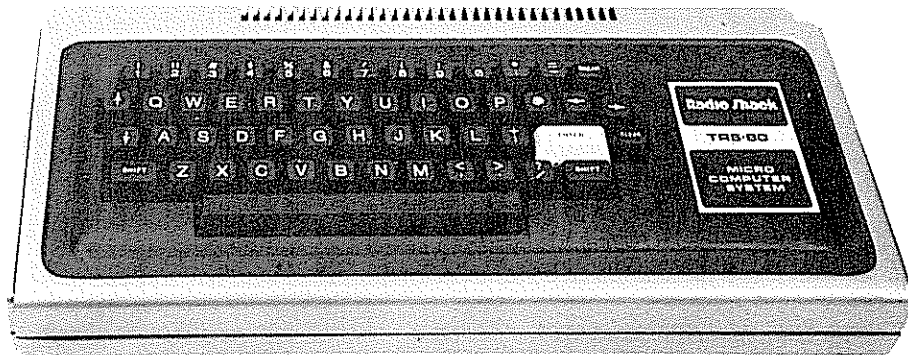
When you turn on the TRS-80, it immediately “knows” BASIC—simply by looking into its ROM.

What is ROM? _____

Read Only Memory

The TRS-80 may read information from ROM, but cannot erase it or change it in any way. That is why this type of memory is called *Read Only Memory*.

6. Perhaps you have noticed that the keyboard unit has a keyboard. (Hmm . . . do you suppose that is why we keep calling it "the keyboard unit"?) Let's see what the keyboard is all about.



The keyboard is your control center, your means of communicating with the TRS-80. You use the keyboard to type information into the computer. As you type, the information that you type is stored in RAM (*Random Access Memory*), located inside the keyboard unit.

RAM is different from ROM. Information can be *stored into*, or *written into*, RAM. Also, the information in RAM can be erased or changed. 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 it can be read from, or written into. 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 entered from the keyboard?

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

7. What you type on the keyboard is stored in RAM. It also appears on the TV screen of the *video display* so that you can see what you type. Video is just a fancy name for TV; so, from now on we will refer to the video display as the TV screen.

As you will soon see, the computer also prints information on the TV screen. The keyboard, together with the TV screen, provides two-way communication.

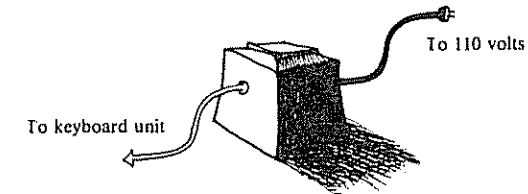
When *you* type on the keyboard, the stuff that you type is stored in the _____ of the TRS-80 and is also displayed, or printed, on the _____.

memory (or RAM); TV screen (or video display)

8. The power supply, when it's plugged in, just sits there humming away. Touch it—it is slightly warm. The power supply provides the right kind of power to the keyboard unit by converting 110-volt AC to low-voltage DC (direct current) required by the computer chips inside the keyboard unit.

Suppose you turn your TRS-80 on and nothing happens. What might you check first? (It is OK to guess!) _____

Make sure the power supply is plugged in: (1) power cord to 110-volt outlet and (2) power cable to TRS-80 computer!



9. Thousands of programs are available on prerecorded tape cassettes. A single cassette, which might cost from \$3 to \$30, may contain one large program or several small programs. Using the cassette recorder, you can quickly load one of these programs into your TRS-80, then enjoy the use of a program written by an expert. The best source of information on commercially available cassettes is Robert Elliott Purser's *Computer Cassette Catalog*, P.O. Box 466, El Dorado, CA 94623.

As you learn to program in BASIC, you can also 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. The methods for recording and reading programs on tape cassettes are described in Appendix B, "The Cassette Recorder."

Relax! No questions. Read on.

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

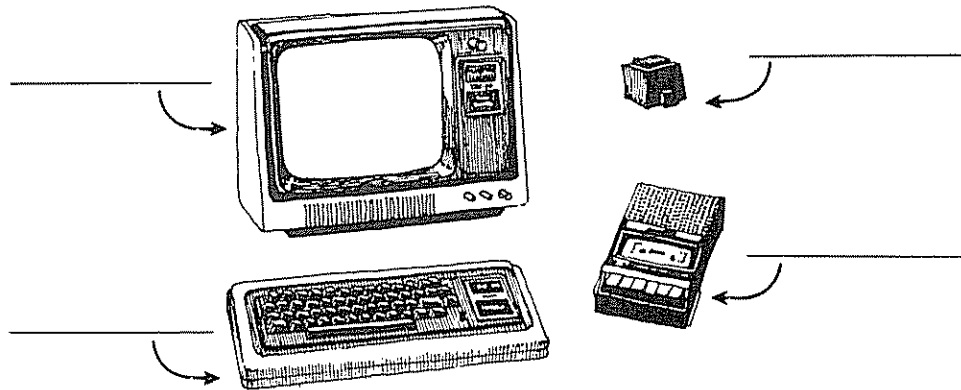
- read Appendix A, "Setting Up Your TRS-80"; or
- read "Setting Up the System" in the *Level II Basic Reference Manual*, available at most Radio Shack Stores; or
- yell for help! Ask someone to show you how to hook up your TRS-80.

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

Try this Self-Test, so you can evaluate how much you have learned so far.

1. The TRS-80 microcomputer has four major parts, described in this chapter. In the following diagram, write the names of the four major parts.



2. A microcomputer is called a microcomputer because (check the one that applies best):
- (a) It can be carried by a strong ant.
 - (b) It contains thousands of very tiny electronic circuits in a small space (1/4" by 1/4").
 - (c) It can be seen only by mountain sheep with telescopic eyesight.
 - (d) It is manufactured by leprechauns using miniature tools.
3. In the TRS-80, where is the microcomputer chip located? _____

4. We "talk" to the TRS-80 by using a _____ called BASIC. The vocabulary and syntax (rules) for this language are located in a special kind of memory in the keyboard unit. What is this type of memory called? _____
5. How do we get information into the TRS-80? _____

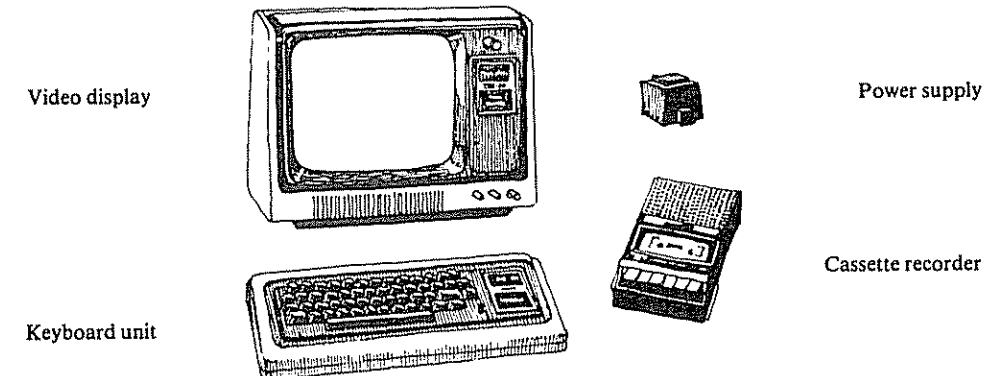
6. When we type information into the TRS-80, it is displayed on the TV screen and also stored in _____.
7. What is a BASIC program? _____

8. What is the function and purpose of the cassette recorder? _____

Answers to Self-Test

The numbers in parentheses refer to the frames in the chapter where the topic is discussed. You may wish to refer to these for quick review.

1. (frame 1)



2. (b) It contains thousands of tiny circuits in a small space (1/4" by 1/4"). (frame 2)
3. In the keyboard unit. (frame 2)
4. Computer language; ROM, or Read Only Memory. (frames 3,5)
5. We type it on the keyboard. (frame 6)
6. RAM, or Random Access Memory. (frames 6,7)

7. A set of instructions that tells the computer what to do and how to do it. (frame 4)
8. It enables us to save, or store, information on tape cassettes from the memory (RAM) of the TRS-80. It also enables us to load information on tape cassettes into the memory (RAM) of the TRS-80. (frame 9)

CHAPTER TWO

Getting Started

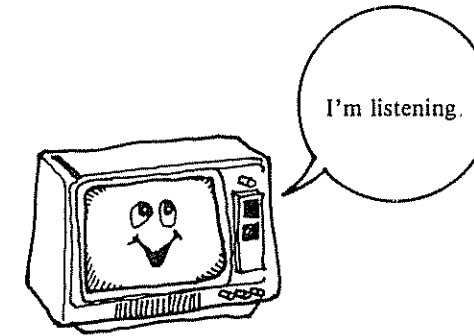
Now the fun *really* begins. In this chapter, you will begin learning how to “talk” to your TRS-80 using instructions called *direct*, or *immediate*, statements. These statements are direct or immediate because the computer executes or does them immediately after you type them and then press the **ENTER** key on the keyboard.

A direct statement tells the TRS-80 to do something; the TRS-80 does it immediately, then waits for your next instruction.

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

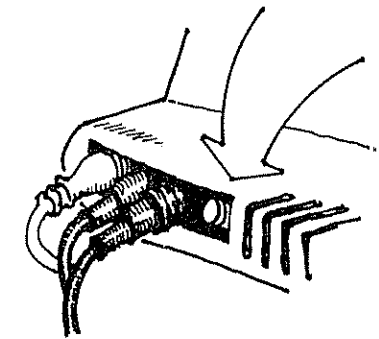
- clear (erase the TV screen);
- use direct PRINT statements to tell the computer to print information on the TV screen;
- recognize error messages from the computer (in case you make an error);
- correct typing errors or delete a direct statement that contains errors;
- use direct statements to do arithmetic.

We assume that your TRS-80 is set up and ready to go. If you have not yet set up your TRS-80, do so now and begin “talking” to the computer. You can find directions on how to set up the TRS-80 in Appendix A, “Setting Up Your TRS-80.”

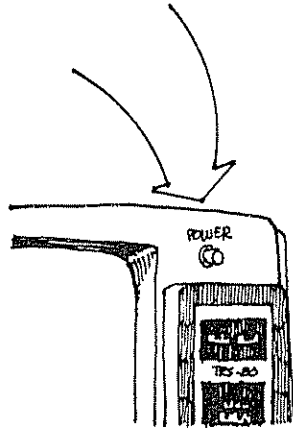


1. Ready? Let's begin. We assume that your TRS-80 is hooked up and ready to go, but is not yet turned on. So, first turn on the keyboard unit and the video monitor.

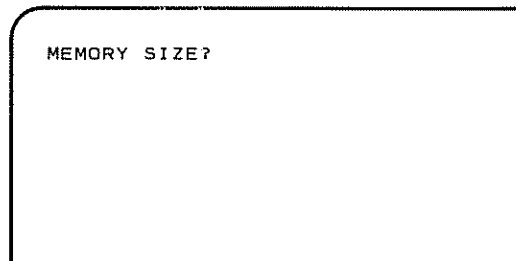
The power switch for the keyboard unit is in the back on the right side, as you face the keyboard unit. To turn *on* the TRS-80, press it in—it will lock. To turn *off* the TRS-80, press it—it will pop out.



The power switch for the video display is in front, on the upper right. To turn *on* the video display, press it in—it will lock. To turn *off* the video display, press it—it will pop out.



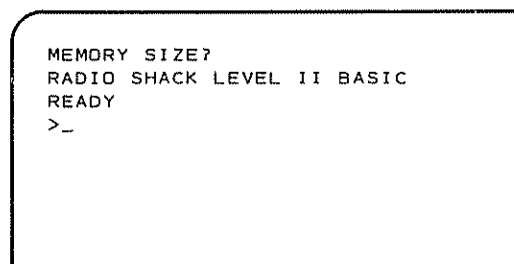
2. When you first turn on the keyboard unit and the video display, the TV screen will look like this:



Ignore the question. In this beginner's book, we will not worry about the answer, which might be of interest to advanced or expert computer users.

Find the large **ENTER** key on the keyboard. Press it. The TV screen should now look like this:

The arrow (>) is called the *prompt* and the dash (—) is called the *cursor*.



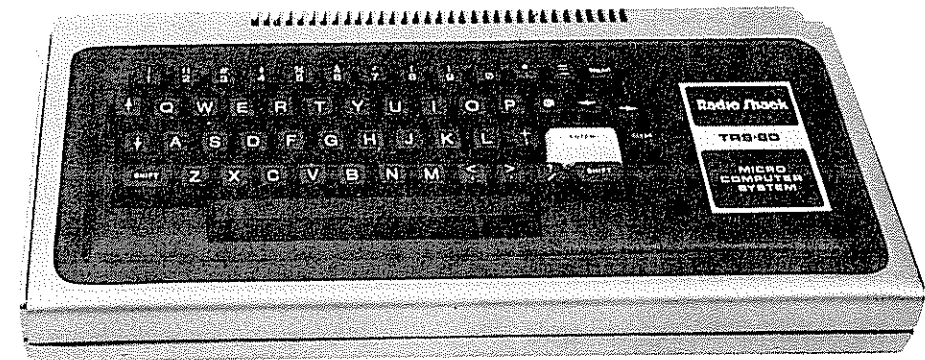
You press the **ENTER** key to make the TRS-80 **READY** for *you* to use. You know that it is ready for you to use when it prints **READY** on one line and a *prompt* and a *cursor* on the line below. The prompt (**>**), followed by the cursor (**_**), tells you that it is *your* turn to do something.

Whenever you see the prompt and the cursor, you know that it is *your* turn to do something. If you don't do something *promptly* (oops! sorry about that pun), the TRS-80 will simply wait patiently until you are ready to use it.

- (a) What does the prompt look like? _____
- (b) What does the cursor look like? _____

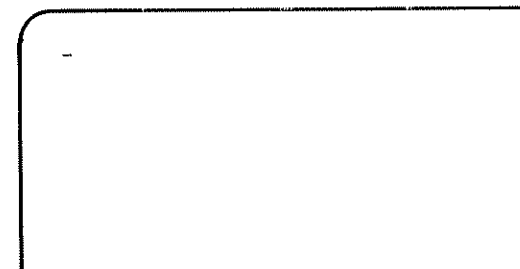
(a) >; (b) _

3. The keyboard is your control center.



Let your eyes wander over the keyboard. Find the **CLEAR** key. Press the **CLEAR** key. This "clears" or "erases" the TV screen. That is *all* that happens; the **CLEAR** key does not clear, or erase, anything *inside* the computer (which, as you know, is in the keyboard unit).

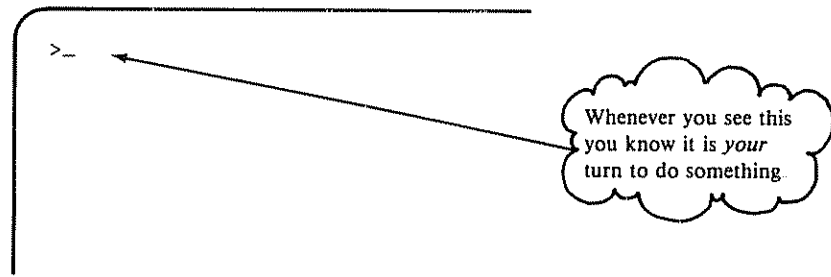
The screen now looks like this:



I get it! The screen is *clear* (empty), except for the cursor.



Now press the **ENTER** key. The screen will look like this:



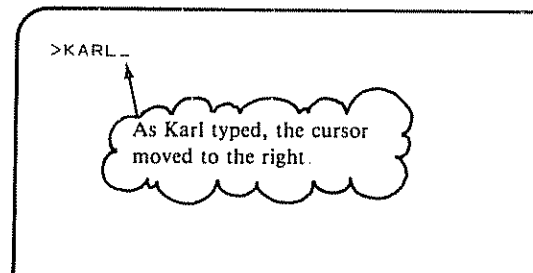
You know that it is your turn to do something because you can see the _____ and the _____ on the screen.

prompt (>); cursor (_). Together, they look like this: >_.

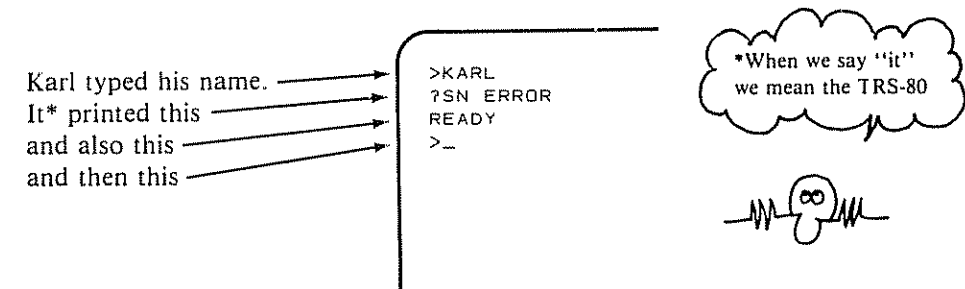
If we ask you to "clear the screen," please press the **CLEAR** key, then press the **ENTER** key. Remember, *only* the TV screen is cleared, or erased. Information inside the keyboard unit is not affected.

4. Your TRS-80 is waiting patiently for you to do something. So, try this: Type your name and then press the **ENTER** key. The TRS-80 will probably print an error message on the screen. Here is what happened when Karl typed his name.

Karl typed his name:



Then Karl pressed the **ENTER** key:



The strange message ?SN ERROR means that the TRS-80 did not understand Karl. Karl (who is eleven and proud of his name) was quite surprised. After all, how could *his* name be an error? We explained to Karl that the computer simply didn't understand him. The word KARL is not one of those special words (in BASIC) which the TRS-80 understands. Aha! said Karl, and he, as you will also, began to learn about those few special words that the TRS-80 *does* understand.

(a) If you type something, then press the **ENTER** key, and the TRS-80 prints ?SN ERROR, what is it trying to tell you? _____

(b) After typing ?SN ERROR, what does the TRS-80 do? _____

- (a) It does not understand you.
- (b) It types READY, then prints a prompt and a cursor.

The READY, the prompt (>), and the cursor (__) are the TRS-80's way to let you know that everything is OK. It just didn't understand you, because it doesn't know English. After you make a "mistake," it will print an "error" message followed by the prompt and the cursor so you know it is ready for you to try again.

SN ERROR means SYNTAX ERROR. You will get this message when you type something that TRS-80 Level II BASIC does not understand. This is one example of an error message. Many other error messages are possible; you will encounter others as you experiment with your TRS-80 or try out our examples and exercises. See Appendix D, "Error Messages," for a complete list of TRS-80 Level II BASIC error messages.

5. The TRS-80 understands only a few words *as instructions to do something*. So, avoid error messages. Learn how to tell the TRS-80 to do what you want it to do. How? By learning the words that the TRS-80 understands.

The TRS-80 understands a language called BASIC. In BASIC, there are special words to tell the computer to do special things.

A special BASIC word \longrightarrow PRINT

PRINT tells the TRS-80 to print something on the TV screen. Here is how to get the TRS-80 to print your name. Karl will demonstrate.

First, Karl clears the screen. He presses the **CLEAR** key, then presses the **ENTER** key.

The screen is clear, except for the prompt and the cursor.

```
>_
```

Then, Karl types \longrightarrow

```
>PRINT "KARL" _
```

Note that KARL is enclosed in quotation marks.

As Karl typed, the cursor moved to the right.

Next, Karl presses the **ENTER** key.

Karl typed this \longrightarrow
It printed this \longrightarrow
and this \longrightarrow
and this \longrightarrow

```
>PRINT "KARL"  
KARL  
READY  
>_
```

No questions. Read on!

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

You press **CLEAR**, then press **ENTER**.

You type `PRINT "KARL"` (remember the quotation marks!).

You press **ENTER**.

If you have done everything exactly as we have suggested, the screen will now look like this:

```
>PRINT "KARL"
KARL
READY
>_
```

If the screen doesn't look like the above screen, try again. Be careful to spell `PRINT` correctly and *do* remember to put quotation marks before and after Karl's name.

```
PRINT "KARL"
```

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

Now, pretend that you are the TRS-80 and complete the following.

We type: `PRINT "MY HUMAN UNDERSTANDS ME"`

It prints: _____

```
-----
MY HUMAN UNDERSTANDS ME
READY
>_
```

7. The statement `PRINT "MY HUMAN UNDERSTANDS ME"` is called a *direct PRINT* statement. It tells the TRS-80 to print something on the screen. The computer prints the verbal message within the quotation marks that follows the word `PRINT`. The message enclosed in quotation marks is called a *string*.

```
PRINT "MY HUMAN UNDERSTANDS ME"
```

This is a string. It is enclosed in quotation marks. The quotation marks enclose the string, but are not part of the string.

A string is any bunch of keyboard characters, typed one after the other. For example:

A string can be a name: `KARL`

A string can be a license plate number: `SAM 123`

A string can be gibberish: `AB#$%JFDZ?*`

A string may include:

Numerals (0, 1, 2, . . .)

Letters (A, B, C, . . .)

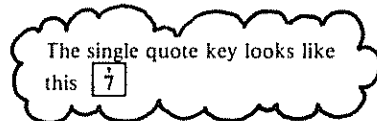
Special characters (+, -, *, /, ↑, #, %, \$, comma, period, space, etc.)

Since double quotation marks (") define the beginning and the end of a string, do you think they can be used as a character in the string? _____

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

We type: PRINT "THEY SAID, 'ALL RIGHT.'"

It prints: THEY SAID, 'ALL RIGHT.'
READY
>_



8. Complete the statement so that the computer prints what we say it prints.

We type: PRINT _____

It prints: THE ROAST IS DONE. TURN OFF THE OVEN
READY
>_


"THE ROAST IS DONE. TURN OFF THE OVEN."

Did you remember the quotation marks? Someday, of course, the computer will turn off the oven! Almost anything electric can be controlled by the computer, with the proper electrical connections *and* program, of course.

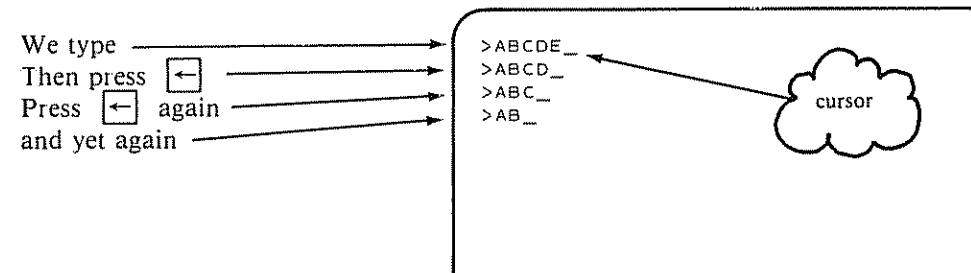
9. Have you made any typing mistakes yet? In case you should make a typing error, BASIC has a dandy way of fixing it. Watch while we make a typing error.

We type: PTINT "DENTIST APPOINTMENT TODAY"

It prints: ?SN ERROR
READY
>_

We misspelled PRINT, so the computer doesn't know what we want. However, 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, which looks like this: . Each time

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



What will happen if we press `←` two more times? _____

The cursor will move two places to the left, erasing A and B.

10. Complete the following so that the computer prints what we want it to print.

We type: `PRIMR _____ "TOMORROW IS YOUR MOTHER'S BIRTHDAY!!!"`

It types: `TOMORROW IS YOUR MOTHER'S BIRTHDAY!!!`
`READY`
`>_`

`← ← N T` (Erase M and R, then type N and T.)

Remember: To erase the last character typed, press `←`. To erase two characters, press `←` twice. And so on. You can't erase the prompt, though, no matter how many times you press the `←` key!

11. The backspace (`←`) key is great for deleting errors that 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 probably get an error message. You would *like* to just scrub the line—that is, erase it from the beginning.

Well, you can! Hold down the **SHIFT** key and press the backspace key.

We type: PRINT "ONCE UPON A TIME THERE WAS A

We hold down the **SHIFT** key and press the backspace key.

Poof! The entire line disappears. The computer erases the entire line and turns on the cursor.

Whenever we see the cursor, what is the computer telling us? _____

It is our turn to do something.

IMPORTANT NOTICE! From now on, we will usually omit the **READY** followed by the prompt (**>**) and cursor (**_**) in order to save time and space.

For example, instead of

```
We type: PRINT "KARL "  
It prints: KARL  
          READY  
          >_
```

We might show only this

```
We type: PRINT "KARL "  
It prints: KARL
```

Please read above
IMPORTANT NOTICE.

12. We can also tell the TRS-80 to do arithmetic and print the answer. In other words, we can use the computer as a calculator. For example, we can use the addition key (+) to tell the computer to add 7 and 5, as follows.

We type: `PRINT 7 + 5`

It prints: 12 (Remember, we are omitting READY and >_.)

The statement `PRINT 7 + 5` tells the TRS-80 to compute the value of $7 + 5$ (do the arithmetic) and then print the answer.

Note that we tell the computer to do actual calculations by *not* using quotation marks in the `PRINT` statement. Compare the formats below.

with ””

We type: `PRINT "13 + 6"`

It prints: 13 + 6

`PRINT "13 + 6"`

This is a string. The computer prints it *exactly* as it appears.

without ””

We type: `PRINT 13 + 6`

It prints: 19

`PRINT 13 + 6`

This is a *numerical* expression (no quotation marks). The computer does the arithmetic and prints the result.

Your turn. Complete the following, showing what the TRS-80 prints.

We type: `PRINT 23 + 45`

It prints: _____

We type: `PRINT 1 + 2 + 3 + 4 + 5`

It prints: _____

We type: `PRINT "1 + 2 + 3 + 4 + 5"`

It prints: _____

68

15

1 + 2 + 3 + 4 + 5

Remember, we are omitting the READY and >_ which the TRS-80 prints when it finishes doing something.

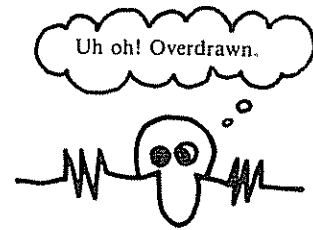
13. Subtraction? Of course. Use the subtract (-) key.

We type: `PRINT 7 - 5`

It prints: 2

We type: `PRINT 25.68 - 37.95`

It prints: -12.27



Complete the following, showing what the computer typed.

We type: `PRINT 29 - 13`

It prints: _____

We type: `PRINT 1500 - 2000`

It prints: _____

We type: `PRINT "1500 - 2000"`

It prints: _____

We type: `PRINT 67 + 32 - 28`

It prints: _____

```
16
-500
1500 - 2000
71
```

14. BASIC uses + for addition and - for subtraction, just as we do with paper and pencil. For multiplication, however, BASIC uses the asterisk (*).

We type: `PRINT 7*5`

It prints: 35

We type: `PRINT 1.23*4.567`

It prints: 5.61741

We type: `PRINT 9*8`

It prints: _____

We type: `PRINT 3.14*20`

It prints: _____



We type: PRINT 2*3 + 4

It prints: _____

72
62.8
10

We do the arithmetic like this: $2*3+4 = 6+4 = 10$.

15. For division, we use the slash (/) key.

We type: PRINT 7/5

It prints: 1.4

We type: PRINT 13/16

It prints: 0.8125

We type: PRINT 24/3

It prints: _____

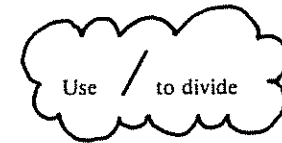
We type: PRINT 3.14/4

It prints: _____

We type: PRINT 3/4 - 2/5

It prints: _____

0
0.785
0.35 $(3/4 - 2/5 = 0.75 - 0.4 = 0.35)$



16. Now let's think metric. Mary Jo is 66 inches tall. How tall is she in centimeters. Hmmm . . . we seem to recall:

1 inch = 2.54 centimeters

We type: PRINT 66*2.54

It prints: 167.64

And so, with the help of our ever-willing TRS-80, we enter the metric age and learn that Mary Jo is 167.64 centimeters tall. (Call it 168 centimeters, if you wish.)

(a) A certain conga drum is 29.5 inches tall. (Interestingly, that is also the usual height of an office desk or a dining room table.) Write a PRINT statement to compute and print the height of that conga drum (or desk or table) in centimeters. _____

(b) A football field is 3,600 inches long. Write a PRINT statement to compute the length of a football field in centimeters. _____

(a) We type: `PRINT 29.5*2.54` OR `PRINT 2.54*29.5`
It prints: 74.93

(b) We type: `PRINT 3600*2.54` OR `PRINT 2.54*3600`
It prints: 9144

Perhaps you are accustomed to thinking of a football field as being 100 yards long. OK, do it this way.

We type: `PRINT 100*36*2.54`
It prints: 9144

17. More metric. Let's turn it around. An ancient king named Fred measures 100 centimeters from the tip of his nose to the end of his outstretched hand (which, of course, is on the end of his outstretched arm). How long is that in inches?

1 centimeter = 1/2.54 inches

We type: `PRINT 100/2.54`
It prints: 39.3701

Does the answer look familiar? Perhaps you recall that 100 centimeters is equal to one meter and one meter is equal to about 39.37 inches, a little more than one yard.

(a) At a certain time during his legendary life, Firedrake the Dragon measured 1,000 centimeters from the tip of his fire-breathing nostril to the longest point of his multiforked tail. Write a PRINT statement to compute Firedrake's length in inches. _____

- (b) It is difficult for some people to think of the length of a dragon in inches. Write a PRINT statement to compute the length of Firedrake in feet.

(Remember, there are 12 inches in one foot.) _____

- (a) We type: `PRINT 1000/2.54`

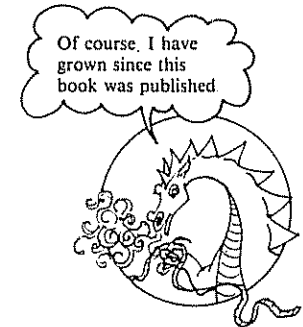
It prints: 393.701

Firedrake is about 393.7 inches long.
To the nearest inch, Firedrake is 394 inches long.

- (b) We type: `PRINT 1000/2.54/12`

It prints: 32.8084

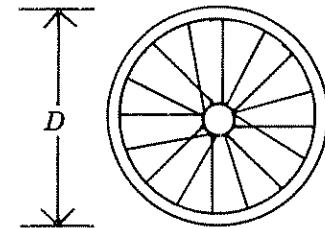
Aha! Firedrake is about 32.8 feet long—longer than our car but shorter than our house. Sigh . . . metric is easy for kids, but hard for those of us who approach Firedrake's age.



18. If you have a bicycle, it probably has a wheel size of 20, 26, or 27 inches in diameter. The wheel size, or diameter, is measured from the outside rim of the tire straight through the center (the hub) to the outside of the tire rim on the other side. Got that? Never mind, if you haven't. Here's a diagram. In the diagram, we use D to represent the diameter of the wheel.

$$C = \pi D$$

where $\pi = 3.14159 \dots$



Our problem: How far does a bike travel in one full turn of the wheel? That is, given the diameter (D) of the wheel, how far does the bike travel in one revolution of that wheel?

Hmmm . . . If D is the wheel diameter, then the distance traveled must be the *circumference* (remember?) of the wheel. Let's call the circumference C . Perhaps you recall:

$$C = \pi D \quad \text{where } \pi = 3.14159 \dots$$

We will use 3.14 as a crude approximation to π .

We type: `PRINT 3.14*20`

It prints: 62.8

So, if your bike has a 20-inch wheel, it travels 62.8 inches each full turn of the wheel.

Your turn. Write PRINT statements to compute the distance traveled for (a) a 26-inch wheel and (b) a 27-inch wheel.

(a) You type: _____

It prints: 81.64

(b) You type: _____

It prints: 84.78

(a) PRINT 3.14*26

(b) PRINT 3.14*27

19. Hark back, if you will, to our metric problems in frame 16. We said that Mary Jo is 66 inches tall. Well, in real life, people usually give their heights in feet and inches. If you asked Mary Jo how tall she is, she would probably tell you that she is 5 feet, 6 inches tall.

Given feet and inches, we can write a simple PRINT statement to provide height in inches.

We type: PRINT 5*12 + 6

It prints: 66

Judy is 5' 3" tall. Write a PRINT statement to compute her height in inches.

We type: _____

It prints: 63

As you probably suspect by now, we are leading up to a new problem.

Problem: Before he reached his full stature, King Kong was once 37' 8" tall. How tall was he in centimeters? We solve it this way.

We type: PRINT(37*12+8)*2.54

It prints: 1148.88

Aha! Note how we have cleverly introduced the use of parentheses (). The rules for using parentheses are very, very similar to the rules that you learned in elementary school math classes. In case you have forgotten, consult Appendix C, which has more information on how the TRS-80 does arithmetic.

So, try one yourself. A certain hobbit, when he joined the fellowship to return The Ring to the fires of Mt. Doom (and thus save Middle Earth), was 2' 7" tall. Write a PRINT statement to compute his height in centimeters.

You type: _____

It prints: 78.74

PRINT 5*12+3

PRINT(2*12+7)*2.54 OR PRINT 2.54*(2*12+7)

20. In BASIC, the rules for arithmetic are very similar to the rules we use in "everyday" math. Remember, though, use an asterisk (*) for multiplication and a slash (/) for division.

It's a very warm day, about 87 degrees Fahrenheit. Let's see now, what's that in Celsius? The formula for converting from degrees Fahrenheit (F) to degrees Celsius (C) is usually shown as follows:

$$C = \frac{5}{9}(F - 32)$$

Here are three ways to compute the degrees Celsius (C) equivalent to 87 degrees Fahrenheit.

(a) We type: PRINT (5/9)*(87 - 32)

It prints: 30.5556

(b) We type: PRINT 5/9*(87 - 32)

It prints: 30.5556

(c) We type: PRINT 5*(87 - 32)/9

It prints: 30.5556

For most people, body temperature is about 98.6 degrees Fahrenheit. Write a PRINT statement to compute body temperature in degrees Celsius.

You type: _____

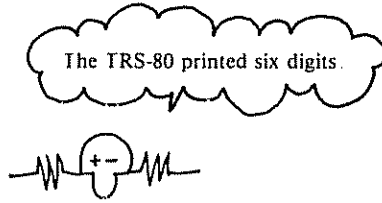
It prints: 37

PRINT (5/9)*(98.6 - 32) OR
PRINT 5/9*(98.6 - 32) OR
PRINT 5*(98.6 - 32)/9

21. Perhaps you have noticed that the TRS-80 will print numbers with up to six digits . . . no more. If the complete result of an arithmetic operation is more than six digits, the TRS-80 will *round* the printed result to six digits.

We type: `PRINT 1.2345*6.789`

It prints: 8.38102,
6 digits



The complete, exact answer is 8.3810205, which has eight digits. The TRS-80 printed the first six digits of the complete answer. The printed answer was rounded at the sixth digit.

Rounded? OK, here is another example.

We type: `PRINT 2/3`

It prints: .666667

The complete answer cannot be given with a finite number of digits. It goes on forever, and ever, and . . . So, our TRS-80 prudently rounded the result to a more printable six digits, rounded at the sixth digit.

six digits
.666666666666 . . . and so on

↑
If the seventh digit is 5 or more, the TRS-80 increases the sixth digit by one.

- (a) Suppose the complete result of an arithmetic operation is 1.23456789. What will the TRS-80 print?

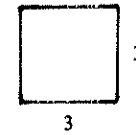
- (b) Suppose the complete result is 3.1415927. What will the TRS-80 print?

- (c) Suppose the complete result is 123.5555. What will the TRS-80 print?

-
- (a) 1.23457
(b) 3.14159
(c) 123.556

IMPORTANT NOTICE! For more information on just how the TRS-80 does arithmetic, see Appendix C.

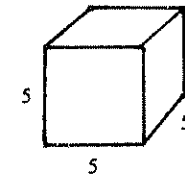
22. Well, as we near the end of this chapter, we would like to introduce you to one more handy operation. This operation is used to compute *powers of numbers*. Here are some examples.



Area of a square: $A = S^2$, where S is the length of a side. The area of the square shown is

$$A = S^2 = 3^2 = 3 \times 3 = 9$$

3^2 is called the *second power* of 3.



Volume of a cube: $V = S^3$, where S is the length of a side. If $S = 5$, then

$$V = S^3 = 5^3 = 5 \times 5 \times 5 = 125$$

5^3 is called the *third power* of 5.

Got it? 3^2 means 3 times 3, or 3×3 ; 5^3 means 5 times 5 times 5, or $5 \times 5 \times 5$.

Complete the following:

(a) 7^2 means _____

(b) 8^4 means _____

Hmmm . . . it is sort of like shorthand—saves time!



(a) 7 times 7, or 7×7 , or the second power of 7.

(b) 8 times 8 times 8 times 8, or $8 \times 8 \times 8 \times 8$, or the fourth power of 8

23. In math books, powers of numbers are shown by using *superscripts*.

The third power of 8 is 8^3 ← This is a superscript

The second power of 7 is 7^2 ← This is a superscript

The seventeenth power of 2 is 2^{17} ← This is a superscript

Of course, 2^{17} could also be written as follows:

$$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$$

So, you see, there really is some advantage in using superscripts. But, alas—our TRS-80 can't print superscripts. (Perhaps a few moments of silence follow.) But

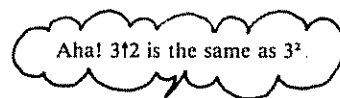
there is another way to do it. Use the up arrow key (\uparrow) to compute powers of numbers.

We type: `PRINT 3↑2`

It prints: 9

We type: `PRINT 5↑3`

It prints: 125



Your turn. Write PRINT statements to compute 8^3 , 7^2 , and 2^{17} .

(a) 8^3
You type: _____

It prints: 512

(b) 7^2
You type: _____

It prints: 49

(c) 2^{17}
You type: _____

It prints: 131072

(a) `PRINT 8↑3`

(b) `PRINT 2↑17`

(c) `PRINT 7↑2`

24. Suppose you put \$123 into a savings account certificate at Erosion Savings and Loan, which yields 8% interest per year. (With inflation at about 12% per year, your money will decrease in value at only 4% per year.) The amount of money in the account after N years can be computed using the formula shown below:

$$A = P(1 + R/100)^N$$

where:

P = Principal, or original amount put into the account

R = Rate of interest per year, in percent

N = Number of years

A = Amount in account at the end of N years

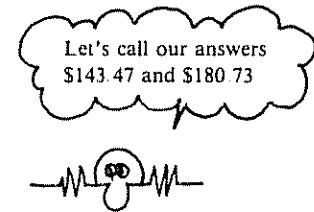
In our problem, $P = \$123$, $R = 8\%$, and we want to know the value of A for $N = 2$, $N = 5$, and $N = 9$ years.

We type: `PRINT 123*(1 + 8/100)^2`

It prints: 143.467 ← After 2 years

We type: `PRINT 123*(1 + 8/100)^5`

It prints: 180.727 ← After 5 years



Your turn. Write the PRINT statement for $N = 9$. We have supplied the result printed by the computer.

We type: _____

It prints: 245.877 ← After nine years our money almost doubled. (But it buys less than it did nine years ago.)

`PRINT 123*(1 + 8/100)^9`

Appendix C shows how to tell the TRS-80 to round money to the nearest penny.

25. Perhaps you have heard the ancient story about the wise person who did a great service for a wealthy king. The king asked this person what reward would be appropriate. The person's request was simple. She asked only for grains of wheat, computed as follows. For the first square on a chessboard, one grain of wheat; for the second square, 2 grains of wheat; for the third square, 4 grains of wheat, and so on, doubling at each square. It goes on as shown below:

| <i>Square Number</i> | <i>Grains of Wheat</i> |
|----------------------|------------------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 4 |
| 4 | 8 |
| 5 | 16 |

And so on. For square N , there will be 2^{N-1} grains. Let's find out how many grains are on square 16.

We type: `PRINT 2^15` (since $N = 16$, $N - 1 = 15$)

It prints: 32768

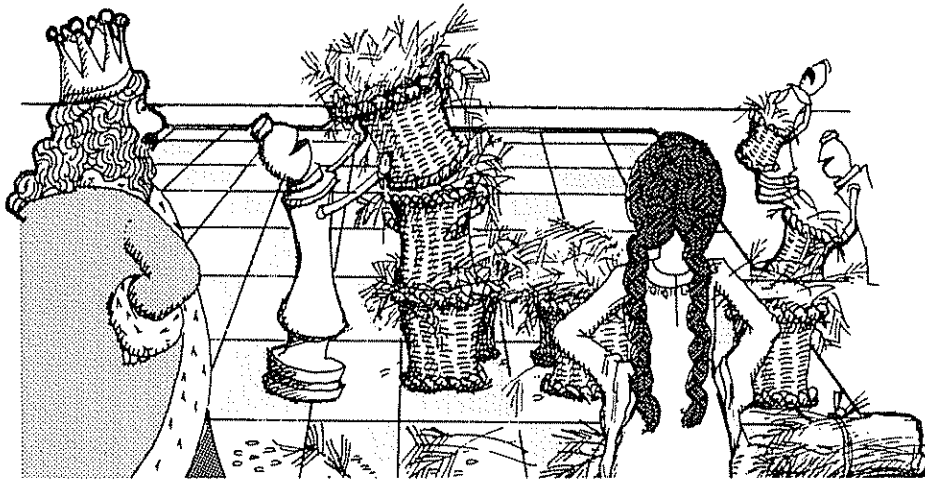
Your turn. Write a PRINT statement to find out how many grains of wheat for square number 64.

We type: _____

It prints: 9.22339E+18

Huh! What kind of number is this?
Read on.

PRINT 2163



26. In the last frame, you encountered a *very large number*: $9.22339E + 18$. Here is another way to write that number:

9,223,390,000,000,000,000

That's a lot of wheat! It was more wheat than existed in all the kingdoms everywhere at the time. The king, of course, soon discovered that he had been tricked . . . but that's another story.

Numbers such as $9.22339E + 18$ are called *floating point numbers* in computerese. Don't worry about floating numbers now. As you begin to use computers, you will encounter them now and then. If you wish to understand floating point numbers better, look at Appendix C. In the meantime, we will move on, using more familiar numbers and leaving floating point numbers to be enjoyed by those few people who . . . enjoy floating point numbers.

Yes, this is a rest frame—a place to relax with no questions to answer!

27. Keep relaxing, for you are nearing the end of this chapter. We hope that you have had a taste of adventure, the adventure of exploring the TRS-80 computer. (If you don't think of it as an adventure, watch a child play with a computer. Especially, watch the eyes!)

And now, you have a choice. You may plunge on into Chapter 3, or you may take the Self-Test that follows. We encourage you to take the Self-Test. It will help you expand on the tiny part of the adventure that you have already had and, at the same time, hone your adventuring skills for further exploration.

Whichever . . . enjoy.

SELF-TEST

Try this Self-Test to find out how much you have already learned. Good luck!

1. In this chapter, we used *direct statements*. When we type a direct statement and press **ENTER**, what does the TRS-80 do?

2. Assume that you are typing a statement into the computer, and you make a typing error. How would you correct the error?

3. How do we erase an entire line? _____

4. Write the symbols used in TRS-80 BASIC for the following arithmetic operations:

addition _____

subtraction _____

multiplication _____

division _____

power of a number _____

5. Complete the following:

We type: PLEASE DO THE HOMEWORK ON PAGE 157

It prints: _____

6. Complete the following:

We type: PRINT "PLEASE DO THE HOMEWORK ON PAGE 157"

It prints: _____

7. Complete the following:

(a) We type: `PRINT 2*3 + 4*5`

It prints: _____

(b) We type: `PRINT "2*3 + 4*5"`

It prints: _____

(c) We type: `PRINT (73 + 68 + 85)/3`

It prints: _____

8. On the surface of planet Earth, one kilogram equals 2.2 pounds (1 kg. = 2.2 lb.).

(a) An obscure giant once weighed 1,234 kilograms. Write a PRINT statement to compute his weight in pounds.

You type: _____

It prints: 2714.8

(b) Suppose that R2D2 weighs (on Earth) about 500 pounds. Write a PRINT statement to compute his weight in kilograms.

You type: _____

It prints: 227.273

9. Our utility company measures gas in *therms* and electricity in *kilowatt-hours* (KWH). They charge us at the rate of \$0.23 per therm and \$0.04 per KWH. In February, we used 116 therms of gas and 297 KWH of electricity. Write a PRINT statement to compute the amount we owe to the utility company.

You type: _____

It prints: 38.56

10. A well-tuned Honda used 22.8 gallons on a trip. At the beginning of the trip, the odometer (mileage counter) reads 12,346 miles. At the end of the trip, it read 13,193 miles. Write a PRINT statement to compute the number of miles per gallon—use the beginning and ending odometer readings and the number of gallons in your PRINT statement.

You type: _____

It prints: 37.1491

11. Watch out for this one! Write a PRINT statement to compute the number of turns per mile for a bike with a 26-inch wheel. In other words, how many times does the wheel go around while the bike goes one mile? Remember, one mile = 5,280 feet.

You type: _____

It prints: 776.89

12. Congratulations! You are the big winner on a TV game show. Your prize is selected, as follows. A number from 1 to 1,000 is chosen at random. Call it N. You then select one, and only one, of the following prizes. You have 60 seconds to make your selection.

PRIZE NO. 1: You receive N dollars

PRIZE NO. 2: You receive D dollars, where $D = 1.01^N$

Perhaps you recognize the formula for D. It is the amount you would receive if you invested \$1 at 1% interest *per day*, compounded daily for N days. Fortunately, you brought your trusty TRS-80 with you. For each of the following values of N, write a PRINT statement to help you decide which prize to take.

- (a) N = 500

You type: _____

It prints: 144.764 Take PRIZE NO. 1!

- (b) N = 789



You type: _____

It prints: 2567.58 Take PRIZE NO. 2!

All right! With \$2,567.58, you can buy three more TRS-80s!

Answers to Self-Test

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

1. The TRS-80 executes the direct statement immediately (or *directly*, if you prefer). After executing the statement, the TRS-80 prints READY, then prints the prompt (>) and the cursor (___). (frame 7)
 2. Press the  key to erase the mistake, then type the correct character. (frame 9)
 3. Hold down the **SHIFT** key and press the  key. This erases the entire line, leaving only the prompt and cursor (>___). (frame 11)
-

-
4. addition +
 subtraction -
 multiplication *
 division /
 power of number ↑ (frames 12-15, 23)
5. 7SN ERROR (The TRS-80 does not understand what is wanted.) (frame 4)
6. PLEASE DO THE HOMEWORK ON PAGE 157 (frames 5-7)
7. (a) 26 (b) $2*3 + 4*5$ (c) 75.3333 (frames 12, 19, 20)
8. (a) PRINT 1234*2.2 OR PRINT 2.2*1234 (frame 16)
 (b) PRINT 500/2.2 (frame 17)
9. PRINT .23*116 + .04*297 OR PRINT .04*297 + .23*116 (frame 18)
10. PRINT (13193 - 12346)/22.8 (frame 19)
11. PRINT 5280/(3.14*26/12) OR PRINT 5280*12/(3.14*26)
- | | |
|---|---|
| $\underbrace{\hspace{10em}}$ This is distance traveled in one turn, in <i>feet</i> | $\underbrace{\hspace{10em}}$ Inches in one mile |
|---|---|
- Note the use of parentheses in the above two ways of solving this problem.
 In the expression $5280/(3.14*26/12)$, the part inside parentheses is computed first, then divided into 5,280. For more information on how the TRS-80 does arithmetic, see Appendix C. (frame 20)
12. (a) PRINT 1.011500
 (b) PRINT 1.011789 (frames 22-25)
-

CHAPTER THREE

Basic Programs

This chapter introduces some of the BASIC statements you will use most often in your TRS-80. From here on, we can begin to write interesting programs for home, school, and personal use.

In this chapter, you will learn what the following statements and commands mean, how they work, and how to use them.

LET, INPUT, GOTO, PRINT, NEW, LIST, RUN

You will also learn how you can enter programs into the memory of the computer, then tell the computer to do what the program tells it to do. You will learn about *variables* and be able to supply values for variables used in BASIC programs.

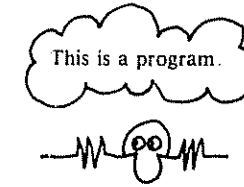
When you have finished this chapter, you will be able to:

- write short programs to fill the screen with your name;
- write programs in which values are assigned to variables by means of LET statements or INPUT statements;
- distinguish between and use numeric variables and string variables;
- write programs in which a value calculated by a BASIC expression is assigned to a numerical variable in a LET statement;
- store a BASIC program in the computer's memory;
- erase an unwanted program from the computer's memory, LIST a program currently in the computer, and RUN (execute) a program;
- edit, correct, and delete statements in a stored program;
- write programs that use the GOTO statement to repeat a portion of a program or to skip over a portion of a program;
- write PRINT statements that identify printed results with descriptive messages.

We will emphasize having fun while learning things that can be used for *both* fun and serious uses of the computer.

1. Now we are ready to *enter a program* in the memory of the TRS-80. Our program causes the TRS-80 to put Karl's name on every line on the screen. Here is the program.

```
10 CLS
20 PRINT "KARL"
30 GOTO 20
```



The above program consists of three *statements*.

This is a statement: 10 CLS

This is a statement: 20 PRINT "KARL"

This is a statement: 30 GOTO 20

In the next few frames, we will describe the program, then show you how to enter it into the memory of the TRS-80 and tell the computer to do what the program says to do.

2. Each statement in the program begins with a line number, which identifies the statement.

10 CLS
↑
Line number

20 PRINT "KARL"
↑
Line number

30 GOTO 20
↑
Line number

When you type a statement that begins with a line number, the statement is *not* executed immediately when you press the **ENTER** key. Instead, it is stored in the computer's memory for later execution. To store a statement in the computer's memory, type it with a line number and press the **ENTER** key.

Statements *without* line numbers are called *direct* statements. You learned about them in Chapter 2. Direct statements are executed immediately after you press the **ENTER** key.

(a) What happens when we type a statement that begins with a line number?

(b) What happens when we type a statement *without* a line number, then press the **ENTER** key? _____

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

- (b) The computer executes the statement immediately after we press the **ENTER** key, then "forgets" the statement. Statements without line numbers do not remain stored in the computer's memory.

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

```
10 CLS
20 PRINT "KARL"
30 GOTO 20
```

How many additional statements could be added *between* line 20 and line 30?

9 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 thirtens or fives or jumping around, go ahead!

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

Here is how we erase old programs and get the TRS-80 ready to accept a new program.

We type: `NEW` and, of course, we press the **ENTER** key.

The screen looks like this:

```
READY
>_
```

The TRS-80 has erased the portion of its memory that stores BASIC programs. It is now ready to accept a new program.

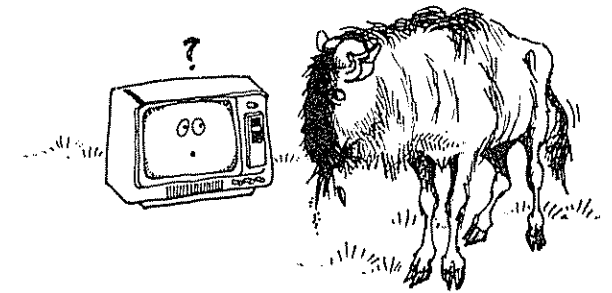
How do we erase, remove, or delete an old program from the computer's memory? _____

We type `NEW` and press the **ENTER** key. Note: If we misspell `NEW`, we may get an error message. For example:

We type: `GNU`

It prints: `?SN ERROR`

Our computer doesn't appreciate puns, but if we type `NEW` correctly and press **ENTER**, all is well.



5. Now we are ready to enter our three-line program shown in frames 1 and 3. Here it is again.

```
10 CLS
20 PRINT "KARL"
30 GOTO 20
```

Before we enter the program, we must first erase any old program by typing _____ and pressing the **ENTER** key. To enter the program, we type the first line (10 CLS) and press the **ENTER** key, then type the second line (20 PRINT "KARL") and press the **ENTER** key, then type the

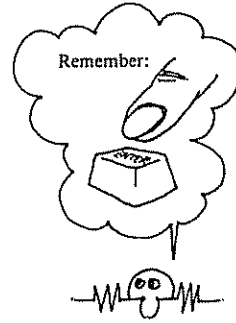
_____ and _____.

`NEW`; third line (30 GOTO 20); press the **ENTER** key.

6. You enter the program. First type **NEW** and press the **ENTER** key.


You type: **NEW**

It prints: **READY**
>_



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

You type: **10 CLS**
20 PRINT "KARL"
30 GOTO 20

If you make a typing error, correct it by using the  key. Or, if you have already pressed **ENTER**, simply retype any line in which you made an error.

The program is now stored in the memory of the TRS-80. The TRS-80 is waiting patiently for your next instruction.

No questions. Move on to the next frame.

7. Hmm . . . we wonder if the program really *is* stored in the memory. Let's find out.

First, clear the screen by pressing **CLEAR**, then pressing **ENTER**. Then, type **LIST** and press the **ENTER** key.

You type: **LIST**

It prints: **10 CLS**
20 PRINT "KARL"
30 GOTO 20
READY
>_

LIST tells the computer to print on the screen all of the program statements that are currently stored in its memory. After listing the program, the TRS-80 prints **READY** to let you know it is your turn again. If there is *no* program stored in its memory, the computer will simply print **READY**.

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

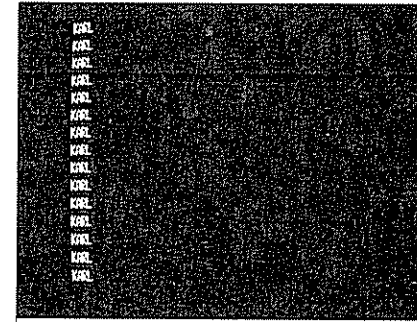
We type **LIST** and press the **ENTER** key.

The TRS-80 will then print a copy of the program in its memory on the TV screen. If there are mistakes in the program, correct them by retyping any line (10, 20, 30) that has a mistake. Then list the program again—repeat until all is well!

8. If all has gone well, our program is correctly stored in memory. (In the next frame, we will show you what happens if the program is *not* stored correctly.) Next, we will tell the TRS-80 *to do what the program tells it to do*. We will tell the TRS-80 to RUN the program.

Type RUN and press **ENTER**.

As you can see, the TRS-80 quickly prints Karl's name everywhere down the left side of the screen. Here is a snapshot of the screen.



Watch the bottom line of the screen. The TRS-80 is continuing to print Karl's name on the bottom line. Each time it does, all the other KARLs are pushed up (scrolled) one place and the top KARL is "pushed off" the screen. This happens so fast, however, that you can't really see what is happening!

Now, how do we *stop* the computer? Find the **BREAK** key. Press it. Congratulations! You have stopped the computer. The bottom portion of the screen looks something like this:

```

KARL
KARL
BREAK IN LINE 20
READY
>_

```

You stopped the TRS-80
while it was doing line 20.

How do we stop a "runaway" TRS-80?

- _____ (a) We turn the computer off by pressing the **ON-OFF** power switch.
- _____ (b) We pull the power cord that connects the power supply to the 110-volt outlet.
- _____ (c) We yell "STOP" in a loud voice several times.
- _____ (d) We press the **BREAK** key on the keyboard.

(a), (b), and (d) will work. However, (a) and (b) cause the TRS-80 to erase its memory. So the program would be erased. The best choice is (d)—that is, to stop the computer, press the **BREAK** key.

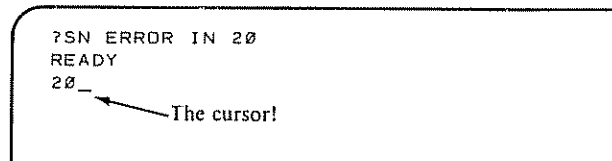
9. Well, maybe the program *wasn't* stored correctly in the TRS-80's memory. Suppose that we had made a typing mistake and the program was stored incorrectly, as follows:

```
10 CLS
20 PTINT "KARL"
30 GOTO 20
```

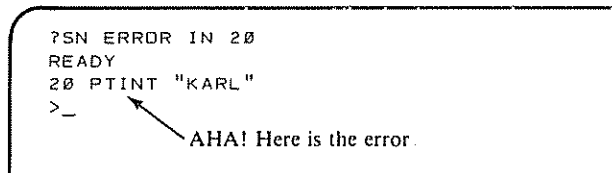


But, alas, we did not notice the error. We typed RUN and pressed the **ENTER** key. Here is what happened.

The TRS-80 printed all of this.



Now, press the **ENTER** key. The TRS-80 will print the rest of line 20 so we can look for the error!



How can we correct the error? _____

We retype and enter line 20 *correctly*. That is, we type 20 PRINT "KARL"; and press the **ENTER** key. Then, we **LIST** the program to make sure the correct line 20 is in memory. If so, then **RUN** the program.

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

10. Look at the error message in the preceding frame.

?SN ERROR IN 20
Line number where the error occurred.

Suppose that the program has been stored incorrectly, as shown below.

```
10 CL
20 PRINT "KARL"
30 GOTO 20
```

Oops! Should have been 10 CLS.

We type RUN and press **ENTER**. Show what the TRS-80 prints.

We type:

RUN

It prints:


```
?SN ERROR IN 10
READY
10_
```

(Syntax error in line 10)

If you now press **ENTER**, the TRS-80 will print the rest of line 10 so that you can find the error.

Remember: To correct the error in line 10, simply retype line 10 . . . correctly, please! The new, all new, *correct* line 10 will replace the old, incorrect line 10 in the TRS-80's memory.

11. How does the program work? The statement:

```
10 CLS
```

tells the computer to clear the screen. The statement:

```
20 PRINT "KARL"
```

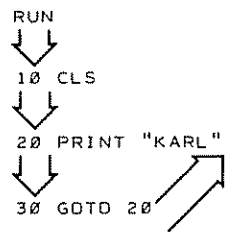
tells the computer to print the word KARL on the screen. The statement:

```
30 GOTO 20
```

tells the computer to go to line 20 and continue.

When we type RUN and press **ENTER**, the computer begins executing the program, beginning with the smallest line number. The computer executes line 10, then line 20, then line 30, then line 20, then line 30, then line 20, then line 30, and so on—until you press the **BREAK** key, which stops the computer.

Follow the arrows.



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

Here is a slightly different program. Draw arrows to show the order in which things are done.

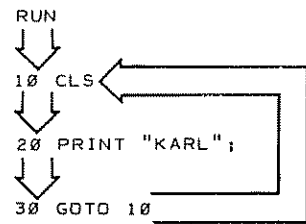
```

RUN

10 CLS

20 PRINT "KARL"

30 GOTO 10
  
```



This program causes Karl's name to blink rapidly in the upper left corner of the TV screen. Why? Because CLS clears the screen and *also* moves the cursor to its "home" position in the upper left corner of the screen. Difficult to explain, but easy to see when you RUN this program on your TRS-80!

12. We assume that our program of the last few frames is still in memory. If not, please enter it into memory as shown in frame 6. Then, **CLEAR** the screen and **LIST** the program.

You type: LIST

It prints: 10 CLS
20 PRINT "KARL"
30 GOTO 20

Now, we want to change line 20, but *only* line 20. We do *not* (repeat, do *not*) type NEW. Instead, we do this.

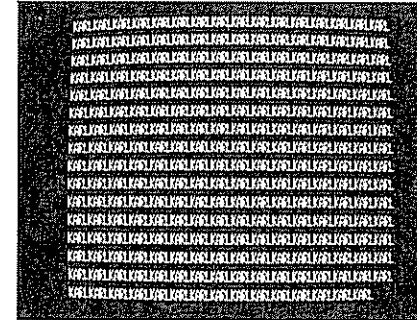
We type: 20 PRINT "KARL"; (semicolon at the end)

We type: LIST

| | | |
|------------|------------------|-------------|
| It prints: | 10 CLS | Old line 10 |
| | 20 PRINT "KARL"; | New line 20 |
| | 30 GOTO 20 | Old line 30 |

Our 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 and press the **ENTER** key. This is what you see:



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

Causes the TRS-80 to print across the screen from left to right. When the TRS-80 gets to the right end of the screen, it simply continues on the next line. Without the semicolon, the TRS-80 prints whatever is between quotation marks in the PRINT statement, then moves immediately to the left side of the next line. We will explain this in more detail later in the book.

13. Try these variations for line 20. For each new line 20, RUN the program to see what happens, then press **BREAK**, enter the next line 20, RUN it, and so on.

20 PRINT "MARY " ;

Put *one* space after MARY. Watch MARY move to the right.

20 PRINT "JUDY " ;

Put *three* spaces after JUDY. Watch JUDY move to the left.

20 PRINT "GANDALF " ;

Put one space after GANDALF. GANDALF moves neither left nor right.

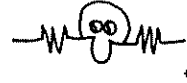
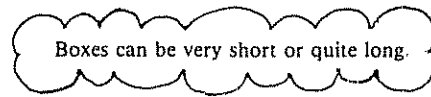
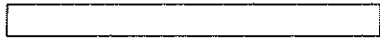
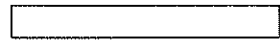
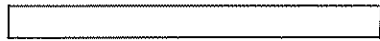
20 PRINT "A " ;

Put four spaces after A. Watch A move to the right. Also, watch the left edge of the screen or the right edge to see A do other interesting things!

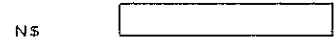
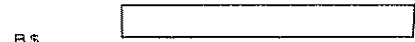
Illusion: That's what is happening. The TRS-80 is simply printing what is between quotation marks across the screen, beginning at top left. Look at the bottom line of the screen. When the screen fills, the TRS-80 keeps right on printing new lines across the bottom of the screen. Each new line pushes all the old lines up one place (computer people call it "scrolling"). It happens so fast, though, that you see everything on the screen moving in some direction.

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

14. Imagine that, inside the computer, there are a bunch of little boxes. Each box is a place in the computer's memory in which we can store information.



We can give a box a name consisting of a letter followed by a dollar sign (\$). For example, below are boxes called A\$, B\$, N\$, X\$, and Z\$.



Into any box, we can put a *string*. A string is simply any bunch of keyboard characters, typed one after the other. For example:

A string can be a name: KARL

A string can be a number: 123456789

A string can be gibberish: AB#\$%JFDZ?*

Almost any keyboard character can be part of a string. One important exception is the double quotation mark ("). It *can't* be part of a string. Instead, quotation marks are sometimes used to *enclose* a string, as follows.

"KARL"

The string KARL is enclosed in quotation marks.

Below are some boxes. We have already stored strings in some of these boxes. For example, KARL is in box A\$ and 415 323 6117 is in box B\$.

| | | | | | |
|-----|---|-----|---|-----|--|
| A\$ | <input type="text" value="KARL"/> | B\$ | <input type="text" value="415 323 6117"/> | C\$ | <input type="text" value="ABC"/> |
| D\$ | <input type="text" value="WADGETS: 23"/> | E\$ | <input type="text" value="\$\$\$\$\$\$\$"/> | F\$ | <input type="text" value="MENLO PARK CA"/> |
| G\$ | <input type="text" value="BROWNSTONE, IRENE"/> | H\$ | <input type="text" value=""/> | | |
| Z\$ | <input type="text" value="TOMORROW IS YOUR MOTHER'S BIRTHDAY"/> | | | | |

- What is in box C\$? _____
- What is in box D\$? _____
- Which box contains seven dollar signs? _____
- What is in box H\$? _____
- Which box has an important message for people who love their mothers?

- Which box has a person's name with last name first, followed by first name? _____
- Which box has the name of the small city in which this book was written?

-
- ABC
 - WADGETS: 23
 - E\$
 - Nothing—this box is "empty."
 - Z\$
-

- (f) G\$
- (g) F\$

15. A box will be as short or as long as necessary to hold the string we want to put into it. Actually, there is a limit on string length. We will tell you about it soon. In the meantime, you may use strings with *up to* 50 characters. Here is how we stuff a string into a box, deep down inside the computer. To tell the computer to put KARL into box A\$, we use a LET statement, without a line number, like this:

We type: LET A\$ = "KARL"

It prints: READY
>_



The TRS-80 has put the string KARL into box A\$. Note how we told the computer to do this.

> LET A\$ = "KARL"

Let's find out what is in box A\$. To do this, we will tell the TRS-80 to print what is in A\$.

We type: PRINT A\$

It prints: KARL
READY
>_

To tell the computer to print the string that is in A\$, we type:

> PRINT A\$

Complete the following:

We type: LET Z\$ = "MENLO PARK, CA 94025"

It prints: _____

We type: PRINT Z\$

It prints: _____

READY

>_

MENLO PARK, CA 94025

READY

>_

16. The names A\$, B\$, C\$, and so on, which identify boxes, are called *variables*. Since these boxes can hold strings, these variables are called *string variables*.

A\$ is a string variable,
B\$ is a string variable,
C\$ is a string variable,
and so on.

A string variable is a _____ followed by a
_____. A string variable is the label, or name, of a box
that can hold a _____.

letter; dollar sign (\$); string

Later, you will learn about numeric variables, which can store *only* numbers. We will use numeric variables when we want to do arithmetic with numbers.

17. A\$, B\$, C\$, . . . , Z\$ are string variables. The string in box A\$ is called the *value of A\$*; the string in box B\$ is the *value of B\$*; the string in box C\$ is the *value of C\$*; and so on.

We use the LET statement to tell the computer to "put a string into a box" or, more technically, "assign a string to a string variable."

We type: LET C\$ = "GREEN" ← Put GREEN into box C\$.

We type: PRINT C\$ ← Print the contents of box C\$.

It prints: GREEN

In the above, the variable is _____ and the string
value assigned to it is _____.

C\$ = "GREEN"

IMPORTANT NOTICE! The word LET is optional; it can be omitted.

Instead of: LET C\$ = "GREEN"

We can type: C\$ = "GREEN"

Some people use LET
Some people don't



A box can hold *one* string at a time. When the computer puts a string in a box, it first *erases* the previous contents of the box.

18. Complete the following:

(a) We type: LET M\$ = "STAR WARS"

We type: PRINT M\$

It prints: _____

(b) We type: M\$ = "STAR WARS"

We type: PRINT M\$

It prints: _____

Remember, a variable can have only one value at a time. When we assign a value to a variable, the computer automatically erases any previous value of that variable.

(c) We type: A\$ = "DOG"

We type: A\$ = "CAT"

We type: PRINT A\$

It prints: _____

(d) We type: F\$ = "ROSE"

We type: F\$ = "DAFFODIL"

We type: PRINT F\$

It prints: _____

Here is a tricky one. Can you figure it out without peeking at the answer?

(e) We type: P\$ = "JUPITER"

We type: Q\$ = P\$

We type: PRINT Q\$

It prints _____

21. Now we will tell you how to put strings in boxes—the easy way—by using a new statement called the INPUT statement.

```
10 CLS
20 INPUT A$ ← This is an INPUT statement.
30 PRINT A$;
40 GOTO 30
```

We stored the above program in the TRS-80, then typed RUN. Here is what happened.

The computer printed a question mark and the cursor.

?_

The TRS-80 is doing the INPUT statement. The statement:

```
20 INPUT A$
```

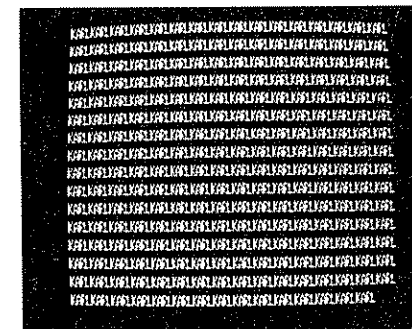
tells the TRS-80 to:

- (a) type a question mark;
- (b) turn on the cursor;
- (c) wait for someone to type a string. When you type a string and press the **ENTER** key, the TRS-80 puts the string into box A\$ and goes on to the next statement in line number sequence.

Let's cooperate with our ever-patient computer and type in a string. To keep things familiar, we will type Karl's name and then press the **ENTER** key.

We type: "KARL" (and press **ENTER**)

It prints:

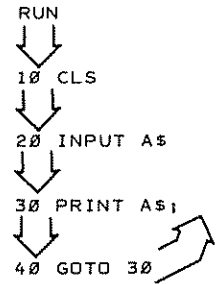


```
KARL
?_
```

Well, that looks familiar! (If not, see frame 12.) How do we stop the computer? _____

We press the **BREAK** key. Yes, occasionally we remind you that you can stop, interrupt, or "break into" what the computer is doing by pressing the **BREAK** key.

22. How does the program work? As usual, follow the arrows.



At this point, we must type a string and press **ENTER**. The TRS-80 puts *our* string in A\$, then moves on to line 30.

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

We now have three ways (programs) to fill the screen with multiple copies of a string.

Program 1

```

10 CLS
20 PRINT "KARL";
30 GOTO 20
  
```

Program 2

```

10 CLS
20 LET A$ = "KARL"
30 PRINT A$;
40 GOTO 30
  
```

Program 3

```

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

In the above three programs, we see examples of five types of statements. What are the five types of statements? We have named *one* type to help you get started.

- (a) CLS
- (b) _____
- (c) _____
- (d) _____
- (e) _____

- (a) CLS
- (b) PRINT
- (c) GOTO
- (d) LET
- (e) INPUT

Remember, the word **LET** may be omitted.

In the rest of this book, you will learn perhaps another fifteen statement types. So you are well on your way. In the rest of this chapter, you will learn more about how to use the INPUT statement.

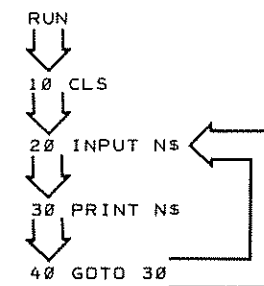
23. The following program will help you learn more about the INPUT statement. You can use it to experiment!

```
NEW
10 CLS
20 INPUT N$
30 PRINT N$
40 GOTO 20
```

This program simply prints out (once) whatever you type in for the value of N\$.

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

```
RUN
10 CLS
20 INPUT N$
30 PRINT N$
40 GOTO 20
```



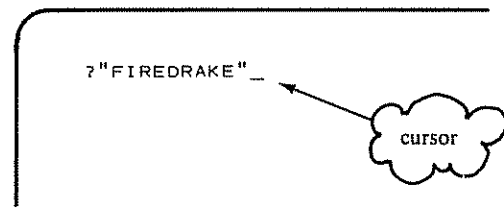
In following this program, the TRS-80 will always return to the INPUT statement, print a question mark, turn on the cursor, and wait for the next value of N\$.

24. We stored the program (frame 23) in the TRS-80, then typed RUN and pressed the ENTER key. Here is what happened.

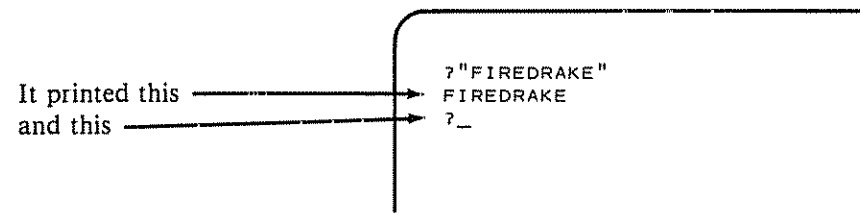
The computer printed a question mark and the cursor

?_

We then typed the string "FIREDRAKE" (including the quotation marks).



Then we pressed the **ENTER** key.



The computer printed the string that we typed, *enclosed in quotation marks*, following the question mark.

- (a) Why did the computer print the question mark? _____
- (b) After printing FIREDRAKE, why did the computer print another question mark? _____

- (a) It was executing the INPUT N\$ statement in line 20. When we typed "FIREDRAKE" and pressed the **ENTER** key, the computer put FIREDRAKE into box N\$ or, more technically, assigned FIREDRAKE as the value of N\$.
- (b) After printing the value of N\$ (line 30), the TRS-80 obeys the GOTO 20 statement in line 40 and goes to line 20, which is the INPUT statement, and executes it.

25. We continued entering strings in response to question marks. Study these examples carefully.

What? No quotation marks?
 Hmm . . . seems to be OK.
 Five spaces before FIREDRAKE.
 The computer ignored our spaces.
 Let's try quotation marks.
 Aha! It printed the spaces.
 Home, sweet, home.
 Huh? What happened?
 Well, we got part of it.
 Let's try quotation marks.
 OK!
 And so on. The computer
 waits for the next string.

```
? "FIREDRAKE"
FIREDRAKE
?FIREDRAKE
FIREDRAKE
?    FIREDRAKE
FIREDRAKE
?"    FIREDRAKE"
    FIREDRAKE
?COMPUTERTOWN, USA
?EXTRA IGNORED
COMPUTERTOWN
?"COMPUTERTOWN, USA"
COMPUTERTOWN, USA
?_
```

Our motto is:

If in doubt, use quotation marks when you enter a *string* as a value of a *string variable*.

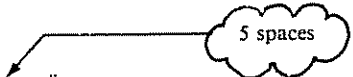
Without quotation marks, the computer will:

- Ignore leading space, but accept *trailing** spaces. We couldn't show that because trailing spaces are invisible.
- Do funny things if a string has a comma.

Never mind. Plunge onward. Most, if not all, will be revealed. Or you can *experiment* and find out for yourself many of the answers to the question: What if?

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

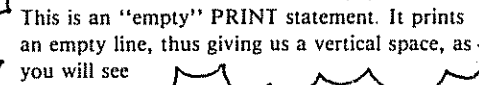
*This string has trailing spaces: "FIREDRAKE "



5 spaces

26. Look at the information printed on the screen in the example in the preceding frame. Crowded, isn't it? Below is a slightly modified version of the program.

```
10 CLS
20 INPUT N$
30 PRINT N$
35 PRINT
40 GOTO 20
```



This is an "empty" PRINT statement. It prints an empty line, thus giving us a vertical space, as you will see

Using the above program, our examples of the preceding frame would look like this on the screen:

```
? "FIREDRAKE"
FIREDRAKE
← An empty line
? FIREDRAKE
FIREDRAKE
← An empty line
?   FIREDRAKE
FIREDRAKE
← An empty line
? and so on . . .
```

The empty lines, caused by line 35, make the stuff on the screen easier to read. Each example is separated from the one above or below it by a vertical line space.

27. The INPUT statement, as we have known it, tells the computer to put a question mark on the screen, then wait for the user to answer. Wouldn't it be nice if, instead of printing just a question mark, the computer would put some informative words on the screen, so that the user knows what the computer wants? The following program has an "enhanced" INPUT statement that is more informative.

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

Let's examine the "enhanced" INPUT statement. The statement:

```
20 INPUT "WHAT IS YOUR NAME"; N$
```

tells the TRS-80 to:

- (a) print WHAT IS YOUR NAME on the screen;
- (b) print a question mark;
- (c) turn on the cursor; and
- (d) wait for someone to type in a string and press the **ENTER** key.

Look at the statement carefully. It consists of:

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

One more time:

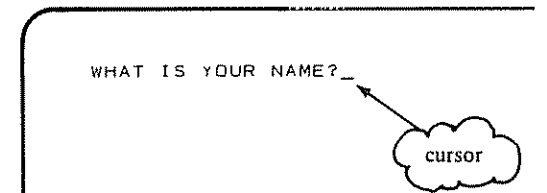
```
(a)  (b)  (c)      (d)      (e)(f)(g)
  ↓    ↓    ↓        ↓        ↓ ↓ ↓
20  INPUT "WHAT IS YOUR NAME" ; N$
```

But where does the question mark come from? _____

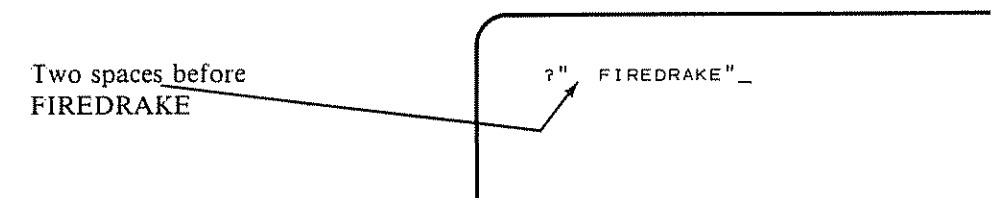
The question mark is always an automatic part of an INPUT statement. When an INPUT statement includes a string, the question mark is printed at the right end of the string.

28. As usual, to find out what a program does, we type NEW, press **ENTER**, then carefully type the program into the TRS-80. Let's assume that the program of the preceding frame is now in the memory of the TRS-80. So, we type RUN and press **ENTER**.

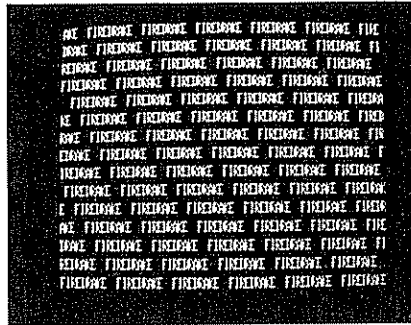
The computer prints:



If you type your name and press the **ENTER** key, the computer will print it "everywhere" on the screen. If you wish to include spaces in front of your name, remember to use quotation marks. For example:



Now, when Firedrake presses **ENTER**, his name will appear everywhere on the screen, but always preceded by two spaces. It might look like this:



29. You already know about *string variables*. In case you have forgotten, review frames 14 through 17. Now let's talk about *numeric variables*. Numeric variables are names of boxes that can hold only *numbers*. Any letter of the alphabet can be a numeric variable.

These are numeric variables: A, B, C, . . . , Z

These are string variables: A\$, B\$, C\$, . . . , Z\$

What is the difference between a numeric variable and a string variable? _____

A string variable *always* ends with a dollar sign (\$). A numeric variable *never* ends with a dollar sign. String variables may have any string as a value. Numeric variables may have *only* numbers as values. And values of numeric variables must be typed *without* quotation marks, as you will see in the next few frames.

30. We can use the LET statement to instruct the computer to "put a number in a box." To say it more technically, we are assigning a numerical *value* to a *variable*.

We type: LET A = 7

Put 7 into box A.

We type: PRINT A

Print the contents of box A.

It prints: 7

In this program the *variable* is A and the *value* assigned to it by the LET statement is 7.

Complete the following:

(a) We type: `LET X = 23`
`PRINT X`

It prints: _____

(c) We type: `LET A = 1`
`LET A = 2`
`PRINT A`

It prints: _____

(b) We type: `LET Z = -1`
`PRINT Z`

It prints: _____

(d) We type: `LET D = 7`
`LET W = D`
`PRINT W`

It prints: _____

-
- (a) 23
(b) -1
(c) 2
(d) 7 (7 is in both W and D)

NOTE: The word LET is optional. Instead of `LET X = 23`, we can write `X = 23`.

As you read through this book, you will see less and less of LET. If you read other books and magazines, however, you may occasionally encounter it. In *your* programs, we encourage you to *not* use LET.

31. Write a LET statement to assign the value 3.14 to P and a PRINT statement to print the value of P.

We type: _____

We type: _____

It prints: 3.14

`LET P = 3.14` OR `P = 3.14`
`PRINT P`

32. Once we have assigned a value to a variable, that variable can be used in mathematical expressions.

We type: `LET A = 7`
`LET B = 5`

A

We now have 7 in box A and 5 in box B.

B

We type: `PRINT A + B`

It prints: 12

Since $A=7$ and $B=5$, $A+B=7+5=12$.

We type: PRINT A - B

It prints: 2 Since A=7 and B=5, A-B=7-5=2.

Your turn. Complete the following:

We type: PRINT A*B

It prints: _____

We type: PRINT A/B

It prints _____

35
1.4

33. We can also use the INPUT statement to assign a value to a numeric variable.

NEW

```
10 CLS
20 INPUT A } ← A is a numeric variable.
30 PRINT A
40 PRINT
50 GOTO 20
```

We entered the above program and ran it. Here is what happened on the screen.

```
?258 ← We typed 258 with no quotation marks.
 258   It printed 258, the value of A.
? -6 ← We typed -6 with no quotation marks.
 -6   It printed -6, the value of A.
?12.95 ← We typed 12.95 with no quotation marks.
12.95 It printed 12.95, the value of A.
?"73" ← We typed 73 with quotation marks.
?REDO It rejected our "73"
?_    and is waiting for our next try.
```

When we enter a number that is to be stored as the value of a *numeric variable*, we (should/should not) _____ enclose it in quotation marks.

should not

However, if we should enter a number as the value of a *string* variable, it is OK to enclose it in quotation marks. Remember though, you *can't do arithmetic with strings*.

34. Remember our bicycle program from Chapter 2? If not, review frame 18 in Chapter 2.

Below is a program to compute the distance traveled in one turn of a wheel of diameter D.

NEW

```
10 CLS
20 INPUT "WHEEL DIAMETER"; D
30 PRINT 3.14*D
40 PRINT
50 GOTO 20
```

```
RUN
WHEEL DIAMETER? 20      We typed the wheel diameter.
62.80                  It printed the distance traveled.

WHEEL DIAMETER? 26      We typed the wheel diameter.
81.64                  It printed the distance traveled.

WHEEL DIAMETER? 27      We typed the wheel diameter.
84.78                  It printed the distance traveled.

WHEEL DIAMETER?        It is waiting for more.
```

What happens when the computer executes line 40 in the above program? _____

The computer prints an empty line on the screen. This separates each example from the previous example. See frame 26 for a discussion of empty PRINT statements.

35. What the computer wants us to type is clearly identified by the string WHEEL DIAMETER in the INPUT statement. We can use the PRINT statement to identify the answer that the computer prints. Change line 30 to the following.

```
30 PRINT "DISTANCE IN ONE TURN IS" 3.14*D
```

Did you make the change? Now list the program.

LIST

```
10 CLS
20 INPUT "WHEEL DIAMETER" ; D
30 PRINT "DISTANCE IN ONE TURN IS" 3.14*D
40 PRINT
50 GOTO 20
```

Yup, there it is



Let's RUN it.

```

RUN
WHEEL DIAMETER? 20
DISTANCE IN ONE TURN IS 62.80

WHEEL DIAMETER? 26
DISTANCE IN ONE TURN IS 81.64

WHEEL DIAMETER? 27
DISTANCE IN ONE TURN IS 84.78

WHEEL DIAMETER?
    
```

Apparently, the statement 30 PRINT "DISTANCE IN ONE TURN IS" 3.14*D tells the TRS-80 to:

- (a) _____, then
- (b) _____

- (a) print the string: DISTANCE IN ONE TURN IS
- (b) compute and print the value of 3.14*D

NOTE: In some versions of BASIC, it is necessary to put a semicolon between the two things in the PRINT statement, as here:

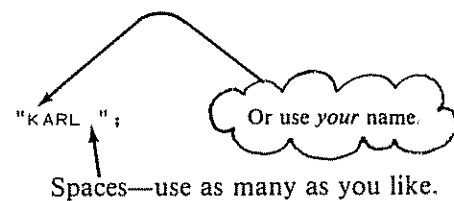
```
30 PRINT "DISTANCE IN ONE TURN IS" ; 3.14*D
```

The TRS-80 will accept either way of doing it.

36. We will end this chapter with some teasers, promises of things to come. You now know how to enter a program. You can do this *even if you don't understand the program*. So here are some programs that you don't understand now, but soon will understand. Go ahead. Give them a try. Type in these programs and RUN them.

Program Chaos

```
10 CLS
20 PRINT @RND(1023), "KARL ";
30 GOTO 20
```



Look for this in Chapter 5.

Program Firefly

```
10 CLS
20 PRINT @RND(1023), "*" ;
30 GOTO 10
```

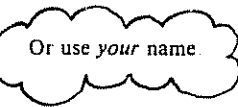
Look for this in Chapter 5.

Program Blink

```

10 CLS
20 PRINT @473, "KARL" ;
30 FOR Z = 1 TO 500 : NEXT Z
40 CLS
50 FOR Z = 1 TO 500 : NEXT Z
60 GOTO 20

```



Or use *your* name.

Look for this in Chapter 4.

Program Starfall

```

10 CLS
20 OVER = RND(127)
30 DOWN = RND(47)
40 SET (OVER, DOWN)
50 GOTO 20

```

Look for this in Chapter 5.

Program ESP

```

100 CLS
110 PRINT "DO YOU HAVE ESP? LET'S FIND OUT."
120 PRINT "GUESS MY NUMBER, IF YOU CAN!"
130 PRINT
140 PRINT "I'M THINKING OF A NUMBER, 1 OR 2."
150 SECRET = RND(2)
160 PRINT
170 INPUT "YOUR GUESS" ; GUESS
180 IF GUESS <> SECRET THEN PRINT "SORRY, THAT'S NOT IT."
190 IF GUESS = SECRET THEN PRINT "YES! YOU GOT IT!"
200 PRINT "OK, LET'S DO IT AGAIN."
210 GOTO 130

```

This is a very simple game. Again, we caution you, do not put quotation marks around your numbers if you play this game. In the above program, we use two numeric variables called SECRET and GUESS. Yes, a numeric variable can be longer than a single letter.

SECRET is a *numeric variable*, the name of the box that holds the computer's secret number (1 or 2).

GUESS is a *numeric variable*, the name of the box that holds your guess.

You have now encountered the mysterious RND several times in the last few programs. You might want to speculate about what this unpredictable creature does. RND will appear in all its glory (and be explained) in Chapter 5.

SELF-TEST

Congratulations! You have completed another chapter. Now try this Self-Test.

1. Before entering a program, we usually type NEW and press ENTER. Why?

2. How do we enter a program in the computer's memory? _____

3. How do we tell the computer to type a complete listing of a program stored in its memory? _____

4. How do we tell the computer to execute a program stored in its memory?

5. In a stored program, each line begins with a _____

6. We store the following program in memory and RUN it.

```
100 PRINT "I WILL NOT CHEW BUBBLE GUM IN CLASS."
110 GOTO 100
```

How do we stop the computer? _____

7. Which program would produce the RUN shown below?

Program A

```
10 CLS
20 PRINT "HA HA ";
30 GOTO 20
```

Program B

```
10 CLS
20 PRINT "HA HA "
30 GOTO 20
```

```
HA HA HA HA HA HA HA HA HA HA HA
A 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 HA HA HA HA HA HA HA
A HA HA HA HA HA HA HA HA HA HA HA
  HA HA HA HA HA HA HA HA HA HA
```

and so on . . .

8. Write a program to "teach the computer to cry." Your program, when RUN, should cause the screen to fill up with BOO HOO.

9. The following program has *already been stored*. There is an error in line 20. How can we correct the error? Remember, the program is already stored.

```

10 CLS
20 PRINT "R2D2 LIVES! " ;
30 GOTO 20

```

We forgot quotation marks.
They go just before the R2D2.

10. Show the RUN for the following program.

```

10 CLS
20 PRINT "  XXXXX"
30 PRINT " X    X"
40 PRINT "(X @ @ X)"
50 PRINT " X V X"
60 PRINT " X < > X"
70 PRINT " X  X"
80 PRINT "   XXX"

```

RUN

11. Show the RUN for the following program.

```

10 CLS
20 A$ = "  XXXXX"
30 B$ = " X    X"
40 C$ = "(X @ @ X)"
50 D$ = " X V X"
60 E$ = " X    X"
70 F$ = " X  X"
80 G$ = "   XXX"
90 PRINT A$
100 PRINT B$
110 PRINT C$
120 PRINT D$
130 PRINT E$
140 PRINT F$
150 PRINT G$

```

RUN

12. Complete the RUN of the following program.

```

10 CLS
20 INPUT J$
30 PRINT J$
40 PRINT J$
50 PRINT J$
RUN
?"WHAT I TELL YOU THREE TIMES IS TRUE."

```

13. Complete the RUN on the line provided.

```
10 CLS
20 INPUT "A = "; A
30 INPUT "B = "; B
40 PRINT "A + B = " A + B
```

RUN

```
A = ? ?
B = ? ?
```

14. Write a program to help us compute our utilities bill. Use .23 as the price per THERM (T) of gas and .04 as the price per KWH (K) of electricity. The amount (A) that we owe is computed by the following formula.

$$A = .23 * T + .04 * K$$

A RUN for such a program might look something like this:

```
RUN
THERMS?116
KWH?297
YOU OWE 38.56
```

15. The U.S. is going metric, so here is a metric problem. Write a program to convert feet and inches to centimeters, as indicated by the following RUN.

RUN

```
FEET = ? 5
INCHES = ? 11
CENTIMETERS = 180.34
```

```
FEET = ? 3
INCHES = ? 0
CENTIMETERS = 91.44
```

```
FEET = ?
```

We enter, as requested, the number of feet, and the number of inches. The computer computes and prints the number of centimeters.

And so on.

16. Write a program to convert temperatures, given in degrees Celsius, to degrees Fahrenheit, using the following formula.

$$F = \frac{9}{5}C + 32$$

A RUN of your program should look like this.

```

RUN
YOU ENTER DEGREES CELSIUS.
I WILL PRINT DEGREES FAHRENHEIT.

DEGREES CELSIUS? 0
DEGREES FAHRENHEIT = 32

DEGREES CELSIUS? 100
DEGREES FAHRENHEIT = 212

DEGREES CELSIUS? 37
DEGREES FAHRENHEIT = 98.6

DEGREES CELSIUS?

```

And so on.

17. Congratulations! You are the big winner on a TV show. Your prize is selected as follows. A number from 1 to 1,000 is chosen at random. Call it N. You then select one, and only one, of the following prizes. You have 60 seconds to make your selection.

PRIZE NO. 1: You receive N dollars

PRIZE NO. 2: You receive D dollars, where $D = 1.01^N$

Perhaps you recognize the formula for D. It is the amount you would receive if you invested \$1 at 1% interest per day, compounded daily for N days. The question, of course, is: For a given value of N, which prize do you take, PRIZE NO. 1 or PRIZE NO. 2? Write a program to help you decide. A RUN might look like this.

```

RUN

N = ? 100
PRIZE NO. 1 = 100
PRIZE NO. 2 = 2.70478
Take PRIZE NO. 1

N = ? 500
PRIZE NO. 1 = 500
PRIZE NO. 2 = 144.764
Take PRIZE NO. 1

N = ? 1000
PRIZE NO. 1 = 1000
PRIZE NO. 2 = 20956.7
Take PRIZE NO. 2

N = ?
And so on.

```

Answers to Self-Test

The numbers in parentheses refer to the frames in the chapter where the topic is discussed. You may wish to refer to these for quick review.

1. This is to erase, or remove, any old program that might be in the computer's memory. If we *don't* do this, statements of an old program might be intermingled with statements of the new program, thus causing mysterious and unpredictable behavior when we try to RUN the new program. (frame 4)
2. We type each line of the program on the keyboard. After each line, we press the **ENTER** key. (frames 5, 6)
3. Type LIST and press the **ENTER** key. (frame 7)
4. Type RUN and press the **ENTER** key. (frame 8)
5. line number (frame 2)
6. Press the **BREAK** key. (frame 8)
7. Program A will produce the RUN as shown. Program B will cause HA HA to print down the left side of the screen, but not all over the screen. (frame 12)
8.


```
10 CLS
20 PRINT "BOO HOO ";
30 GOTO 20
```

 (frame 12)
9. Retype line 20 as follows.


```
20 PRINT "R2D2 LIVES! "
```

 The new, correct line 20 will replace the old, incorrect line 20 in the memory. (frames 6, 9)
10.


```
XXXXX
X      X
(X @ @ X)
X V X
X < > X
X      X
XXX
```

 (frames 17-20)
11. Same as the result from question 10. The "face" is defined in lines 20 through 80, then printed by lines 90 through 150. (frames 17-20)
12.


```
RUN
?"WHAT I TELL YOU THREE TIMES IS TRUE." ← We typed.
WHAT I TELL YOU THREE TIMES IS TRUE.
WHAT I TELL YOU THREE TIMES IS TRUE. ← It printed.
WHAT I TELL YOU THREE TIMES IS TRUE.
```

 (frames 21, 23)

13. A + B = 12 (frame 35)
14. 10 CLS
20 INPUT "THERMS" ; T
30 INPUT "KWH" ; K (frames 27, 28, 34, 35)
40 A = .23*T + .04*K
50 PRINT "YOU OWE" A
15. 10 CLS
20 INPUT "FEET = " ; F
30 INPUT "INCHES = " ; I (frames 27, 28, 34, 35)
40 C = (F*12 + I)*2.54
50 PRINT "CENTIMETERS = " ; C
60 PRINT
70 GOTO 20
16. 10 CLS
20 PRINT "YOU ENTER DEGREES CELSIUS."
30 PRINT "I WILL PRINT DEGREES FAHRENHEIT."
40 PRINT (frames 27, 28, 34, 35)
50 INPUT "DEGREES CELSIUS" C
60 F = (9/5)*C + 32
70 PRINT "DEGREES FAHRENHEIT = " F
80 GOTO 40
17. 10 CLS
20 INPUT "N = " ; N (frames 27, 28, 34, 35)
30 PRINT "PRIZE NO. 1 = " N
40 PRINT "PRIZE NO. 2 = " 1.01*N
50 PRINT
60 GOTO 20
-

CHAPTER FOUR

FOR-NEXT Loops

In this chapter, you will explore the wonderful ways of the FOR-NEXT loop. The FOR-NEXT loop takes over with automatic counting for all your repeatable tasks. You decide where to start and where to stop—it does the rest. You will learn the FOR-NEXT loop from the inside out and top to bottom.

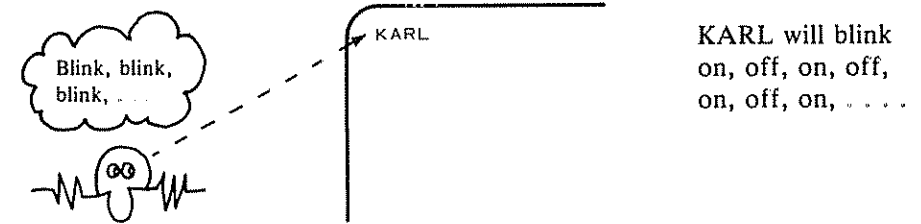
After completing this chapter, you will be able to:

- use the FOR-NEXT loop as a time delay;
- use the time delay as a *subroutine*;
- use the GOSUB statement to use, or *call*, a subroutine;
- use the RETURN statement to return from a subroutine;
- use multiple statements per line;
- position information on the screen with the PRINT @ statement;
- use the FOR-NEXT loop for repeatable parts of a program (other than time delays);
- perform mathematical feats with the FOR-NEXT loop.

1. The following program will cause Karl's name to blink on, off, on, off, on . . . and so on, until someone presses the **BREAK** key. Try it.

| <i>The Program</i> | <i>Comments about the Program</i> |
|------------------------|-------------------------------------|
| 100 CLS | Clear the screen. |
| 110 PRINT "KARL" | Print the word KARL. |
| 120 FOR Z = 1 TO 500] | Something new! Go ahead and use it. |
| 130 NEXT Z] | We will explain later. |
| 140 CLS | Clear the screen. |
| 150 FOR Z = 1 TO 500] | Here it is again. Plunge bravely |
| 160 NEXT Z] | onward—all will be revealed. |
| 170 GOTO 100 | |

Type NEW, enter the above program, and RUN it.



- Press **BREAK** to stop the computer.
- Change line 110 to: 110 PRINT “_____”
- RUN it again. *Your* name will blink on, off, on, off, on, off,

Try other blinking messages, perhaps one of the following:

```
110 PRINT "TOMORROW IS YOUR MOTHER'S BIRTHDAY"
110 PRINT "MAN THE PUMPS-THE BOAT IS SINKING!"
110 PRINT "MAY THE FORCE BE WITH YOU"
110 PRINT "LIVE LONG AND PROSPER"
110 PRINT "TAKE A DRAGON TO LUNCH"
```

2. FOR??? NEXT??? What do they mean? OK, in response to popular demand, we will explain what is happening in lines 120 and 130, and again in lines 150 and 160.

```
120 FOR Z = 1 TO 500
130 NEXT Z
```

← This is called a *FOR-NEXT* loop.

The above FOR-NEXT loop simply tells the computer to count from 1 to 500.

Start here stop here

```
120 FOR Z = 1 TO 500
130 NEXT Z
```

As the computer counts from 1 to 500, each counting number is put, momentarily, into box Z.

- First the computer puts 1 into box Z.
- Then the computer *erases* 1 and puts 2 into box Z.
- Then the computer erases 2 and puts 3 into box Z.
- Then the computer erases 3 and puts 4 into box Z.
- Then the computer erases 4 and puts 5 into box Z.
- ⋮
- And so on. Eventually, the computer puts 500 into box Z. (Of course, it first erased 499.)

This all happens very quickly, in *less than one second!*

Lines 150 and 160 do the same thing. Both FOR-NEXT loops are simply *time delays*. The computer is counting silently to itself while we see KARL on the screen or we see a completely blank screen. We see KARL for a little while, then we see a blank screen for a little while, then we see KARL for a little while, then we see a blank screen for a little while, then . . . and so it goes.

- (a) How can we change lines 120 and 150 so that Karl's name blinks more rapidly? _____
- (b) How can we change lines 120 and 150 so that Karl's name blinks less rapidly? _____

- (a) Use a number less than 500. For example, change line 120 to 120 FOR Z = 1 TO 200 and line 150 to 150 FOR Z = 1 TO 200. Now the TRS-80 will count from 1 to 200, which takes *less* time than counting from 1 to 500.
- (b) Use a number larger than 500. For example, change line 120 to 120 FOR Z = 1 TO 1000 and line 150 to 150 FOR Z = 1 TO 1000. Now the TRS-80 will count from 1 to 1,000, which takes *more* time than counting from 1 to 500.

3. Let's make it easier to change the name.

```
100 CLS
110 INPUT "WHAT IS YOUR NAME" ; N$
```

```
120 CLS
130 PRINT N$
140 FOR Z = 1 TO 500
150 NEXT Z
```

Name is ON
while TRS-80
counts to 500.



```
160 CLS
170 FOR Z = 1 TO 500
180 NEXT Z
```

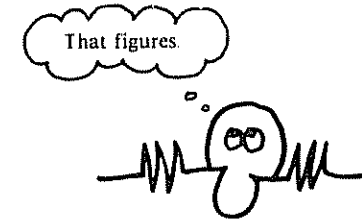
Name is OFF (screen is clear)
while TRS-80
counts to 500.

```
190 GOTO 120
```

- (a) Suppose we change line 140 to: 140 FOR Z = 1 TO 1000. Will the name be ON for a shorter time or a longer time? _____
- (b) Suppose we change line 170 to: 170 FOR Z = 1 TO 250. Will the name be OFF for a shorter time or a longer time? _____

- (a) Longer. It takes the computer about twice as long to count to 1,000 as it took to count to 500.

- (b) Shorter. It takes the computer about half as long to count to 250 as it took to count to 500.



4. Follow the arrows to see how the program works.

```

RUN
↓
100 CLS
↓
110 INPUT "WHAT IS YOUR NAME" ; N$
↓
120 CLS ←
↓
130 PRINT N$
↓
140 FOR Z = 1 TO 500
150 NEXT Z
↓
160 CLS
↓
170 FOR Z = 1 TO 500
180 NEXT Z
↓
190 GOTO 120
    
```

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

—A numeric variable must follow the word FOR:

```

140 FOR Z = 1 TO 500
    
```

—The same numeric variable follows the word NEXT:

```

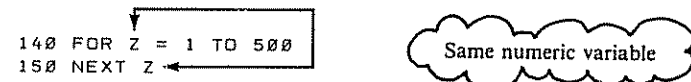
150 NEXT Z
    
```

Here is another way to say it:

```

140 FOR Z = 1 TO 500
150 NEXT Z

```



Let's not overwork the variable Z. *Any* numeric variable may be used.

—These are OK FOR-NEXT loops:

```

20 FOR K = 1 TO 3           73 FOR A = 1 TO 10
30 NEXT K                   89 NEXT A

```

—But these FOR-NEXT loops are *not* OK:

```

(a) 50 FOR X = 1 TO 100     (b) 110 FOR Z$ = 1 TO 200
60 NEXT Y                   120 NEXT Z$

```

What is wrong with (a)? _____

What is wrong with (b)? _____

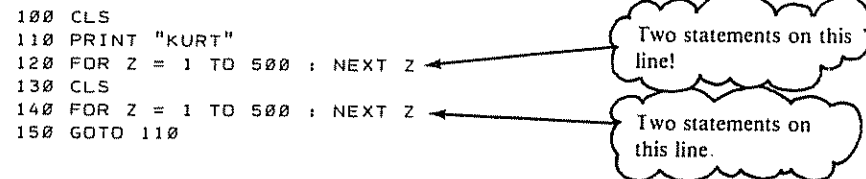
- (a) The variable following NEXT is different from the variable following FOR. These must be the *same* variable.
- (b) Oops! Z\$ is a *string* variable. We can use only numeric variables in FOR and NEXT statements.

6. And now, a handy space-saving, time-saving gimmick! The following program features *multiple statements per line*.

```

100 CLS
110 PRINT "KURT"
120 FOR Z = 1 TO 500 : NEXT Z
130 CLS
140 FOR Z = 1 TO 500 : NEXT Z
150 GOTO 110

```



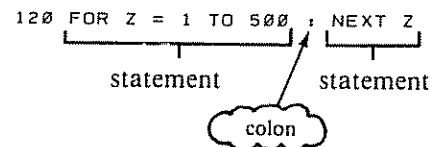
In the above program, line 120 contains two statements. Line 140 also contains two statements. In a line that contains two statements, what symbol, or character, is used *between* the statements? _____

a colon (:)

```

120 FOR Z = 1 TO 500 : NEXT Z

```



We usually put a space on each side of the colon. However, this is not necessary. We could have typed line 120, as follows:

```
120 FOR Z = 1 TO 500: NEXT Z
```

In fact, we can leave out *all* spaces and do it like this:

```
120FORZ=1TO500:NEXTZ
```

The TRS-80 will understand perfectly well, but it is hard for people to read! So, use spaces to make programs easier for humans to read.

7. Here is another example, using multiple statements per line. The following is a rewrite of the program in frame 3.

```
100 CLS : INPUT "WHAT IS YOUR NAME" ; N$
110 CLS : PRINT N$
120 FOR Z = 1 TO 500 : NEXT Z
130 CLS : FOR Z = 1 TO 500 : NEXT Z
140 GOTO 110
```

Follow the arrows.

```

RUN
↓
100 CLS : INPUT "WHAT IS YOUR NAME" ; N$
↓
110 CLS : PRINT N$
↓
120 FOR X = 1 TO 500 : NEXT Z
↓
130 CLS : FOR Z = 1 TO 500 : NEXT Z
↓
140 GOTO 110

```

The arrow from line 140 to line 110 shows you that multiple statements in a line are done from *left to right*. In line 110, the CLS statement is done first, then the PRINT statement. In line 130, the CLS statement is done first, then the FOR-NEXT loop. For the above program, complete the following:

| <i>Line Number</i> | <i>Number of Statements</i> |
|--------------------|-----------------------------|
| 100 | 2 |
| 110 | 2 |
| 120 | — |
| 130 | — |
| 140 | — |

| | | |
|-----|---|----------------------|
| 120 | 2 | (FOR and NEXT) |
| 130 | 3 | (CLS, FOR, and NEXT) |
| 140 | 1 | (GOTO) |

8. Your turn. Rewrite the following program, using multiple statements per line.

| | |
|-------------------------|--------------|
| <i>Ours</i> | <i>Yours</i> |
| 100 CLS | |
| 110 PRINT "DARTH VADER" | |
| 120 FOR Z = 1 TO 500 | |
| 130 NEXT Z | |
| 140 CLS | |
| 150 PRINT "LIVES!" | |
| 160 FOR Z = 1 TO 500 | |
| 170 NEXT Z | |
| 180 GOTO 100 | |

Here is one way to do it. Yours may be yet another way!

```
100 CLS : PRINT "DARTH VADER"
110 FOR Z = 1 TO 500 : NEXT Z
120 CLS : PRINT "LIVES!"
130 FOR Z = 1 TO 500 : NEXT Z
140 GOTO 100
```

When we use multiple statements per line, we try to put "related" statements on the same line. For example, line 100 tells the computer to clear the screen, then print DARTH VADER on the screen. Line 110 is a time delay, all on one line. Then, line 120 clears the screen and prints LIVES! Line 130 is another time delay, and line 140 causes the whole thing to repeat from the top.

We could have put the entire program on three lines, as follows:

```
100 CLS : PRINT "DARTH VADER" : FOR Z = 1 TO 500 : NEXT Z
110 CLS : PRINT "LIVES!" : FOR Z = 1 TO 500 : NEXT Z
120 GOTO 100
```

And, of course, there are many other ways. When you use multiple statements per line, try to make your program easy for *people* to understand!

9. Imagine a TRS-80 with a giant screen. It's the big game of the season and the TRS-80 is firing up the Rooter Club—building energy to help win the game. The program might look like this:

| | |
|--------------------------------|---------------------------------|
| 100 CLS : PRINT "GO" | Time delay |
| 110 FOR Z = 1 TO 500 : NEXT Z | |
| 120 CLS | Time delay |
| 130 FOR Z = 1 TO 500 : NEXT Z | |
| 140 PRINT "TEAM" | Time delay |
| 150 FOR Z = 1 TO 500 : NEXT Z | |
| 160 CLS | Time delay |
| 170 FOR Z = 1 TO 500 : NEXT Z | |
| 180 PRINT "GO!" | Time delay |
| 190 FOR Z = 1 TO 500 : NEXT Z | |
| 200 CLS | |
| 210 FOR Z = 1 TO 1000 : NEXT Z | Time delay (longer than others) |
| 220 GOTO 100 | |

The time delay in line 210 is longer than the others in order to separate (in time) each GO TEAM GO! from the next GO TEAM GO! Feel free to change these time delays to suit *your* sense of rhythm.



10. Six time delays in that last program. Lines 110, 130, 150, 170, and 190 are exactly the same and line 210 differs only slightly. Aha! A splendid opportunity for your cunning (but compassionate) authors to introduce . . . (fanfare!) . . . *subroutines*. We will write the time delay *once* as a subroutine, then use it as often as we need it.

In the following program, we have a *time delay subroutine* in lines 900 and 910. This subroutine is used in lines 110, 120, 140, 150, 170, and 180.

| <i>The Program</i> | <i>Comments about the Program</i> |
|---|--|
| 100 CLS : PRINT "GO" 110 GOSUB 900 | Look for line 900. It is our time delay subroutine (whatever that is!). |
| 120 CLS : GOSUB 900 | Aha! We use the time delay subroutine again. |
| 130 PRINT "TEAM" 140 GOSUB 900 | And yet again. |
| 150 CLS : GOSUB 900 | Ho hum—once more. |
| 160 PRINT "GO!" 170 GOSUB 900 | What? Again? |
| 180 CLS : GOSUB 900 : GOSUB 900 | Use it twice to get a longer delay. |
| 190 GOTO 100 | Go around again for another cheer. |
| 900 FOR Z = 1 TO 500 : NEXT Z 910 RETURN | Here it is! The long-awaited <i>time delay subroutine</i> . The RETURN statement tells the TRS-80 to return to the line following the GOSUB 900 that <i>called</i> the subroutine. |

The time delay subroutine is *called* (used) by the GOSUB 900 statement in lines 110, 120, 140, 150, 170, and twice in line 180. Obedient as ever, the TRS-80 goes to line 900, does it, then moves on to line 910. Aha! Line 910 says RETURN. So, the TRS-80 RETURNS to the statement following the GOSUB 900, and continues from there. Clever!

Subroutines are helpers. You *call* them when you need them. They do what they are designed to do, then return to where you called them.

11. Your turn. Complete the following program to blink Jjago's name on, off, on, off, on . . . and so on.

```
100 CLS : PRINT "JJAGD"           (Pronounced Yah-go)
110 _____ Use the time delay subroutine!
120 CLS : _____ Use it again.
130 GOTO 100

200 FOR Z = 1 TO 500 : NEXT Z     Time delay subroutine
210 RETURN
```

```
110 GOSUB 200
120 CLS : GOSUB 200
```

12. How about a *variable* time delay! Try it with Fonte's name (pronounced "Fon-tay").

```
100 CLS : PRINT "FONTE" : GOSUB 200
110 CLS : GOSUB 200
120 GOTO 100

200 T = 500           A variable
210 FOR Z = 1 TO T : NEXT Z     time delay
220 RETURN           subroutine
```

Hmmm . . . these programs
keep getting shorter.



The amount of delay (how far the computer counts) is controlled by the value of T in line 200.

Z goes from 1 to T

```

200 T = 500
210 FOR Z = 1 TO T : NEXT Z
220 RETURN

```

To speed up or slow down the blinking, change the value of T in line 200.

Faster: 200 T = 250 (Z will count from 1 to 250)
 Slower: 200 T = 1000 (Z will count from 1 to 1,000)

The time delay subroutine is called twice, once in line _____
 and once in line _____

 100; 110

13. In the program of the previous frame, the delay for FONTE *on* is the same as the delay for FONTE *off*. Here is a different way to do it, in which we set the value of T, then call the subroutine—which uses the value of T. The subroutine itself is different.

| | |
|-----------------------------|-------------------------------------|
| 100 CLS : PRINT "FONTE" | FONTE <i>on</i> . |
| 110 T = 1000 : GOSUB 200 | Set delay (T) and use subroutine. |
| 120 CLS | FONTE <i>off</i> (screen is clear). |
| 130 T = 500 : GOSUB 200 | Set delay (T) and use subroutine. |
| 140 GOTO 100 | Go around again. |
| 200 FOR Z = 1 TO T : NEXT Z | Time delay subroutine. |
| 210 RETURN | |

In the above program, FONTE will be *on* while the TRS-80 counts to 1,000 (T = 1000 in line 110), then the screen will be blank (clear) while the TRS-80 counts to 500 (T = 500 in line 130). So, FONTE will be *on* for about twice as long as *off*. Change the program so that things happen much more rapidly. Make the blinking much faster, with FONTE *on* about three times as long as FONTE *off*. Which two lines will you change?

(a) Lines _____ and _____ .
 How might you change them?

(b) Change line _____ to _____ .

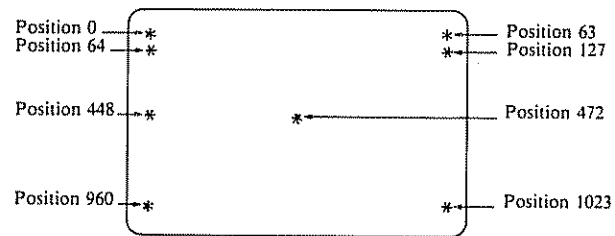
(c) Change line _____ to _____ .

- (a) 110; 130
- (b) 110; 110 T = 300 : GOSUB 200
- (c) 130; 130 T = 100 : GOSUB 200

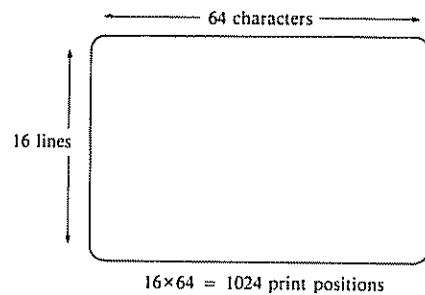
You may have used different values for T in lines 110 and 130. That's OK, as long as the value of T in line 110 is about three times as big as the value of T in line 130.

14. We seem to be doing everything in the upper left corner of the screen. Well, let's not wear out that corner! Instead, we will now make things happen elsewhere on the screen.

The TRS-80 can print in any one of 1024 print positions on the screen, from print position zero (0) in the upper left corner to print position 1023 in the bottom right corner. Each print position can hold one character of information.



Print positions are numbered from 0 (top left) to 1023 (bottom right). The screen is 64 characters wide and 16 lines from top to bottom.



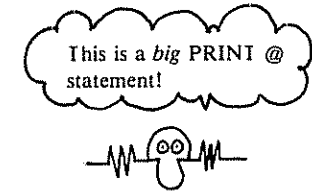
The print positions are numbered from _____ to _____.

0; 1023

See Appendix E for a map of the screen, showing all positions.

15. One of the nicest things about TRS-80 BASIC is how easy it is to tell the computer to print something *anywhere* on the screen.* Easy! Just use the PRINT @ statement.

> PRINT @



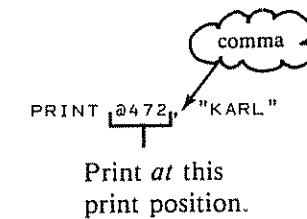
Here are some examples.

The statement: PRINT @63, "*"

tells the TRS-80 to print a star (*) at print position 63.

The statement: PRINT @472, "KARL"

tells the TRS-80 to print the word KARL at print position 472.



The statement: PRINT @128, "LUCY"

tells the TRS-80 to print the word _____ at print position _____

 LUCY; 128

16. Experiment! Here is a program to help you print whatever you want, wherever you want it.

```
100 CLS
110 INPUT "WHAT SHALL I PRINT" ; WS
120 INPUT "WHERE SHALL I PRINT IT" ; W

130 CLS
140 PRINT @W, WS

150 T = 2000
160 FOR Z = 1 TO T : NEXT Z

170 GOTO 100
```

*You will really appreciate this when you use a different computer that doesn't have this feature!

In this program, W\$ is a _____ variable and W is a _____ variable.

string; numeric

17. Enter the program from the previous frame and RUN it. Here are some possible RUNs.

Before pressing ENTER

```
WHAT SHALL I PRINT? KELLY  
WHERE SHALL I PRINT IT? 472_
```

After pressing ENTER

```
          KELLY  
          / \  
         /   \  
        /     \  
       /       \  
      /         \  
     /           \  
    /             \  
   /               \  
  /                 \  
 /                   \  
/                     \  
 \                     /  
  \                   /  
   \                 /  
    \               /  
     \             /  
      \           /  
       \         /  
        \       /  
         \     /  
          \   /  
           \ /  
            /  
           /  
          /  
         /  
        /  
       /  
      /  
     /  
    /  
   /  
  /  
 /  
/
```

This stayed on for about 5 seconds

```
WHAT SHALL I PRINT? KERRY  
WHERE SHALL I PRINT IT? 1022
```

```
          KE  
          / \  
         /   \  
        /     \  
       /       \  
      /         \  
     /           \  
    /             \  
   /               \  

```

Oops! When we tried to print KERRY in position 1022, only the first two letters would fit because print positions run from 0 to 1023 and each letter takes up one print position. After printing KE, the TRS-80 pushed everything on the screen up one line and printed the rest of KERRY on the bottom line, left side of the screen. (Sorry, Kerry.)

Hmmm, what do you think will happen if we tell the computer to print DANNY at position 1024? Let's try it.

```
WHAT SHALL I PRINT? DANNY  
WHERE SHALL I PRINT IT? 1024
```

```
?FC ERROR IN 140  
READY  
>_
```

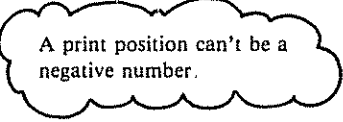

Well, there isn't a print position 1024, so the computer printed an error message. FC means "illegal function call," which doesn't tell us much. In this case, the error message means that we tried to print something at a nonexistent print position.

Print positions are numbered from _____ to _____

0; 1023

This is also illegal: `PRINT @-37, "A"`

But this is OK: `PRINT @23.67, "A"`



A print position can't be a negative number.

In this case, the computer will ignore the part of the number to the right of the point and print the letter A at print position 23.

18. 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).

Write your program in the space below.

```

100 CLS
110 INPUT "WHAT IS YOUR MESSAGE"; M$
120 INPUT "WHERE SHALL I BLINK IT" ; P

130 CLS
140 PRINT @P, M$
150 GOSUB 200

160 CLS
170 GOSUB 200

180 GOTO 130

200 T = 500
210 FOR Z = 1 TO T : NEXT Z
220 RETURN

```

You may have used different variables.

Message *on* at position P.
Use time delay subroutine

Message *off* (everywhere).
Use time delay subroutine.

Go around again.

Time delay subroutine.

Here is another way, using more multiple statements per line.

```

100 CLS
110 INPUT "WHAT IS YOUR MESSAGE" ; M$
120 INPUT "WHERE SHALL I BLINK IT" ; P

130 CLS : PRINT @P, M$ : GOSUB 200

140 CLS : GOSUB 200

150 GOTO 130

200 T = 500
210 FOR Z = 1 TO T : NEXT Z : RETURN

```

See Appendix E for a map of the screen showing the print positions from 0 to 1023. You may wish to experiment with your program relating actual positions on the screen of *your* TRS-80 with positions on the screen map in Appendix E.

19. OK, we confess—there is much 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 C = 1 TO 5 ]
30 PRINT "C = " C ]
40 NEXT C

```

This FOR-NEXT loop has *three* statements.



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

We could have written the above program like this:

```

10 CLS
20 FOR C = 1 TO 5 : PRINT "C = " C : NEXT C

```

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

(a) 30 PRINT "C = " C

(b) PRINT "C = " C



20. What do you think will happen if we enter the program of the preceding frame and then RUN it? Think about that for a moment. OK, we did it. Here is what happened after we typed RUN.

Just what you might expect. The TRS-80 counted to 5, using box C. It paused briefly to print each value of C so that we could C (oops, that should be "see") what is happening.

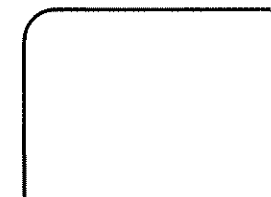
```
C = 1
C = 2
C = 3
C = 4
C = 5
READY
>_
```

Now, *you* be the computer. Show what will appear on the screen if we enter and RUN the following program.

The Program (ours)

```
10 CLS
20 FOR N = 1 TO 3
30 PRINT N
40 NEXT N
50 PRINT "GO!"
```

The Run (yours)



```

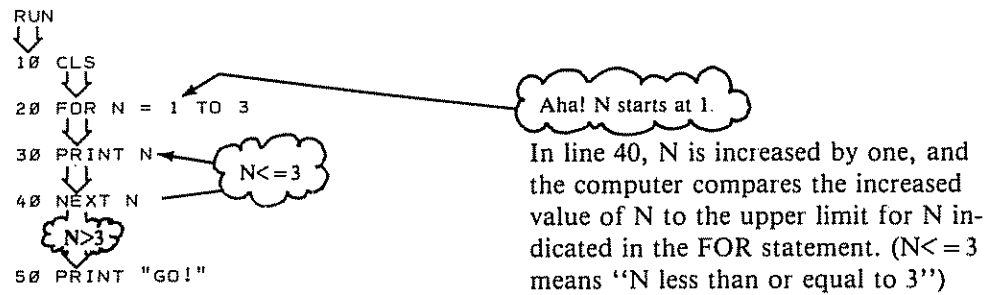
1
2
3
GO!
READY
>_

```

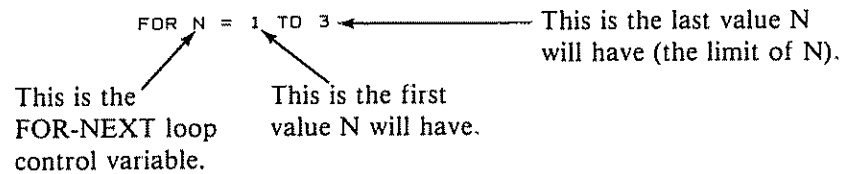
It is OK with us if you omitted these.

21. A FOR-NEXT loop can consist of *three* things.
- A FOR statement
 - A NEXT statement
 - Statements *between* the FOR statement and the NEXT statement

Of course, as you already know, the last item is optional. How does the FOR-NEXT loop work? Follow the arrows.



Let's look at the FOR statement.



When N goes past 3, the computer stops executing the loop and continues on with the rest of the program past the FOR-NEXT loop (if there is more to the program). We call this 3 the *limit* of N, or the limit of the FOR variable.

22. Each time the computer comes to a NEXT N statement, it increases the value of N by one, and checks the new value against the limit for N. In this case, the limit is 3, because the FOR statement reads: FOR N = 1 TO 3. When the value of N is greater than 3, the computer continues on to the statement following the NEXT statement, if there is one. If not, the computer has finished executing the program and stops. Got that? Let's see.

Statement 20 means that for the first time through the loop, N = 1. The second time through, N = N + 1 = 1 + 1 = 2. The third time through, N =

_____ = _____ = _____ .

$$N = N + 1 = 2 + 1 = 3$$

And now, a special treat for the older members of our audience. (Younger readers are encouraged to ask older readers about these programs.)

```
10 CLS
20 FOR E = 1 TO 3
30 PRINT E "O'CLOCK"
40 NEXT E
50 PRINT "ROCK!!!"
```

Or, if your nostalgia is even more seasoned . . .

```
10 CLS
20 FOR W = 1 TO 4
30 PRINT "AH" W
40 NEXT W
50 PRINT "(NOW IMAGINE THE SCREEN FULL OF BUBBLES)"
```

Later, we will show you how to fill the screen with "bubbles."

23. There is much more to learn about FOR-NEXT loops, and we will get to it eventually. But, for now, we will be using FOR-NEXT loops primarily to count from 1 to something.

So, experiment! You, with the help of your ever-patient, always-willing TRS-80, can teach yourself far more than we can. Try these programs to aid and abet your self-teaching, self-learning.

A variable FOR-NEXT loop:

```
10 CLS
20 INPUT "HOW FAR SHALL I COUNT" : F
30 CLS
40 FOR N = 1 TO F
50 PRINT N
60 NEXT N
```

(I bet you can't make your eyes go in different directions, as I can.)

Want to try some variations?

VARIATION: 50 PRINT N;

semicolon

VARIATION: 50 PRINT N,

comma

24. More experiments. Try the program. When the TRS-80 asks, respond. If something strange happens, ponder. And, remember our motto:

We can't give you all the answers.

So . . . *experiment!*

Try this one.

```

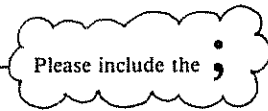
100 CLS
110 PRINT "YOU CAN COUNT ON ME...OR, PERHAPS I SHOULD"
120 PRINT "SAY, I CAN COUNT FOR YOU. WHEN I ASK, TELL"
130 PRINT "ME WHERE TO START AND WHERE TO STOP COUNTING."
140 PRINT
150 INPUT "WHERE SHALL I START" ; A
160 INPUT "WHERE SHALL I STOP" ; Z

170 CLS
180 FOR K = A TO Z
190 PRINT K ;
200 NEXT K

210 FOR T = 1 TO 100
220 NEXT T

230 GOTO 100

```



Time delay! Remember?

IMPORTANT NOTICE! Our programs are getting longer. Perhaps you have noticed that they take longer to type on the keyboard.

Well, you can type them in *once*, then save them on a tape cassette. If you save a program on a tape cassette, you can reenter it by using the cassette recorder. You will really appreciate this later when programs get even *l o n g e r !*

So, if you can possibly tear yourself away from the inspired prose of this chapter (you can always return—use a bookmark), we suggest that you now read Appendix B, “The Cassette Recorder.”

If you would rather wait, OK. Just remember that there is an appendix that shows you how to save *your* program on tape cassettes so you can read them back into the computer *without* retyping them.

25. Here is a mystery program for you to puzzle out. *Before* you enter and run this program, figure out what will happen on the screen.

```

100 FOR X = 0 TO 1023
110   CLS : PRINT @X, "->";
120   GOSUB 200
130 NEXT X

140 GOTO 100

200 T = 20
210 FOR Z = 1 TO T : NEXT Z
220 RETURN

```



with **SHIFT** key down

In the program above, a FOR-NEXT loop is in lines 110 through 130.

- (a) What is the variable in the FOR-NEXT loop? _____
- (b) What values will this variable have as the FOR-NEXT loop is executed?

- (c) Where is the variable used inside the loop? _____

(d) What will happen when we RUN the program? _____

- _____
- _____
-
- (a) X
 (b) 0, 1, 2, 3, and so on up to 1023
 (c) In the PRINT statement (line 110) to tell *where* to print the arrow (—>) on the screen.
 (d) Animation! The arrow (—>) will scamper across the screen from top left to bottom right. In its journey, it will pass through all 1024 (0 to 1023) printing positions. Eventually, it will complete its “grand tour” of the screen and . . . yup, start all over again.

26. Note how we wrote the FOR-NEXT loop in the previous frame.

```
100 FOR X = 0 TO 1023           Top of the loop
110   CLS : PRINT @X, "->";     Inside the loop
120   GOSUB 200
130 NEXT X                       Bottom of the loop
```

We indented the statements that are inside the loop two spaces. We did this simply to make it easier for *people* to read the program. The TRS-80 ignores the extra spaces, but most people find programs easier to read if we include the spaces. This will be even more important later on, as our programs get longer. We will begin using more devices to make programs more readable and understandable by humans. Some people call this “good programming style.”

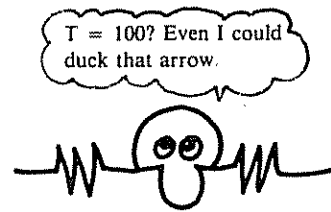
In line 120, the GOSUB 200 sends the computer to the time delay subroutine shown below.

```
200 T = 20
210 FOR Z = 1 TO T : NEXT Z
220 RETURN
```

The numeric variable T controls the amount of delay.

- (a) How can we make the arrow go faster? _____
- (b) How can we make the arrow go slower? _____
-

- (a) Make T smaller; for example: `200 T = 10`
 (b) Make T bigger; for example: `200 T = 100`



27. It's math time. If you are absolutely convinced that math is dreadful, you can skip ahead to the Self-Test. However, you might find the next few frames interesting and maybe even fun!

The FOR statement defines a *sequence* of values for its variable. Sequence? That's just a fancy math word meaning that the values come one after the other—sort of like breakfast, lunch, and dinner, the *sequence* of meals you eat, or the sequence of months of the year: January, February, March, and so on.

Here is a table showing a FOR statement, the variable it uses, and the sequence of values which the variable assumes, one after the other. Read the sequence of values from left to right.

| <i>FOR Statement</i> | <i>Variable</i> | <i>Sequence of Values</i> |
|-----------------------------|-----------------|---------------------------|
| <code>FOR N = 1 TO 3</code> | N | 1, 2, 3 |
| <code>FOR K = 1 TO 3</code> | K | 1, 2, 3 |
| <code>FOR A = 0 TO 6</code> | A | 0, 1, 2, 3, 4, 5, 6 |
| <code>FOR B = 3 TO 8</code> | B | 3, 4, 5, 6, 7, 8 |
| <code>FOR T = 1 TO 1</code> | T | 1 |

Try some yourself.

| <i>FOR Statement</i> | <i>Variable</i> | <i>Sequence of Values</i> |
|------------------------------|-----------------|---------------------------|
| <code>FOR C = 1 TO 3</code> | | |
| <code>FOR D = 0 TO 2</code> | | |
| <code>FOR R = 7 TO 8</code> | | |
| <code>FOR X = -1 TO 1</code> | | |

C 1, 2, 3
 D 0, 1, 2
 R 7, 8
 X -1, 0, 1 This one was tricky! Did you get it?

28. Hang on. Here we go! The following program will compute the *sum* of the *positive integers* from 1 to N. In case you have forgotten, the positive integers are the numbers we count with: 1, 2, 3, 4, 5, and so on. Here is our program.

```

10 CLS
20 PRINT : INPUT "N = " ; N
30 S = 0
40 FOR K = 1 TO N
50 S = S + K
60 NEXT K
70 PRINT "THE SUM IS" S
80 GOTO 20

```

The value of N must be a positive integer, or this program won't work the way we meant it to work.



The variable S (for sum) is used in lines 30, 50, and 70. Line 30 occurs *before* the FOR-NEXT loop. It sets the value of S to zero. Line 50 is *inside* the FOR-NEXT loop. It is executed for $K = 1$, $K = 2$, $K = 3$, and so on, until $K = N$. Each time, a new value of S is computed.

$50 \ S = S + K$
 New value ← ← Old value ← Most recent value of K, courtesy of FOR-NEXT loop

Suppose the INPUT value of N is 4. Then the sequence of values for K will be 1, 2, 3, 4. So, S will become:

```

In line 30:  S = 0
In line 50:  S = S + K = 0 + 1 = 1
In line 50:  S = S + K = 1 + 2 = 3
In line 50:  S = S + K = 3 + 3 = 6
In line 50:  S = S + K = 6 + 4 = 10

```

End of the FOR-NEXT loop. This value is printed by line 70.

Let's RUN the program. Here is a possible RUN.

```

N = ?4
THE SUM IS 10

N = ?7
THE SUM IS 28

```

N = ? and so on. Press **BREAK** to stop.

29. The product of the numbers from 1 to 3 is:

$$1 \times 2 \times 3 = 6$$

The product of the numbers from 1 to 5 is:

$$1 \times 2 \times 3 \times 4 \times 5 = 120$$

Your turn. Complete the following program to compute the *product* of the numbers from 1 to N.

```

10 CLS
20 PRINT : INPUT "N = " ; N

30 _____

40 FOR K = 1 TO N

50 _____

60 NEXT K

70 PRINT "THE PRODUCT IS" P
80 GOTO 20

```

Elsewhere in the program, use P to mean "product."

```

-----
P = 1
P = P*K   OR   P = K*P

```

30. The product of the numbers from 1 to N is called *N factorial*. It is the number of different ways to arrange N objects. For example, N factorial is the number of ways to arrange, or shuffle, N playing cards.

—The number of ways to arrange 3 playing cards is:

$$3 \text{ factorial} = 1 \times 2 \times 3 = 6 \text{ ways}$$

—The number of ways to arrange 5 playing cards is:

$$5 \text{ factorial} = 1 \times 2 \times 3 \times 4 \times 5 = 120 \text{ ways}$$

Let's RUN the program and have the TRS-80 compute N factorial for a few values of N.

```

N = 7
THE PRODUCT IS 5040

N = 710
THE PRODUCT IS 3.6288E+06

N = ? and so on.

```

Wow! 5040 ways to shuffle 7 cards.



Aha! A floating point number. See Appendix C.

Try some yourself. If you try a value of N larger than 33, you will get an error message as follows:

```

N = 34
70V ERROR IN 50

```

This is an Overflow error. The TRS-80 can work with numbers up to $1.7E+38$. Since 34 factorial is larger than $1.7E+38$, the computer prints an error message and stops. See Appendix C for a discussion of floating point numbers such as $1.7E+38$.

31. In the FOR-NEXT loops you have seen so far, the control variable increases by exactly one (1) each time. You can also write a FOR statement in which the control variable increases by a number other than 1.

```
FOR X = 1 TO 10 STEP 2
```

Tells the computer to increase the value of X by 2 each time, until X is greater than 10.

```
FOR Y = 0 TO 50 STEP 10
```

Tells the computer to increase the value of Y by 10 each time, until Y is greater than 50.

```
FOR D = 1 TO 2 STEP .25
```

Tells the computer to increase the value of D by .25 each time, until D is greater than 2.

| <i>FOR Statement</i> | <i>Variable</i> | <i>Sequence of Values</i> |
|-------------------------|-----------------|---------------------------|
| FOR X = 1 TO 10 STEP 2 | X | 1, 3, 7, 9 |
| FOR Y = 0 TO 50 STEP 10 | Y | 0, 10, 20, 30, 40, 50 |
| FOR D = 1 TO 2 STEP .25 | D | 1, 1.25, 1.5, 1.75, 2 |

Try some yourself.

| <i>FOR Statement</i> | <i>Variable</i> | <i>Sequence of Values</i> |
|--------------------------|-----------------|---------------------------|
| FOR E = 0 TO 10 STEP 2 | _____ | _____ |
| FOR R = 0 TO 10 STEP 3 | _____ | _____ |
| FOR Z = 10 TO 12 STEP .5 | _____ | _____ |

| | |
|---|------------------------|
| E | 0, 2, 4, 6, 8, 10 |
| R | 0, 3, 6, 9 |
| Z | 10, 10.5, 11, 11.5, 12 |

32. We can even make the TRS-80 count "backwards" by using a negative number following STEP.

FOR C = 10 TO 0 STEP -1

C will start at 10 and count down to 0.

Tells the TRS-80 to "increase" C by -1 each time. Or, if you prefer, tells the TRS-80 to decrease C by 1 each time.

What is the sequence of values of the variable C in the above FOR statement?

10,9,8,7,6,5,4,3,2,1,0

33. We call the following program "Countdown—Blastoff!" Enter it into your TRS-80 and RUN it.

```

100 REM***COUNTDOWN-BLASTOFF!
110 CLS

200 REM***COUNTDOWN FROM 10 TO 0
210 FOR C = 10 TO 0 STEP -1
220   PRINT C
230 FOR Z = 1 TO 300 : NEXT Z
240 NEXT C
250 PRINT "BLASTOFF!!!" : T = 400 : GOSUB 910

300 REM***SHOW SPACESHIP ON LAUNCH PAD
310 CLS
320 PRINT @512, "  *  "
330 PRINT   " *U* "
340 PRINT   " *S* "
350 PRINT   " *A* "
360 PRINT   " ***** "
370 PRINT   "*****"
380 T = 400 : GOSUB 910

400 REM***LAUNCH THE SPACESHIP
410 PRINT   " !!! " : T = 300 : GOSUB 910
420 PRINT   " !!! " : T = 200 : GOSUB 910
430 PRINT   " !!! " : T = 100 : GOSUB 910
440 FOR K = 1 TO 16
450   PRINT : T = 100 : GOSUB 910
460 NEXT K

500 REM***ANNOUNCE A SUCCESSFUL LAUNCH AND STOP
510 CLS
520 PRINT "ALL SYSTEMS ARE GO. EVERYTHING IS AOK!"
530 END

900 REM***TIME DELAY SUBROUTINE
910 FOR Z = 1 TO T : NEXT Z
920 RETURN

```

This stops the TRS-80

Now that you are launched into space, try the Self-Test to pass time until you reach your destination.

SELF-TEST

Well, here we are at another Self-Test, another chance for you to increase your self-esteem. Go for it!

1. Oops! We wanted to write a program to blink the word BLINK. But when we ran it, BLINK didn't blink. Please fix it by adding one statement, with an appropriate line number.

```
100 CLS
110 PRINT "BLINK"
120 FOR Z = 1 TO 500
130 NEXT Z
140 FOR Z = 1 TO 500
150 NEXT Z
160 GOTO 100
```

What statement did you add? _____

2. Would you believe? We did it again. Our second program to make BLINK blink also had a bug (error). Please find the bug and correct it.

```
100 CLS
110 PRINT "BLINK"
120 FOR Z = 1 TO 500
130 NEXT Z
140 CLS
150 FOR Z = 1 TO 500
160 NEXT Z
170 GOTO 120
```

Your fix? _____

3. Complete the following program to ask for a name and then ask how long to leave it on.

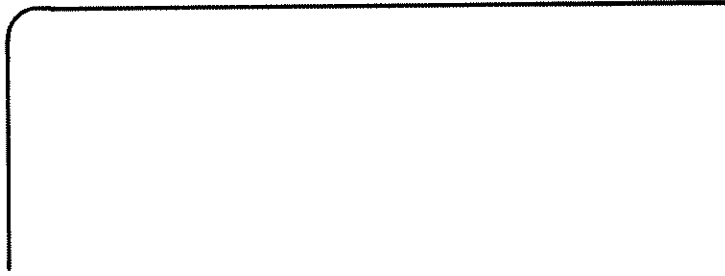
```
100 CLS
110 INPUT "WHAT IS YOUR NAME" ; N$
120 INPUT "HOW LONG SHALL I LEAVE IT ON" ; L

130 CLS: _____
140 FOR Z = 1 TO _____ : NEXT Z
150 GOTO 100
```

4. *You* are the computer. Show what a RUN will look like on your screen.

```
100 CLS
110 FOR B = 0 TO 10
120 PRINT 10-B
130 NEXT B
140 PRINT "BLASTOFF!!!"
```

Your screen:



5. Rewrite the following program so that it is all on *one* line.

```
100 CLS
110 PRINT "ONE LINER";
120 FOR X = 1 TO 100
130 NEXT X
140 GOTO 110
```

6. Write a time delay subroutine to count from 1 to 250 for the following program.

```
100 CLS
110 FOR X = 1 TO 5
120 PRINT "THE BLINKING COMPUTER"
130 GOSUB 1000
140 CLS
150 GOSUB 1000
160 NEXT X
```

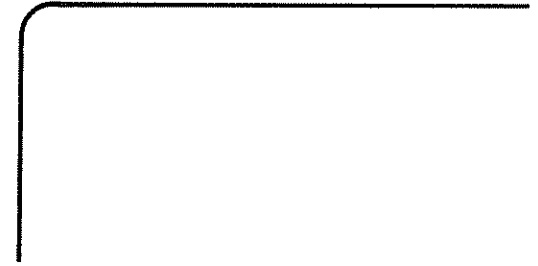
Subroutine:

7. Write a statement to print "EIGHTH LINE" at the beginning of the eighth line on the screen using PRINT @. If you wish, you may look at the screen map in Appendix E.

```
100 CLS
110 _____
120 GOTO 120
```

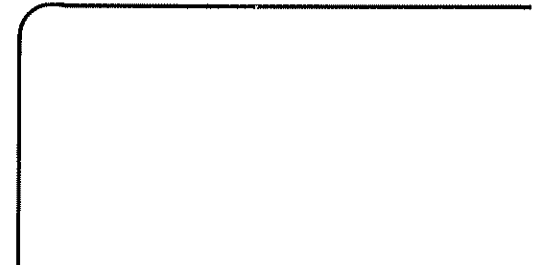
8. Show what will be on the screen when the following program is run.

```
100 CLS
110 FOR N = 1 TO 5
120   PRINT N;
130 NEXT N
```



9. Show what will be on the screen when the following program is run.

```
100 CLS
110 FOR N = 10 TO 2 STEP
120   PRINT N;
130 NEXT N
```



10. Write a program to compute the sum of the *odd* integers from 1 to N. A RUN of your program might look like this:

```
N = 75      (because 1 + 3 + 5 = 9)
THE SUM IS 9
```

```
N = 79      (because 1 + 3 + 5 + 7 + 9 = 25)
THE SUM IS 25
```

11. Write the sequence of values for the control variable in each FOR statement.

(a) FOR A = 1 TO 7

(b) FOR B = 1 TO 7 STEP 2

(c) FOR C = 0 TO 7 STEP 2

(d) FOR D = 0 TO 4 STEP 5

(e) FOR E = 1 TO -1 STEP -1

The next two are tricky! It is OK to guess.

(f) FOR F = 5 TO 3 _____

(g) FOR G = 1 TO 2 STEP 0 _____

Answers to Self-Test

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

1. 135 CLS (frames 1, 2)

2. Change 170 GOTO 100 (frames 1, 2)

3. 130 CLS : PRINT NS (frames 3, 4)
140 FOR Z = 1 TO L : NEXT Z

4. Your screen:

```
10
9
8
7
6
5
4
3
2
1
0
BLASTOFF!!!
```

(frames 24, 27)

5. 100 CLS : PRINT "ONE LINER"; : FOR X = 1 TO 100 : NEXT X : GOTO 100
(frames 6, 8)

6. 1000 FOR Y = 1 TO 250
1010 NEXT Y
1020 RETURN (frame 10)

You may have a different variable and maybe a multiple statement line.

7. 110 PRINT @448, "EIGHTH LINE"

Here is another way to do it. Line 1 begins with print position 0, line 2 with position 64, line 3 with position 128, and so on. So line 8 will begin with position $7 \times 64 = 448$. We could write the answer as follows.

```
110 PRINT @7*64, "EIGHTH LINE"
```

(frames 14-18)

8.

```

1 2 3 4 5
READY
>-
```

All on one line because of the semicolon at end of line 120.

(frames 23, 27)

9.

```

10 8 6 4 2
```

All on one line because of the semicolon at the end of line 120.

(frame 32)

10.

```

100 CLS
110 INPUT "N = "; N
120 S = 0
130 FOR K = 1 TO N STEP 2
140 S = S + K
150 NEXT K
160 PRINT "THE SUM IS"; S
170 PRINT
180 GOTO 110
```

Note: If you enter an *even* integer for N, the computer will compute the sum of the odd integers from 1 to N-1. For example:

N = ?10 because 1 + 3 + 5 + 7 + 9 = 25
 THE SUM IS 25 (frames 28, 31)

11. (a) 1, 2, 3, 4, 5, 6, 7
 (b) 1, 3, 5, 7
 (c) 0, 2, 4, 6
 (d) 0
 (e) 1, 0, -1
 (f) 5 only. There is no STEP to cause the computer to count backwards.
 (g) 1, 1, 1, . . . forever! Beware—don't use STEP 0.
 (frames 27, 31, 32)

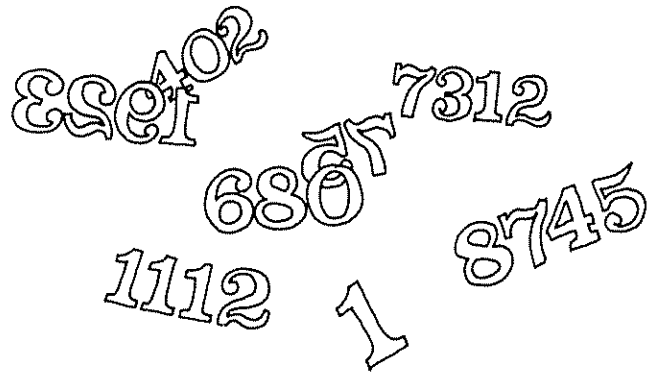
CHAPTER FIVE

Meandering

In this chapter, you will learn to create and use random numbers. These numbers will enable you to write your own games and simulations so that events happen in a random way.


After completing this chapter, you will be able to:

- generate whole random numbers over the range of 1 to 32,767;
- use random numbers in PRINT statements and time delays;
- use the IF statement to test a condition for its truth value;
- use IF-THEN to branch out of the computer's normal sequential order of line number operation;
- use inequalities: >, <, >=, <=, <>;
- use flags to find the end of a data list;
- create games: "Guess My Number," "Guess My Letter";
- annotate your programs with REMARK statements.



1. We will soon enter the fun-filled realm of computer games. Most computer games use *random numbers*.

Random numbers are numbers that are chosen at random from a given set of numbers. A die has six sides, numbered 1 through 6. We can roll one die to get a random number from 1 to 6.

This is one *die*: 

Two or more of them are called *dice*.

Suppose we want to get 1 or 2, at random. We flip a coin. If it comes up "heads," call it 1; if it comes up "tails," call it 2.

In BASIC, when we want a random number, we use the *RND function*. RND is an abbreviation for RaNDom 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 | Second RUN |
|--|--|
| <pre>2 2 1 1 2 2 2 2 BREAK IN 20 >_</pre> | <pre>1 1 1 2 2 1 2 BREAK IN 20 >_</pre> |

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!

One might even say
"unpredictable."



2. The program in the previous frame prints a list of random numbers. Because the program says PRINT RND(2), each random number is 1 or 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. Instead of a number, we can put a variable in parentheses following the word RND.

The function: RND(N)

will give a random number from 1 to N.

If N = 2, then RND(N) will be 1 or 2.

If N = 3, then RND(N) will be 1, 2, or 3.

If N = 6, then RND(N) will be 1, 2, 3, 4, 5, or 6.

Complete the following:

(a) If N = 4, then RND(N) will be _____ .

(b) If N = 100, then RND(N) will be a random number in the range _____ to _____ , inclusive.

(a) 1, 2, 3, or 4; (b) 1, 100

Any variable may be used in the RND function.

If A = 3, then RND(A) is 1, 2, or 3.

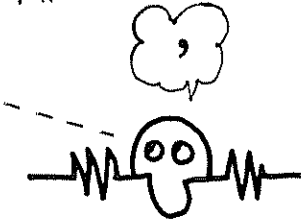
If Z = 7, then RND(Z) is 1, 2, 3, 4, 5, 6, or 7.

4. Experiment. Use the following program to print random numbers from 1 to N, where *you* supply the value of N.

```
100 CLS
110 PRINT "I WILL COMPUTE AND PRINT RANDOM NUMBERS"
120 PRINT "FROM 1 TO N. YOU ENTER THE VALUE OF N."

200 PRINT : INPUT "N = " ; N

300 CLS
310 PRINT RND(N),
320 GOTO 310
```



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

This is what
you see.

```
I WILL COMPUTE AND PRINT RANDOM NUMBERS
FROM 1 TO N. YOU ENTER THE VALUE OF N.

N = ?
```

Enter any value of N and press **ENTER**. The TRS-80 will print random numbers, four on each line, until you press **BREAK**.

Why four numbers per line? In line 310, the comma at the end of the PRINT statement causes the TRS-80 to move to the next *standard print position*. There are four such print positions across the screen, equally spaced.

Try some values of N, including the following:

- N = 2.9 The TRS-80 will ignore the .9 and print only the numbers, 1 or 2.
- N = -1 The TRS-80 will stop with the error message: ?FC ERROR IN 310. We can't use negative numbers in the RND function.
- N = 40,000 The TRS-80 will stop with the error message: ?OV ERROR IN 310. The largest value allowed in the RND function is 32,767. (Important! Remember this.)
- N = 0 What? Instead of whole numbers, we get decimals such as .566995 or .121945 or .465594. In this book, we will not use this type of random numbers.

5. OK, so RND gives random numbers. What do we *do* with them? Soon we will use random numbers in computer games. But first, let's use random numbers to put a name here, there, or anywhere on the screen. Here for a moment, there for a moment, somewhere else for a moment.

```

100 CLS
110 X = RND(1023)
120 PRINT @X, "KARL";
130 GOSUB 200
140 GOTO 100

```

```

200 T = 100
210 FOR Z = 1 TO T : NEXT Z } Time delay subroutine
220 RETURN

```

Look at lines 110 and 120. Line 110 computes a random number from 1 to 1,023 and puts this number in box X. So the value of X will be a random number from 1 to 1,023.

Aha! Line 120 uses the value of X to print KARL in a random place on the screen.

How can we slow things down so that KARL jumps about the screen a little less quickly? _____

Increase the value of T in line 200. Try T = 300 or T = 500 or T = 1,000.

Since X is a random number from 1 to 1,023, KARL will never be printed in screen position 0 (upper left corner). If we wish to include all 1,024 screen positions, from 0 to 1,023, we can change line 110, as follows:

```
110 X = RND(1024) - 1
```

See Appendix E for a screen map showing all 1,024 printing positions on the screen.

6. 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 changed lines 100, 110, 120, and 140, as follows:

```
110 CLS : INPUT "WHAT IS YOUR NAME" : N$
110 CLS : X = RND(1023)
120 PRINT @X, N$ :
140 GOTO 110
```

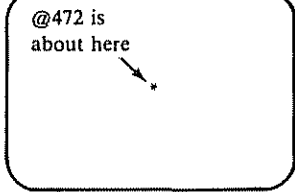
7. We call this program "One Star Twinkling." It causes one star (*) to blink on and off near the center of the screen. The blinking is uneven because we use a *random* time delay subroutine.

```
100 CLS : PRINT @472, "*"           Star ON
110 GOSUB 300

200 CLS : GOSUB 300                 Star OFF

300 T = RND(1000)                   Random
310 FOR Z = 1 TO T : NEXT Z         time delay
320 RETURN                           subroutine
```

@472 is
about here



Why is the time delay random? _____

Line 310 is a FOR-NEXT loop that causes the TRS-80 to count from 1 to T. The value of T is a random number from 1 to 1,000, computed by RND(1000) in line 300. So the time delay can vary from a tiny fraction of a second (T = 1) to about two seconds (T = 1,000).

To see even more variation in the blinking, change line 300 to:

```
300 T = RND(2000)
```

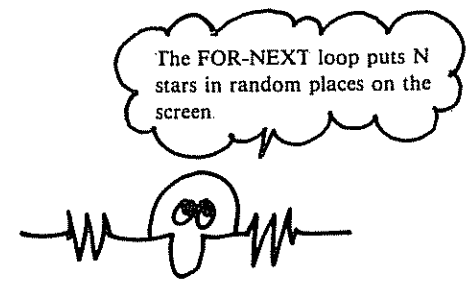
8. This program will put N stars on the screen in random positions. They wait a few seconds while you stargaze, then start over.

```
100 CLS
110 INPUT "HOW MANY STARS" : N

200 CLS
210 FOR K = 1 TO N
220   X = RND(1023)
230   PRINT @X, "*" :
240 NEXT K

300 T = 3000
310 FOR Z = 1 TO T : NEXT Z
320 GOTO 100
```

The FOR-NEXT loop puts N
stars in random places on the
screen.



The FOR-NEXT loop could also be written as follows:

```
210 FOR K = 1 TO N
220   PRINT @RND(1023), "*"
230 NEXT K
```

We increased the time delay by changing line 300 to: 300 T = 30000 (about 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 get fewer stars than the number we asked for. _____

Each star is printed in a random place, 1 to 1023, 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 ($K = 3$) was printed in position 235. Then, later on, the thirty-seventh star ($K = 37$) was also printed in position 235. Although two stars had been printed in position 235, we would see only one.

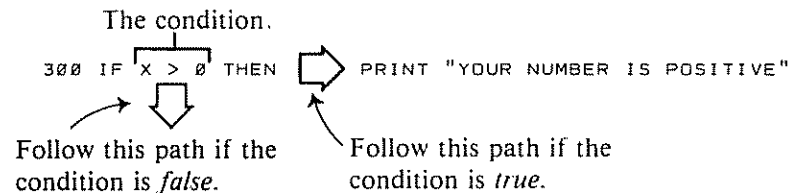
9. And now we present a very important and useful feature of BASIC: *the IF statement*.

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:

```
300 IF X > 0 THEN PRINT "YOUR NUMBER IS POSITIVE"
```

This IF statement tells the computer: If the value of X is greater than zero, then print the message, YOUR NUMBER IS POSITIVE, and go on to the next statement in line number order. The symbol > means "is greater than."

If the value of X is *less than zero* or *equal to zero*, the computer does *not* print the message. Instead, it continues executing the program in line number order.



What is the *condition* in the above IF statement? _____

$X > 0$ or X is greater than zero

10. Here again is the IF statement from frame 9:

```
300 IF X > 0 THEN PRINT "YOUR NUMBER IS POSITIVE"
```

This is the *condition*.

- (a) Suppose $X = 3$. Is the condition *true* or *false*? _____
- (b) Suppose $X = -7$. Is the condition *true* or *false*? _____
- (c) Suppose $X = 0$. Is the condition *true* or *false*? _____

-
- (a) True. The computer prints the message: YOUR NUMBER IS POSITIVE.
- (b) False. The computer does *not* print the message.
- (c) False. The computer does *not* print the message.

11. The symbol $>$ means "is greater than." The symbol $=$ means "is equal to." You can guess that the symbol $<$ means _____ .

"is less than"

12. Here is a simple program illustrating the use of the IF statement. This program has three IF statements that tell the computer to "compare and decide."

```
100 CLS
110 PRINT "ENTER A NUMBER AND I WILL TELL YOU WHETHER"
120 PRINT "YOUR NUMBER IS POSITIVE, NEGATIVE OR ZERO."

200 PRINT : INPUT "WHAT IS YOUR NUMBER" ; X

300 IF X > 0 THEN PRINT "YOUR NUMBER IS POSITIVE"
310 IF X < 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
320 IF X = 0 THEN PRINT "YOUR NUMBER IS ZERO"

400 GOTO 200
```

The following is a sample RUN.

```
ENTER A NUMBER AND I WILL TELL YOU WHETHER
YOUR NUMBER IS POSITIVE, NEGATIVE OR ZERO.

WHAT IS YOUR NUMBER? -7
YOUR NUMBER IS NEGATIVE ← This message printed by line 310.

WHAT IS YOUR NUMBER? 3
YOUR NUMBER IS POSITIVE ← This message courtesy of line 300.

WHAT IS YOUR NUMBER? 0
YOUR NUMBER IS ZERO ← Thank you, line 320.

WHAT IS YOUR NUMBER? ← And so on.
```

- (a) What is the condition in line 300? _____
- (b) What is the condition in 310? _____
- (c) What is the condition in line 320? _____

-
- (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

13. In running the program from the previous frame, the computer executes lines 100 through 120 once, since they are "outside the loop." Lines 200 through 400 are included in the loop and are executed for each value of X supplied by the user after an INPUT question mark. Suppose the user runs the program, types 13 after the question mark, and then presses ENTER. This assigns the value 13 to the variable X . Now look back at the program. Since $X = 13$ (13 is the value of X), the condition in line 300 is *true* and the conditions in lines 310 and 320 are *false*. So the computer will print the message in line 300, but will *not* print the messages in lines 310 and 320.

- (a) Suppose $X = -7$. The condition in line 310 is (*true* or *false*) _____ and the conditions in lines 300 and 320 are (*true* or *false*) _____.
- (b) Suppose $X = 0$. Which condition is *true*? (Give a line number.) _____ Which conditions are *false*? (Give line numbers.) _____

-
- (a) True; false. The computer will print the message in line 310, but will *not* print the messages in lines 300 and 320.
- (b) Line 320; lines 300 and 310. The computer will print the message in line 320, but will *not* print the messages in lines 300 and 310.

14. 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:

| <i>BASIC Symbol</i> | <i>Comparison</i> | <i>Math Symbol</i> |
|---------------------|-----------------------------|--------------------|
| = | 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 condition in proper BASIC:

- (a) G is equal to X _____
- (b) G is less than or equal to X _____
- (c) G is greater than X _____
- (d) G is greater than or equal to X _____
- (e) G is not equal to X _____

-
- (a) G = X
- (b) G <= X
- (c) G > X
- (d) G >= X
- (e) G <> X

15. Game time! Here is our first computer game, a simple number-guessing game.

```

100 CLS
110 PRINT "I WILL THINK OF A NUMBER. MY NUMBER WILL BE 1 OR 2"
120 PRINT "GUESS MY NUMBER!!!"

200 X = RND(2)

300 PRINT : INPUT "YOUR GUESS (1 OR 2)" ; G
310 IF G <> X THEN PRINT "I FOOLED YOU! THAT'S NOT MY NUMBER."
320 IF G = X THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"

400 T = 2000
410 FOR Z = 1 TO T : NEXT Z

500 GOTO 100

```

Your turn. Answer these questions:

- (a) In line 200, the TRS-80 "thinks" of a secret number, X. What are the possible values for 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 310? _____
- (e) Suppose this condition is *true*. What happens? _____

- (f) Suppose the condition is *false*. What happens? _____

- (g) What is the condition in line 320? _____

- (h) Suppose this condition is *true*. What happens? _____
- (i) Suppose the condition is *false*. What happens? _____

-
- (a) 1 or 2
- (b) G (lines 300, 310, 320)
- (c) 310 and 320
- (d) $G \neq X$ (G is not equal to X)
- (e) The TRS-80 prints: I FOOLED YOU! THAT'S NOT MY NUMBER. It then moves on to line 320.
- (f) Nothing. The TRS-80 moves on to line 320.
- (g) $G = X$ (G is equal to X)
- (h) The TRS-80 prints: THAT'S IT. YOU MUST HAVE ESP! It then moves on to line 400.
- (i) Nothing. The TRS-80 moves on to line 400.

16. One common use of the IF statement is to recognize a signal called a "flag" that terminates one process and begins another. In the following program, we use the number -1 as a flag to interrupt the computer and get the desired answer.

```
100 CLS
110 PRINT "I AM THE WORLD'S MOST EXPENSIVE ADDING MACHINE."
120 PRINT "WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. TO"
130 PRINT "QUIT, ENTER -1 AND I WILL PRINT THE TOTAL OF"
140 PRINT "ALL YOUR PREVIOUS NUMBERS." ; PRINT
```

```
200 T = 0
```

```
300 INPUT "X = " ; X
310 IF X = -1 THEN GOTO 400
320 T = T + X ; GOTO 300
```

```
400 PRINT ; PRINT "THE TOTAL IS" T
```

Lines 300 to 320 are a GOTO loop. But, if someone enters -1 as the value of X, line 310 jumps out of the loop to line 400.

A RUN might look like this:

```
I AM THE WORLD'S MOST EXPENSIVE ADDING MACHINE.
WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. TO
QUIT, ENTER -1 AND I WILL PRINT THE TOTAL OF
ALL YOUR PREVIOUS NUMBERS.
```

```
X = ? 2.50
X = ? 6.49
X = ? 27.95
X = ? -1
```

Aha! Here is our flag saying "that's all, folks."

```
THE TOTAL IS 36.94
```

In our program, the flag is -1. Line 310 checks each value of X and, if X = -1, jumps out of the loop to line 400. Line 400 prints the total (T) of all numbers previous to the flag.

Any unusual number that will not be used as a normal INPUT value could be used as a flag.

Modify the program so that, instead of using -1 as the flag, we use 999999 as the flag. You will have to change lines 130 and 310.

```
130 _____
310 _____
```

```
130 PRINT "QUIT, ENTER 999999 AND I WILL PRINT THE TOTAL OF"
310 IF X = 999999 THEN GOTO 400
```

17. Using -1 as a flag may not be a good idea if some of the values we wish to use are negative. For example, here are temperatures recorded during one cold week in Minneapolis, Minnesota:

| S | M | T | W | T | F | S |
|----|---|----|-----|-----|-----|-----|
| 10 | 3 | -9 | -15 | -23 | -25 | -30 |

In this case, using 999999 as the flag would prevent confusion between a temperature of -1 and an end-of-data flag of -1.

With a few changes, we can modify the program in the previous frame and obtain a program to compute the average of a set of numbers. The formula for determining the average of N numbers is as follows:

$$\text{Average} = \frac{\text{Sum or total of the numbers}}{\text{Number of numbers}} = \frac{T}{N}$$

In our program, we use the variable T for the total (sum) of the numbers, and N for the number of numbers. Complete the program.

```

100 CLS
110 PRINT "I WILL COMPUTE THE AVERAGE OF A BUNCH OF NUMBERS."
120 PRINT "WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. TO"
130 PRINT "QUIT, ENTER 999999 AND I WILL PRINT THE"
140 PRINT "AVERAGE OF YOUR PREVIOUS NUMBERS." : PRINT

200 T = 0 : N = _____ ← Initialize N

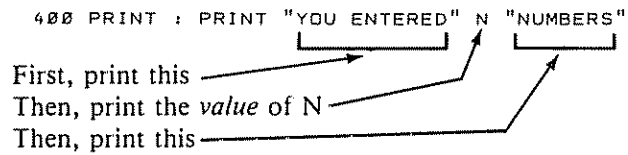
300 INPUT "X = " ; X
310 IF X = 999999 THEN GOTO 400
320 T = T + X : N = _____ : GOTO 300

400 PRINT : PRINT "YOU ENTERED" N "NUMBERS"
410 PRINT "THE TOTAL IS" T
420 PRINT "THE AVERAGE IS" T/N
    
```

N is the number counter.

0 Before any numbers are entered, N is zero.
 N + 1 After each new number (except 999999), increase N by 1.

Do you understand line 400?



Got it? If not, the RUN in the next frame may help.

18. Here is a RUN of the program in the previous frame, using those (brrr!) temperatures from Minneapolis:

```

I WILL COMPUTE THE AVERAGE OF A BUNCH OF NUMBERS.
WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. TO
QUIT, ENTER 999999 AND I WILL PRINT THE
AVERAGE OF YOUR PREVIOUS NUMBERS.

X = ? 10
X = ? 3
X = ? -9
X = ? -15
X = ? -23
X = ? -25
X = ? -30
X = ? 999999 ← the flag!
    
```

```

YOU ENTERED 7 NUMBERS
THE TOTAL IS -89
THE AVERAGE IS -12.7143
    
```

Yes, that was a cold week in Minneapolis!



Relax! No questions. But look at those temperatures. Can you imagine what the next week was like?!

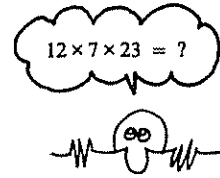
19. OK, your turn. Write a program to multiply the numbers entered by the user. Use 999999 as a flag to terminate the entering of numbers and get the answer. Here is a RUN of our program:

```
I AM THE WORLD'S MOST EXPENSIVE NUMBER CRUNCHER.
WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. I
WILL MULTIPLY ALL YOUR NUMBERS. TO QUIT, ENTER
999999 AND I WILL PRINT THE ANSWER.
```

```
X = ? 12
X = ? 7
X = ? 23
X = ? 999999
```

```
I MULTIPLIED YOUR NUMBERS. THE ANSWER IS 1932.
```

Write your program below:



```
100 CLS
110 PRINT "I AM THE WORLD'S MOST EXPENSIVE NUMBER CRUNCHER."
120 PRINT "WHEN I PRINT 'X = ?' YOU ENTER A NUMBER. I"
130 PRINT "WILL MULTIPLY ALL YOUR NUMBERS. TO QUIT, ENTER"
140 PRINT "999999 AND I WILL PRINT THE ANSWER." : PRINT
200 P = 1
300 INPUT "X = " ; X
310 IF X = 999999 THEN GOTO 400
320 P = P*X : GOTO 300
400 PRINT : PRINT "I MULTIPLIED YOUR NUMBERS. THE ANSWER IS" P
```

20. Game time again. This game is very much like our simple number-guessing game in frame 15. In this game, the computer "thinks" of a letter instead of a number.

```

100 CLS
110 PRINT "I WILL THINK OF A LETTER.. MY LETTER WILL BE A OR Z"
120 PRINT "GUESS MY LETTER!!!"

200 X = RND(2)
210 IF X = 1 THEN X$ = "A"
220 IF X = 2 THEN X$ = "Z"

300 PRINT : INPUT "YOUR GUESS (A OR Z)" ; G$
310 IF G$ <> X$ THEN PRINT "I FOOLED YOU! THAT'S NOT MY LETTER"
320 IF G$ = X$ THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"

400 T = 2000
410 FOR Z = 1 TO T : NEXT Z

500 GOTO 100

```

Your turn. Answer the questions.

In line 200, X will be 1 or 2. If X is 1, the computer's secret letter will be A (line 210).

- (a) But what if X is 2? What will the secret letter be? _____
- (b) What string variable is used to hold the secret letter? _____
- (c) What string variable is used to hold the player's guess? _____

Lines 310 and 320 compare the guess (G\$) with the secret number (X\$).

- (d) What is the condition in line 310? _____
- (e) What is the condition in line 320? _____
- (f) OK, got it? _____

-
- (a) z (line 220)
- (b) x\$ (line 210 or line 220)
- (c) g\$ (line 300)
- (d) g\$ <> x\$
- (e) g\$ = x\$
- (f) We hope you said *yes*. If not, then compare this program with the one in frame 15. The only real differences are the manner of choosing the secret letter (lines 200–220) and the use of string variables, X\$ instead of X, and G\$ instead of G. In fact, in lines 200 to 220, we use both a numeric variable (X) and a string variable (X\$).

21. This game is similar to the game in frame 15.

```

100 CLS
110 PRINT "I WILL THINK OF A NUMBER. MY NUMBER IS 1, 2 OR 3."
120 PRINT "GUESS MY NUMBER!!!"

200 X = RND(3)                                (X will be 1, 2, or 3)

300 PRINT : INPUT "YOUR GUESS (1,2,OR 3)" ; G
310 IF G <> X THEN PRINT "I FOOLED YOU! THAT'S NOT MY NUMBER."
320 IF G = X THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"

400 T = 2000
410 FOR Z = 1 TO T : NEXT Z

500 GOTO 100

```

No questions. Enjoy!

22. Instead of 1, 2, or 3, guess A, B, or C. *You* complete the program.

```

100 CLS
110 PRINT "I WILL THINK OF A LETTER. MY LETTER IS A, B, OR C."
120 PRINT "GUESS MY LETTER!!!"

200 X = RND( )
210 IF X = 1 THEN X$ = "A"
220 IF X = 2 THEN X$ = _____
230 _____

300 PRINT : INPUT "YOUR GUESS (A,B, OR C)" ; G$
310 IF G$ <> X$ THEN PRINT "I FOOLED YOU! THAT'S NOT MY LETTER"
320 IF G$ = X$ THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"

400 T = 200
410 FOR Z = 1 TO T : NEXT Z

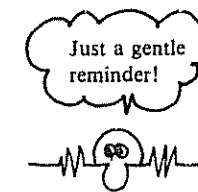
500 GOTO 100

-----

3
"B"
IF (X = 3) THEN X$ = "C"

```

Condition Statement



23. Our next game is an ancient "guess my number" game. Note the multiple statements per line in lines 310 and 320. Enjoy the game first, then read the next few frames to find out how the game works.

```

100 CLS
110 PRINT "I WILL THINK OF A NUMBER FROM 1 TO 100."
120 PRINT "GUESS MY NUMBER!!!"

200 X = RND(100)

300 PRINT : INPUT "YOUR GUESS" ; G
310 IF G < X THEN PRINT "TRY A BIGGER NUMBER." : GOTO 300
320 IF G > X THEN PRINT "TRY A SMALLER NUMBER." : GOTO 300

400 IF G = X THEN PRINT "THAT'S IT! YOU GUESSED MY NUMBER."
410 T = 2000
420 FOR Z = 1 TO T : NEXT Z

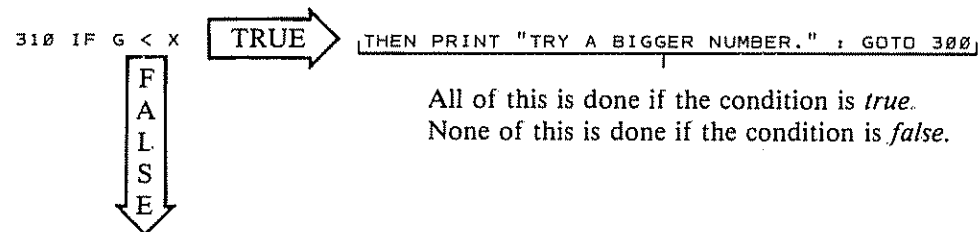
500 GOTO 100

```

Line 200 tells the computer to generate a random number as the value of X. This number will be a(n) _____ from _____ to _____, inclusive.

integer (or whole number); 1; 100

24. In lines 310 and 320, we use another feature of the IF statement. In line 310, if the condition is *true*, *everything* following the word THEN will be done. But if the condition is *false*, *neither* the PRINT statement *nor* the GOTO statement will be done.



The *condition* is: $G < X$

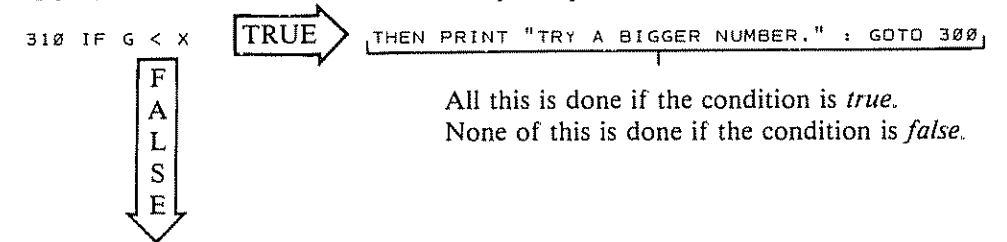
- (a) Suppose $G = 70$ and $X = 73$. Is the condition *true* or *false*? _____
- (b) Suppose $G = 90$ and $X = 73$. Is the condition *true* or *false*? _____
- (c) Suppose $G = 73$ and $X = 73$. Is the condition *true* or *false*? _____

- (a) *True*. The computer will print TRY A BIGGER NUMBER and go to line 300.
- (b) *False*. The computer will ignore everything to the right of THEN and move on to line 320.
- (c) *False*. The computer will ignore everything to the right of THEN and move on to line 320.

25. Line 300 causes the computer to print YOUR GUESS? and wait for a guess. When the player enters a guess, the computer stores it in box G. Lines 310, 320, and 400 compare the guess (G) with the secret number (X). Let's look again at line 310:

```
310 IF G < X THEN PRINT "TRY A BIGGER NUMBER." : GOTO 300
```

If the guess G is less than the number X, the computer will print the message TRY A BIGGER NUMBER and will then GOTO line 300 to ask for another guess. However, if G is greater than X or equal to X, *neither* the PRINT *nor* the GOTO will be done. Here is another way to "picture" that idea:



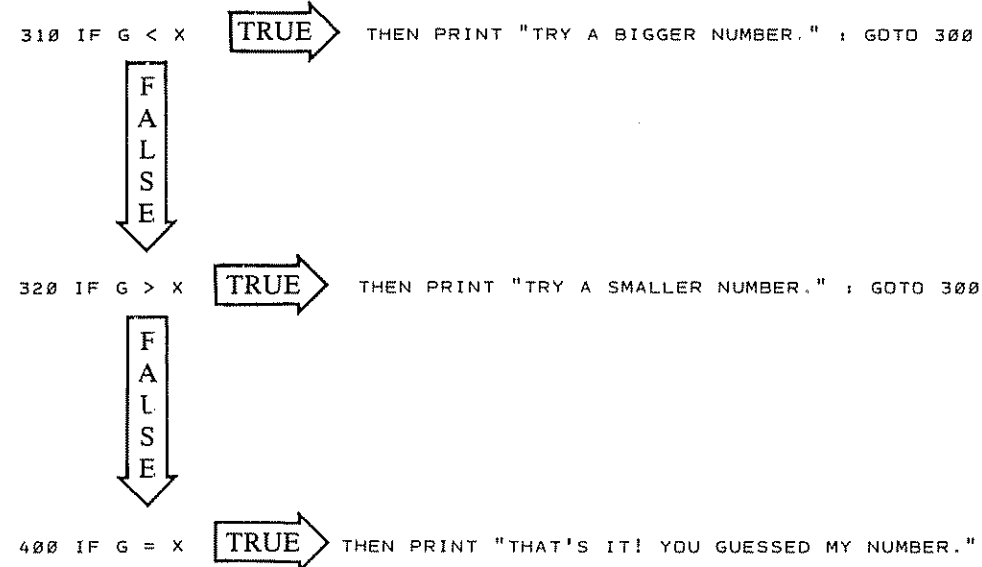
So, if $G > X$ is *false*, the computer goes on to line 320. Describe what happens when the computer executes line 320.

```
320 IF G > X THEN PRINT "TRY A SMALLER NUMBER." : GOTO 300
```

- (a) If G is greater than X ($G > X$ is *true*), then _____
- (b) However, if G is *not* greater than X ($G > X$ is *false*), then _____

- (a) the computer will print the message TRY A SMALLER NUMBER and then GOTO line 300 for another guess.
- (b) neither the PRINT nor the GOTO will be done. The computer will move on to line 400, which follows line 320 in line number order.

26. Eventually, the player will guess the number. Then, $G > X$ is *false* and $G < X$ is *false*. The computer will finally arrive at line 400. Here is a picture of the entire process of arriving at line 400:



Suppose the player has guessed the number and entered the correct guess. Therefore, $G > X$ is *false*, $G < X$ is *false*, and, of course, $G = X$ is *true*. What happens?

The computer prints THAT'S IT! YOU GUESSED MY NUMBER and, since there is no GOTO in line 400, moves on to line 410.

Hmmmm . . . If $G < X$ is *false* and $G > X$ is *false*, then $G = X$ *must* be *true*. So we could have written line 400 as follows:

```
400 PRINT "THAT'S IT! YOU GUESSED MY NUMBER."
```

There are three paths in lines 300 through 400. We suggest that you return to frame 23 and mark these paths in three colors. For example,

Red path: 300 to 310 to 300, etc.
 Blue path: 300 to 310 to 320 to 300, etc.
 Green path: 300 to 310 to 400 to 410, etc.

27. We may wish to change the range of numbers in this game. For example, for very young children, we might want to make the range 1 to 10 or 1 to 20. For advanced players, 1 to 1,000.

- (a) Modify lines 110 and 200 so that the range is 1 to 20.

110 _____

200 _____

- (b) Modify lines 110 and 200 so that the range is 1 to 1,000.

110 _____

200 _____

- (a) 110 PRINT "I WILL THINK OF A NUMBER FROM 1 TO 20."
-
- 200 X = RND(20)

- (b) 110 PRINT "I WILL THINK OF A NUMBER FROM 1 TO 1000."
-
- 200 X = RND(1000)

28. This way makes it easier to change the range.

```
100 CLS
110 R = 100
120 PRINT "I WILL THINK OF A NUMBER FROM 1 TO" R
130 PRINT "GUESS MY NUMBER!!!"
```

200 X = RND(R)

Now the range is 1 to R, where R is assigned a value in line 110. This value of R is used in lines 120 and 200. So, to change the range, change only line 110. Suppose you want the range to be 1 to 500. What do you write for line 110?

110 _____

110 R = 500

29. Why not let the player choose the range? Here is a RUN of our program.
- You*
- write a program that will produce this type of RUN.

```
I WILL THINK OF A NUMBER FROM 1 TO R.
FIRST, YOU MUST TELL ME THE VALUE OF R.
```

WHAT IS R? 1000

OK, I HAVE A NUMBER FROM 1 TO 1000 ← Value of R.

```
YOUR GUESS? 500
TRY A SMALLER NUMBER
```

YOUR GUESS? and so on.

Write your program in the meager space provided by our publisher.



```
100 CLS
110 PRINT "I WILL THINK OF A NUMBER FROM 1 TO R."
120 PRINT "FIRST, YOU MUST TELL ME THE VALUE OF R."
130 PRINT : INPUT "WHAT IS R" ; R

200 X = RND(R)
210 PRINT : PRINT "OK, I HAVE A NUMBER FROM 1 TO" R

300 PRINT : INPUT "YOUR GUESS" ; G
310 IF G < X THEN PRINT "TRY A BIGGER NUMBER" ; GOTO 300
320 IF G > X THEN PRINT "TRY A SMALLER NUMBER" ; GOTO 300

400 PRINT "THAT'S IT! YOU GUESSED MY NUMBER."
410 T = 2000
420 FOR Z = 1 TO T : NEXT Z

500 GOTO 100
```

30. Trouble! Here is what *might* happen when we RUN our program of the previous frame:

```
I WILL THINK OF A NUMBER FROM 1 TO R.
FIRST, YOU MUST TELL ME THE VALUE OF R.

WHAT IS R? 40000
?OV ERROR IN 200
```

Oops! The RND function works with whole numbers up to 32,767, or with 0.

RND(R)

↑
Must be less than or equal to 32,767,
or can be zero (0).

But zero will also be trouble in this game. If $R = 0$, then $RND(0)$ will be a decimal number between 0 and 1—terribly difficult to guess!*

Avoid trouble. We suggest that you add the following two statements to our program. These statements will “trap” undesirable values of R .

```
140 IF R > 32767 THEN PRINT "R MUST BE LESS THAN 32768" : GOTO 130
150 IF R = 0 THEN PRINT "R MUST BE 1 OR MORE" : GOTO 130
```

Hmmmm . . . almost forgot. Negative values of R also give trouble.

```
160 IF R < 0 THEN PRINT "R MUST BE A POSITIVE NUMBER" : GOTO 130
```

Or we could combine lines 150 and 160 into a single line.

```
150 IF R < 1 THEN PRINT "R MUST BE 1 OR MORE" : GOTO 130
```

31. 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 our number-guessing game from frame 23. We have rewritten it, using REMARK statements to describe—to people—what is happening in the program.

```
100 REMARK THIS IS A NUMBER GUESSING GAME
110 REMARK EXPLAIN THE GAME TO THE PLAYER
120 PRINT "I WILL THINK OF A NUMBER FROM 1 TO 100."
130 PRINT "GUESS MY NUMBER!!!"

200 REMARK COMPUTER 'THINKS' OF A SECRET NUMBER, X
210 X = RND(100)

300 REMARK GET GUESS (G), COMPARE WITH X, GIVE HINT
310 PRINT : INPUT "YOUR GUESS" : G
320 IF G < X THEN PRINT "TRY A BIGGER NUMBER" : GOTO 310
330 IF G > X THEN PRINT "TRY A SMALLER NUMBER" : GOTO 310

400 REMARK PLAYER HAS GUESSED THE NUMBER
410 PRINT "THAT'S IT! YOU GUESSED MY NUMBER."
420 T = 2000
430 FOR Z = 1 TO T : NEXT Z      : REMARK TIME DELAY
440 GOTO 100                    : REMARK GOTO BEGINNING
```

Since REMARK is a statement, it can occupy a line all by itself or be on the same line as other statements. Which lines contain REMARK statements along with other statements? Lines _____ and _____ .

*One of the nicest things about the TRS-80 is the way the RND function works. It gives *integer* random numbers in a way that is easy to understand and use by people who are not math wizards. Most other BASICs give random numbers between 0 and 1. We can't understand why—most applications (especially games!) require *integer* random numbers. In most BASICs, much mind-boggling math is required to get the desired *integer* random numbers. In the TRS-80, you can use $RND(0)$ to do random numbers the way other computers do them. For more information, consult your *Level II BASIC Reference Manual*, available from most Radio Shack stores.

430 ; 440

32. Your turn. Here is a rewrite of our game in frame 20. Complete the REMARK statements describing the program.

```

100 REMARK _____
110 REMARK _____

120 CLS
130 PRINT "I WILL THINK OF A LETTER. MY LETTER WILL BE A OR Z."
140 PRINT "GUESS MY LETTER!!!"

200 REMARK _____
210 X = RND(Z)
220 IF X = 1 THEN X$ = "A"
230 IF X = 2 THEN X$ = "B"

300 REMARK _____
310 PRINT : INPUT "YOUR GUESS (A OR Z)" ; G$
320 IF G$ <> X$ THEN PRINT "I FOOLED YOU! THAT'S NOT MY LETTER."
330 IF G$ = X$ THEN PRINT "THAT'S IT. YOU MUST HAVE ESP!"

400 REMARK _____
410 T = 2000
420 FOR Z = 1 TO T : NEXT Z

500 GOTO 120           : REMARK _____

```

```

-----
100 REMARK THIS IS A LETTER GUESSING GAME, GUESS A OR Z
110 REMARK EXPLAIN THE GAME TO THE PLAYER
200 REMARK COMPUTER 'THINKS' OF SECRET LETTER (X$), A OR Z
300 REMARK GET GUESS (G$), COMPARE WITH X$, TELL WHAT HAPPENED
400 REMARK TIME DELAY, DEPENDS ON VALUE OF T
500 ... REMARK GOTO BEGINNING

```

Instead of REMARK, we can write REM. For example, line 100 can be written as follows:

```
100 REM THIS IS A LETTER GUESSING GAME, GUESS A OR Z
```

To call attention to REM statements, we may also write them like this:

```
100 REM***THIS IS A LETTER GUESSING GAME, GUESS A OR Z
```

You have now learned how to use random numbers in games and recreations. Try the following Self-Test, if you like, to see how well you have learned to use the mysterious and unpredictable RND functions.

SELF-TEST

1. What possible values would be printed when these statements are executed?

(a) PRINT RND(5) _____

(b) PRINT RND(9) _____

(c) PRINT RND(8)-1 _____

2. Complete the following:

(a) If $N = 3$, then RND(N) will be _____

(b) If $X = 150$, then RND(X) will be a random number in the range of _____ to _____.

(c) If $D = 5.3$, then RND(D) will be _____

3. Where would the word HERE be printed if this program were executed?

```
100 CLS
110 WS = "HERE"
120 N = RND(60)-1
130 PRINT @N, WS
```

4. Tell how many stars would be printed and where they would be printed by this program. _____

```
100 CLS
110 FOR P = 960 TO 973
120 PRINT @P, "*" ;
130 NEXT P
140 GOTO 140
```

5. This line is added to the program in exercise 4:

```
115 IF P = 973 THEN GOTO 130
```

If rerun with this modification, how many stars will be printed? _____

6. Tell what these BASIC conditions mean.

(a) $A = X$ _____

(b) $A <> X$ _____

(c) $A > X$ _____

(d) $A < = X$ _____

(e) $A < X$ _____

(f) $A > = X$ _____

7. Study this "Seeing Stars" program and answer the questions that follow it.

```

100 CLS
110 PRINT "I AM SEEING STARS. I SEE LESS THAN 4 STARS."
120 INPUT "HOW MANY STARS DO YOU THINK I SEE" ; A
130 N = RND(3)
140 CLS
150 IF A = N THEN PRINT "YOU ARE RIGHT!"
160 PRINT @192, "YOU SAID" ; A
170 FOR W = 1 TO A : PRINT "*" ; : NEXT W
180 PRINT @384, "I SAW" ; N
190 FOR W = 1 TO N : PRINT "*" ; : NEXT W
200 FOR D = 1 TO 500 : NEXT D
210 GOTO 100

```

- (a) What are the possible values for N at line 130? _____
- (b) How many stars will be printed by line 170? _____
- (c) The number of stars printed by line 190 depends upon the value assigned to what variable? _____
- (d) Which other line(s) would have to be changed if line 110 is changed to:

```
110 PRINT "I AM SEEING STARS. I SEE LESS THAN 6 STARS."
```

- (e) Will all the stars resulting from line 170 be on the same line?
- _____

8. Here is another program to calculate N factorial. Line 120 contains a flag that can be used to terminate the program.

```

100 CLS : T = 1
110 INPUT "X = " ; X
120 IF X < 0 THEN GOTO 190
130 FOR N = 1 TO X
140   T = T*N
150 NEXT N
160 PRINT X ; "FACTORIAL = " ; T
170 FOR X = 1 TO 1000 : NEXT X
180 GOTO 100
190 END

```

- (a) What number(s) can you input to terminate the program? _____
- _____

- (b) What's the purpose of line 170? _____
- _____

9. Here is a program called "A Reward If You Guess My Letter":

```

100 CLS
110 PRINT "I PROMISE A REWARD IF YOU GUESS MY LETTER."
120 X = RND(3)
130 IF X = 1 THEN X$ = "A"
140 IF X = 2 THEN X$ = "B"
150 IF X = 3 THEN X$ = "C"
160 INPUT "WHAT IS MY LETTER (A, B, C)" ; G$
170 IF G$ = X$ THEN GOTO 200
180 INPUT "SORRY, NO REWARD. TRY AGAIN!" ; G$
190 GOTO 170
200 CLS : PRINT "REWARD TIME"
210 FOR C = 11 TO 1023
220   PRINT @C, "*"
230   FOR W = 1 TO 10 : NEXT W
240 NEXT C
250 GOTO 100

```

- (a) The probability for getting a reward is very high. You should be able to receive a reward in at least how many tries? _____
- (b) If your first guess is wrong, what happens? _____
- (c) Suppose you want the player to have only one guess. Change line 180 so that it will have a time delay, and change 190 so that the program will go back to the beginning.

180 _____ (multiple statement)

190 GOTO _____

10. "Guess My Number Again." This time a scorecard is added.

```

100 CLS : N = 0
110 PRINT "I WILL THINK OF A NUMBER FROM 1 TO 100."
120 PRINT "THEN YOU GUESS MY NUMBER."
200 X = RND(100)
300 PRINT : INPUT "YOUR GUESS" ; G
310 N = N + 1
320 IF G < X THEN PRINT "TRY A BIGGER NUMBER" ; GOTO 300
330 IF G > X THEN PRINT "TRY A SMALLER NUMBER" ; GOTO 300
400 CLS
410 PRINT "YOU GUESSED IT IN" ; N ; "TRIES"
420 PRINT "YOU GET ONE STAR FOR EACH TRY"
430 FOR Y = 1 TO N : PRINT "*" ; NEXT Y
440 PRINT "TRY AGAIN AND SEE HOW CLOSE YOU CAN COME TO ONE STAR."
450 FOR Z = 1 TO 100 : NEXT Z
460 GOTO 100

```

- (a) Why was N set to zero in line 100? _____
- (b) Why was line 310 added? _____
- (c) How many stars will be printed by line 430? _____
- (d) What is the smallest number of stars when this program is RUN?

11. Test your ESP again. We will give you exactly ten tries at guessing a number 1 or 2. Your ESP score will be given at the end.

```

100 CLS : C = 0
105 FOR N = 1 TO 10
110   PRINT "I AM THINKING OF A NUMBER."
120   PRINT "IT IS EITHER 1 OR 2."
200   X = RND(2)
300   PRINT: INPUT "YOUR GUESS (1 OR 2)" ; G
310   IF G <> X THEN PRINT "I FOOLED YOU! THAT'S NOT MY NUMBER."
320   IF G = X THEN C = C + 1 : PRINT "THAT'S CORRECT. SCORE ONE FOR YOU."
330   FOR Z = 1 TO 100 : NEXT Z
340   CLS
350   NEXT N
360   PRINT "YOU GUESSED" ; C ; "CORRECT OUT OF 10."
370   PRINT "THAT'S" ; C*10 ; "PERCENT."
400   FOR Z = 1 TO 1000 : NEXT Z
410   GOTO 100

```

- (a) Which program lines limit the number of tries? _____
- (b) What trouble would arise from changing line 410 to:

```
410 GOTO 105
```

- (c) Suppose we want to change the program to twenty tries instead of ten. Tell which lines must be changed and show how you would change them.

Answers to Self-Test

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

1. (a) 1, 2, 3, 4, or 5 (frames 1, 2)
- (b) 1, 2, 3, 4, 5, 6, 7, 8, or 9 (frames 1, 2)
- (c) 0, 1, 2, 3, 4, 5, 6, or 7 (frames 1, 2, 5)
2. (a) 1, 2, 3 (frames 3, 4)
- (b) 1 to 150 (frames 3, 4)
- (c) 1, 2, 3, 4, or 5 (the .3 is ignored) (frame 4)
3. Somewhere on the first line of the display starting anywhere from position 0 to position 59. (frames 5, 6)
4. 14 stars would be printed at the bottom of the screen (positions 960-973). (frame 8)

-
5. 13 (the PRINT statement will not be executed when P = 973). (frames 9, 10, 12)
6. (a) A = x A equals X
(b) A <> x A does not equal X
(c) A > x A is greater than X
(d) A <= x A is less than or equal to X
(e) A < x A is less than X
(f) A >= x A is greater than or equal to X (frames 11, 12, 14)
7. (a) 1, 2, or 3
(b) 1, 2, or 3 (it depends on your input at line 120)
(c) the variable N
(d) only line 130 to 130 N = RND(5)
(e) yes
8. (a) Any negative number (-1, -2, -3, etc.)
(b) Time delay so that you can read the result (feel free to change the upper limit)
9. (a) 3 at the most
(b) Sorry message is printed and another input is requested; it is then checked.
(c)

```
180 FOR W = 1 TO 200 : NEXT W
190 GOTO 100
```
10. (a) To initialize the number of guesses—to start at zero.
(b) To count the number of guesses
(c) Whatever number of guesses is needed to get the correct answer
(d) One (correct guess the first time)
11. (a) Line 105 and line 350 (FOR-NEXT)
(b) The count would not be set back to zero and the score would accumulate.
(c)

```
105 FOR N = 1 TO 20
350 PRINT "YOU GUESSED" ; C ; "CORRECT OUT OF 20."
360 PRINT "THAT'S" ; C*5 ; "PERCENT."
```
-

CHAPTER SIX

Patterns and Games

In this chapter, you will begin to learn how to put patterns on the screen, use graphics statements to put collections of “stars” on the screen, and draw simple “pictures.” You’ll also learn about the INKEY\$ function and how to use it to stop and restart the action on the screen. The chapter ends with a game that you can use to test your reaction time.

After completing this chapter, you will be able to:

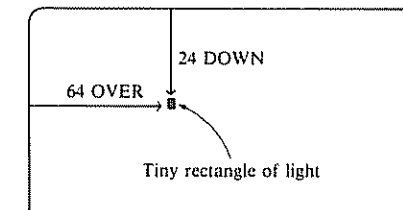
- turn on tiny rectangles of light on the screen using the SET statement;
- control the placement of rectangles on the screen—the key to TRS-80 graphics;
- light up rectangles in randomly chosen positions;
- draw borders around messages;
- paint the screen white and then “punch” holes in it by using the RESET statement;
- blink the rectangles of light on and off;
- draw mandalas;
- use the INKEY\$ function to input from the keyboard without stopping the program;
- test your keyboard reaction time in response to signals from the screen.

1. Patterns! That’s what we will do in this chapter. We will show you how to put patterns on the screen, using tiny rectangles of light. To do this, we will use the SET statement.

SET turns on a tiny rectangle of light somewhere on the screen.

Somewhere on the screen? Well, of course, we must tell the TRS-80 *where* to turn on the tiny rectangle of light.

SET (64 , 24)
 This far OVER. This far DOWN.

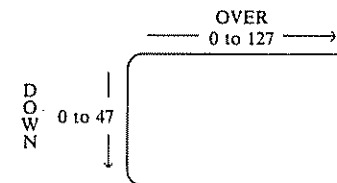


- (a) SET(100,30) tells the TRS-80 to turn on a tiny rectangle of light.
 How far OVER? _____ How far DOWN? _____
- (b) SET(0,0) tells the TRS-80 to turn on a tiny rectangle of light.
 How far OVER? _____ How far DOWN? _____

(a) 100,30; (b) 0,0 (This is the upper left corner of the screen.)

2. Think of it like this: SET(OVER,DOWN)

- OVER can be a whole number, 0 to 127.
- DOWN can be a whole number, 0 to 47.



- (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? _____

(a) 0; (b) 127; (c) 0; (d) 47

For either OVER or DOWN, a value less than zero (0) will cause an error message. For OVER, a value greater than 127 will cause an error message; for DOWN, a value greater than 47 will cause an error message.

In all, there are $128 \times 48 = 6,144$ places to put tiny rectangles of light on the screen.

3. Experiment! Try these to see how SET works. If you are not using a TRS-80 just now, make clever guesses.

Press **CLEAR**, then press **ENTER**.

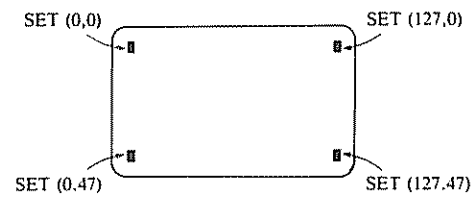
(a) Type SET(0,0) and press **ENTER**. Where is the tiny rectangle?

(b) Type SET(127,0) and press **ENTER**. Where is the tiny rectangle?

(c) Type SET(0,47) and press **ENTER**. Where is the tiny rectangle?

(d) Type SET(127,47) and press **ENTER**. Where is the tiny rectangle?

(a) upper left corner; (b) upper right corner; (c) lower left corner; (d) lower right corner



4. Now, imagine that the screen is the night sky. Pretend that each tiny rectangle of light is a star appearing in the night sky. You make it happen by entering and running this program.

```

100 REM***STARFALL, A PROGRAM BY FIREDRAKE THE DRAGON
110 CLS

200 REM***TURN ON RANDOM STARS
210 OVER = RND(127)
220 DOWN = RND(47)
230 SET(OVER,DOWN)
240 GOTO 210
    
```


When you RUN this program, the "sky" will begin filling with "stars." Try to find . . . constellations, shapes, patterns . . . as stars turn on.

Did you notice? In lines 210, 220, and 230, we used OVER and DOWN as numeric variables! Yes, this is OK. In the past, we have used only single letters as numeric variables. But, if we are careful, we can use words as variables . . . and will do so occasionally from now on.

Beware! You can get into lots of trouble using words as variables. For example, you can't use LETTER as a variable because it contains LET, which is a special, reserved BASIC word. And you can't use SPRINT because it contains PRINT. And you can't use DIFFERENCE.

| | | |
|---------------|---------------|-------------------|
| <u>LETTER</u> | <u>SPRINT</u> | <u>DIFFERENCE</u> |
| ↑ | ↑ | ↑ |
| LET | PRINT | IF |

For a list of reserved words, see Appendix F.

More trouble! *BASIC looks only at the first two letters of a variable.* So it thinks that DISTANCE and DIAMETER are the *same* variable. Of course, you can't use DISTANCE anyway, because it contains TAN, which is a reserved word. But BASIC thinks that ABC and ABD are the same variable . . . and they aren't even words!

Fortunately, neither OVER nor DOWN is a reserved word. And their first two letters are different! So we can use OVER and DOWN as numeric variables.

5. Do you understand everything you know about "Starfall"? If not, follow the arrows (we omit the REMARK statements since the TRS-80 ignores them).

```

100 CLS
↓
210 OVER = RND(127) ←
220 DOWN = RND(47)
↓
230 SET(OVER,DOWN)
↓
240 GOTO 210 —————

```

Line 210 computes a value of OVER in the range 1 to 127.

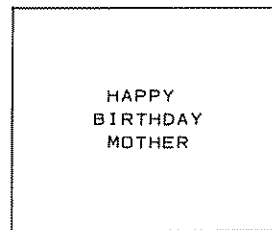
- (a) Line 220 computes a value of DOWN in the range _____ to _____.
- (b) Will OVER ever be zero? _____
- (c) Will DOWN ever be zero? _____
-

(a) 1, 47; (b) no; (c) no

To use the entire screen, change lines 210 and 220 as follows:

```
210 OVER = RND(128)-1      Now OVER will be a random number
                           from 0 to 127.
220 DOWN = RND(48)-1      Now DOWN will be a random number
                           from 0 to 47.
```

6. A birthday card for mother. We want to "paint" a border around the screen, then print HAPPY BIRTHDAY MOTHER, as shown below:

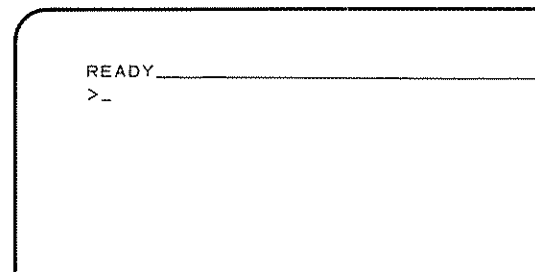
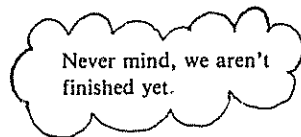


First, let's paint a "line" across the top of the screen.

```
100 REM***A BIRTHDAY CARD FOR MOTHER
110 CLS

200 REM***PAINT A LINE ACROSS THE TOP OF THE SCREEN
210 FOR OVER = 0 TO 127
220   SET(OVER,0)
230 NEXT OVER
```

Try it. The TRS-80 will draw a line across the top of the screen, then print READY, which wipes out part of the line.



You write a FOR-NEXT loop to paint a line across the *bottom* of the screen.

```
300 REM***PAINT A LINE ACROSS THE BOTTOM OF THE SCREEN
```

```
310 _____
```

```
320 _____
```

```
330 _____
```

```
-----
```

```
310 FOR OVER = 0 TO 127
```

```
320   SET(OVER,47)
```

```
330 NEXT OVER
```

7. Let's add a FOR-NEXT loop to draw a line down the right side of the screen. Look for it in lines 400-430:

```
100 REM***A BIRTHDAY CARD FOR MOTHER
```

```
110 CLS
```

```
200 REM***PAINT A LINE ACROSS THE TOP OF THE SCREEN
```

```
210 FOR OVER = 0 TO 127
```

```
220   SET(OVER,0)
```

```
230 NEXT OVER
```

```
300 REM***PAINT A LINE ACROSS THE BOTTOM OF THE SCREEN
```

```
310 FOR OVER = 0 TO 127
```

```
320   SET(OVER,47)
```

```
330 NEXT OVER
```

```
400 REM***PAINT A LINE DOWN THE RIGHT SIDE OF THE SCREEN
```

```
410 FOR DOWN = 0 TO 47
```

```
420   SET(127,DOWN)
```

```
430 NEXT DOWN
```

Your turn again. Paint the line down the left side of the screen.

```
500 REM***PAINT A LINE DOWN THE LEFT SIDE OF THE SCREEN
```

```
510 _____
```

```
520 _____
```

```
530 _____
```

```
-----
```

```
510 FOR DOWN = 0 TO 47
```

```
520   SET(0,DOWN)
```

```
530 NEXT DOWN
```

If you RUN our program, it draws a box around the screen. But then it stops and prints READY, which clobbers part of our box. Aha! Add a do-nothing line, as follows.

```
999 GOTO 999
```

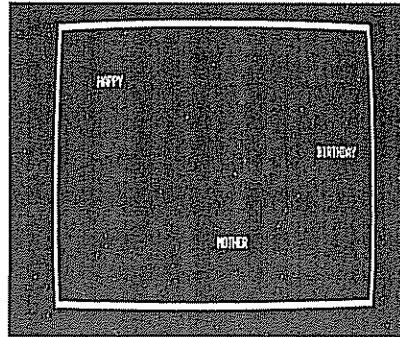
Now RUN the program. When you tire of looking at our box, press **BREAK**.

8. We have the border painted on the screen. Now let's print the birthday message to mother.

```
600 REM***PRINT THE HAPPY BIRTHDAY MESSAGE TO MOTHER
610 PRINT @200, "HAPPY";
620 PRINT @500, "BIRTHDAY";
630 PRINT @800, "MOTHER";

700 REM***DO NOTHING WHILE MOTHER ENJOYS!
710 GOTO 710
```

Hmmmm . . . we ran the entire program. Here is what happened:



We will leave it to you to adjust the numbers following @ in lines 610, 620, and 630 in order to get a "birthday card" more like the one shown in frame 6.

9. Our birthday card program has four FOR-NEXT loops, one for each line on the edge of the screen. Here is a single FOR-NEXT loop to draw the lines across the top and bottom of the screen, at the same time:

```
200 REM***PAINT LINES ACROSS TOP AND BOTTOM OF SCREEN
210 FOR OVER = 0 TO 127
220   SET(OVER,0)
230   SET(OVER,47)
240 NEXT OVER
```

You write a single FOR-NEXT loop to draw the vertical lines at the left and right edges of the screen.

```
300 REM***PAINT LINES DOWN LEFT AND RIGHT EDGES OF SCREEN
```

```
310 _____
320 _____
330 _____
340 _____
```

```

310 FOR DOWN = 0 TO 47
320   SET(0,DOWN)
330   SET(127,DOWN)
340 NEXT DOWN

```

10. Paint the screen white.

```

100 REM***THIS PROGRAM PAINTS THE SCREEN WHITE
110 CLS

200 REM***PAINT THE SCREEN WHITE
210 FOR OVER = 0 TO 127
220   FOR DOWN = 0 TO 47 ]   Inside ]   Outside
230     SET(OVER,DOWN) ]   loop   ]   loop
240   NEXT DOWN
250 NEXT OVER

300 REM***WAIT UNTIL SOMEONE PRESSES 'BREAK'
310 GOTO 310

```

This program paints the screen with vertical stripes, beginning at the left edge. Rewrite lines 210–250 so that the screen is painted with horizontal stripes, beginning at the top.

```

210 _____
220 _____
230 _____
240 _____
250 _____

```

```

-----
210 FOR DOWN = 0 TO 47
220   FOR OVER = 0 TO 127 ]   Inside ]   Outside
230     SET(OVER,DOWN) ]   loop   ]   loop
240   NEXT OVER
250 NEXT DOWN

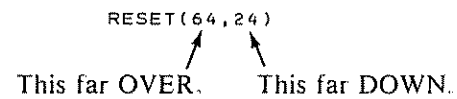
```

DOWN begins at 0 (line 210). For DOWN = 0, lines 220–240 paint a stripe across the top of the screen (OVER goes from 0 to 127). Then DOWN becomes 1 (line 250). For DOWN = 1, lines 220–240 paint a stripe across the screen, just below the previous stripe. And so on . . . for DOWN = 2, 3, 4, . . . , 47.

11. The SET statement turns on a tiny rectangle of light somewhere on the screen. Another statement, called RESET, turns *off* a tiny rectangle of light, if it was on.

—RESET turns off a tiny rectangle of light (if it was on) somewhere on the screen.

As with SET, we must tell the TRS-80 *where* to turn off the tiny rectangle of light.



The following program tells the TRS-80 to blink a tiny rectangle of light on and off at OVER = 64, DOWN = 24 (near the center of the screen):

```

10 CLS
20 SET(64,24)           } Turn light on and
30 GOSUB 70             } use time delay subroutine.
40 RESET(64,24)        } Turn light off and
50 GOSUB 70             } use time delay subroutine.
60 GOTO 20              } Go around again.

70 T = 500
80 FOR Z = 1 TO T : NEXT Z } Time delay.
90 RETURN
    
```

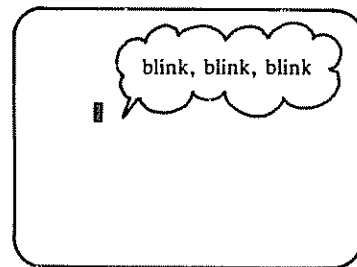
Change lines 20 and 40 so that the light blinks on and off at OVER = 32, DOWN = 12.

```

20 _____
40 _____

-----

20 SET(32,12)
40 RESET(32,12)
    
```



You might want to experiment with the following program. It has several lines with multiple statements.

```

10 CLS
20 INPUT "HOW FAR OVER (0 TO 127)"; OVER
30 INPUT "HOW FAR DOWN (0 TO 47)"; DOWN

40 SET(OVER,DOWN) : GOSUB 70
50 RESET(OVER,DOWN) : GOSUB 70

60 GOTO 40

70 T = 500
80 FOR Z = 1 TO T : NEXT Z : RETURN
    
```

Try some of these values for (OVER,DOWN):

(0,0) (24,0) (50,0) (0,3) (24,3) (24,4)

For each value, type RUN and press **ENTER**. Then enter the values of OVER and DOWN, as requested by the computer. The TRS-80 will blink a tiny rectangle at the place *you* specified. When you are tired of this blinking nonsense, press **BREAK** and . . . if you are not *too* tired, type RUN again.

12. Aha! We *could* have done the first program in the previous frame as follows:

```

10 CLS
20 SET(64,24)
30 GOSUB 70
40 CLS
50 GOSUB 70
60 GOTO 20

70 T = 500
80 FOR Z = 1 TO T : NEXT Z
90 RETURN

```

} Turn on light and
use time delay subroutine.

} Clear entire screen and
use time delay subroutine.

Trouble! To turn off one tiny rectangle of light at (64,24), we *clear the entire screen!*

That just won't work for us if we want to blink a light on and off, while leaving the rest of the screen unchanged. So we give you the following program, which draws a box on the screen, then blinks a light on and off, *inside the box*. The box stays all the time.

```

100 REM***TWINKLE, TWINKLE, LITTLE STAR, INSIDE A BOX

200 REM***DRAW THE BOX
210 CLS
220 FOR OVER = 59 TO 69
230   SET(OVER,19)
240   SET(OVER,29)
250 NEXT OVER
260 FOR DOWN = 19 TO 29
270   SET(59,DOWN)
280   SET(69,DOWN)
290 NEXT DOWN

300 REM***BLINK 'STAR' IN CENTER OF BOX
310 SET(64,24) : GOSUB 910
320 RESET(64,24) : GOSUB 910
330 GOTO 310

900 REM***TIME DELAY SUBROUTINE
910 T = 500
920 FOR Z = 1 TO T : NEXT Z : RETURN

```

} Draw top of box.
} Draw bottom of box.

} Draw left side of box.
} Draw right side of box.

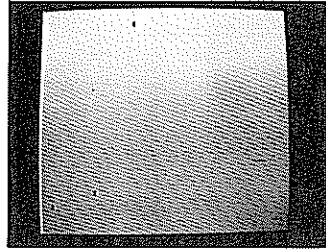
} Light on.
} Light off.

13. OK, it is your turn. Before you proceed, look at "Starfall" in frame 4 and also at frame 10.

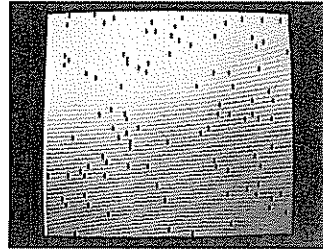
Are you back? Yes? Good. Write a program to:

- (a) Paint the entire screen white (frame 10).
- (b) Fill the white sky with random black holes. Use RESET to "punch black holes" in the white sky.

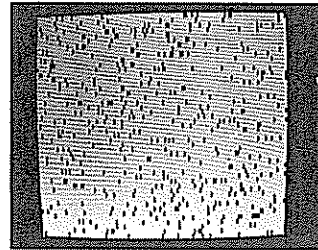
A RUN of your program might look like this:



Just after painting
the screen white



A little later



Much later

Write your program in the space below:

```
100 REM***WHITE SKY, BLACK HOLES
```

```
200 REM***PAINT THE SKY WHITE
210 CLS
220 FOR OVER = 0 TO 127
230   FOR DOWN = 0 TO 47
240     SET(OVER,DOWN)
250   NEXT DOWN
260 NEXT OVER
```

```
300 REM***TURN DN RANDOM 'BLACK HOLES'
310 OVER = RND(128)-1
320 DOWN = RND(48)-1
330 RESET(OVER,DOWN)
340 GOTO 310
```

We are now using more REM statements so
that *you* can more easily read our programs.



14. A mandala is a pattern used in meditation. Usually, a mandala is symmetric in some way. A giant snowflake is beautifully symmetric in all directions about the center. Snowflakes are great mandalas, but they melt all too soon. We call the following program "Mandala, Ever Changing" because it puts an ever-changing symmetric pattern on the screen.

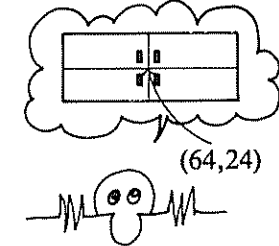
```

100 REM***MANDALA, EVER CHANGING
110 CLS

200 REM***FOUR RANDOM LIGHTS, SYMMETRIC ABOUT THE CENTER
210 X = RND(64)-1
220 Y = RND(24)-1
230 SET(64 + X, 24 + Y)
240 SET(64 + X, 24 - Y)
250 SET(64 - X, 24 + Y)
260 SET(64 - X, 24 - Y)

300 REM***DELAY, THEN CONTINUE MANDALA
310 T = 100
320 FOR Z = 1 TO T : NEXT Z
330 GOTO 210

```



RUN this program. See this: The computer puts four lights on the screen at a time. They are *symmetric* about the center of the screen (64,24).

If you don't see this, increase the time delay. Change line 310 to:

```
310 T = 500
```

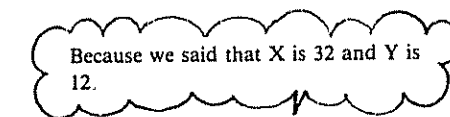
then RUN the program again.

Live long and prosper!

15. Lines 210 and 220 compute random values for X and Y.

- What are the *possible* values for X? From _____ to _____.
- What are the *possible* values for Y? From _____ to _____.

Lines 230–260 turn on four lights. Assume that X is 32 and Y is 12. Therefore, line 230 will turn on a light at (96,36) as follows:



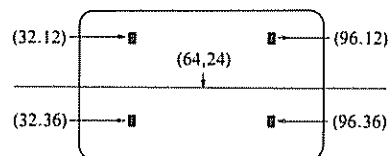
```

SET(64 + X, 24 + Y)   SET(64 + 32, 24 + 12)   SET(96,36)

```

- Line 240 will turn on a light at (_____,_____)
- Line 250 will turn on a light at (_____,_____)
- Line 260 will turn on a light at (_____,_____)

- (a) 0 to 63; (b) 0 to 23
- (c) 96,12 (because $64 + X = 96$ and $24 - Y = 12$)
- (d) 32,36 (because $64 - X = 32$ and $24 + Y = 36$)
- (e) 32,12 (because $64 - X = 32$ and $24 - Y = 12$)



16. If *both* X and Y are zero (unlikely, but possible), all four lights will turn on at the same place, at (64,24) near the center of the screen. Of course, you will see only one light!

Suppose X is 63 and Y is 23. These are the largest values they may have. Where will the lights appear?

- (a) Line 230 will put a light at _____
- (b) Line 240 will put a light at _____
- (c) Line 250 will put a light at _____
- (d) Line 260 will put a light at _____

-
- (a) (127,47); (b) (127,1); (c) (1,47); (d) (1,1)

Note that "Mandala" will never turn on a light at (0,0) or (0,23) or (73,0). The values within the SET statements will never be zero. For example:

$$260 \text{ SET } (64 - X, 24 - Y)$$

↑
↑
 Possible values Possible values
 are 1 to 64. are 1 to 24.

Why? We leave that question to you.

17. Suppose we start near the center of the screen and . . . meander. Maybe left, maybe right, maybe up, maybe down. Everywhere we go, we turn on a light to show where we have been.

```

100 REM***MEANDER ABOUT THE SCREEN
110 CLS

200 REM***START NEAR THE CENTER OF THE SCREEN
210 OVER = 64
220 DOWN = 24

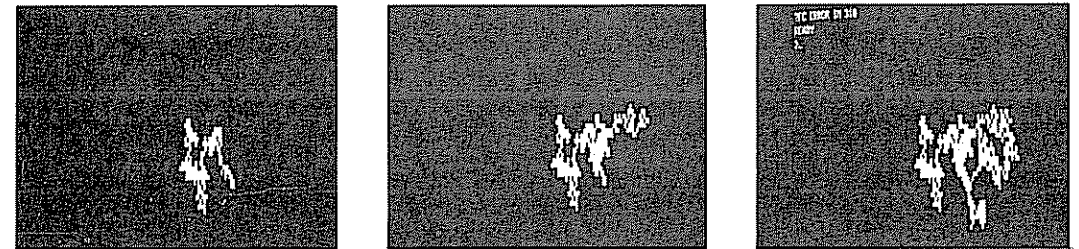
300 REM***TURN ON LIGHT AT (OVER, DOWN)
310 SET(OVER,DOWN)

400 REM***CHANGE OVER & DOWN BY +1, 0 OR -1
410 A = RND(3)-2
420 OVER = OVER + A
430 B = RND(3)-2
440 DOWN = DOWN + B

500 REM***GO AROUND AGAIN FOR A NEW LITE
510 GOTO 310

```

We ran the program. Here is what happened:



When you RUN this program, it eventually will stop with the message: ?FC ERROR IN 310. Read on!

18. The pattern starts near the middle of the screen at (64,24).

```

210 OVER = 64
220 DOWN = 24

```

Line 310 turns on a light at (OVER,DOWN). Then, in lines 410-440, the values of OVER and DOWN might be changed slightly.

- (a) In line 410, what are the possible values of RND(3)? _____
- (b) What are the possible values of A? _____ (Remember, A = RND(3) - 2)

- (c) Suppose, in line 410, A is zero (0). What happens to the value of OVER in line 420? _____
- (d) Suppose, in line 410, A is one (1). What happens to the value of OVER in line 420? _____
- (e) Suppose, in line 410, A is minus one (-1). What happens to the value of OVER in line 420? _____

-
- (a) 1, 2, or 3
 - (b) -1, 0, or 1 (1 - 2 = -1, 1 - 1 = 0, 2 - 1 = 1)
 - (c) It is not changed: OVER = OVER + 0
 - (d) It is increased by 1: OVER = OVER + 1
 - (e) It is decreased by 1: OVER = OVER + (-1)

19. Similarly, the value of DOWN is changed in lines 430 and 440. So OVER and DOWN get bigger or less. Eventually:

- OVER might become bigger than 127 or less than 0.
- DOWN might become bigger than 47 or less than 0.

If any of the above happen, the TRS-80 stops with:

```
7FC ERROR IN 310
```

Let's fix that. Add these statements to the program in frame 17:

```
423 IF OVER > 127 THEN OVER = OVER - 1
427 IF OVER < 0 THEN OVER = OVER + 1
```

In other words, if OVER is too big (greater than 127), reduce it by 1. If OVER is too small (less than zero), increase it by 1.

We fixed OVER. You fix DOWN.

433 _____

437 _____

```
433 IF DOWN > 47 THEN DOWN = DOWN - 1
437 IF DOWN < 0 THEN DOWN = DOWN + 1
```

With the two "fixes," the program will wander all day! Occasionally, it may seem to bounce along the edge of the screen.

20. We return to program "Starfall," from frame 4. Here it is again, with something added: the INKEY\$ function in line 310.

```

100 REM***STARFALL, A PROGRAM BY FIREDRAKE THE DRAGON
110 CLS

200 REM***TURN ON A RANDOM STAR
210 OVER = RND(128)-1
220 DOWN = RND(480-1)
230 SET(OVER,DOWN)

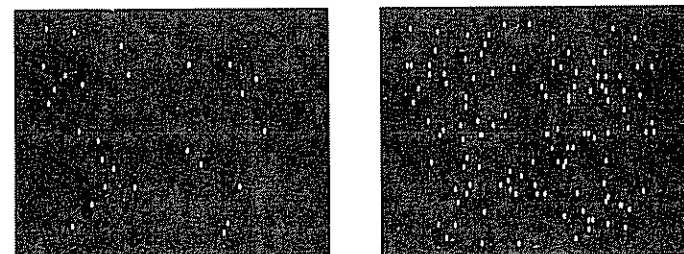
300 REM***IF THE A KEY IS NOT PRESSED, GO BACK FOR NEW STAR
310 IF INKEY$ <> "A" THEN 210

400 REM***THIS HAPPENS IF THE "A" KEY IS PRESSED
410 T = 2000
420 FOR Z = 1 TO T : NEXT Z
430 GOTO 210

```

This time delay is about four seconds.

RUN it. The screen will begin filling with stars. Press the A key. "Starfall" pauses for about four seconds, then continues. Press the A key. "Starfall" pauses for about four seconds, then continues. The first two times, the screen *might* look like this:



As usual, press **BREAK** to *stop* all this!

21. The INKEY\$ function scans the keyboard. If you press a key while INKEY\$ is scanning the keyboard, then the value of INKEY\$ will be a one-character string—the character on the key that you pressed.

Now, look at line 310 in the previous frame. The statement:

```
310 IF INKEY$ <> "A" THEN 210
```

tells the TRS-80:

- If someone has *not* pressed the A key on the keyboard, then go to line 210.
- However, if the A key has been pressed, go to the next line number following 310.

Well, following line 310, we have a time delay. So *nothing* happens on the screen for about four seconds while the TRS-80 counts from 1 to 2,000. After this delay, the TRS-80 goes to line 210 (see line 430) and continues putting stars on the screen.

(a) Rewrite line 310 so that pressing the **Z** key interrupts the program for about four seconds.

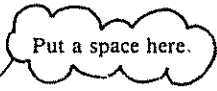
310 _____

(b) This one is tricky! If you don't know the answer, guess. Rewrite line 310 so that pressing the *space bar* interrupts the program for about four seconds.

310 _____

(a) 310 IF INKEY\$ <> "Z" THEN 210

(b) 310 IF INKEY\$ <> " " THEN 210



The following will not work:

```
310 IF INKEY$ = "SPACE" THEN 210
```

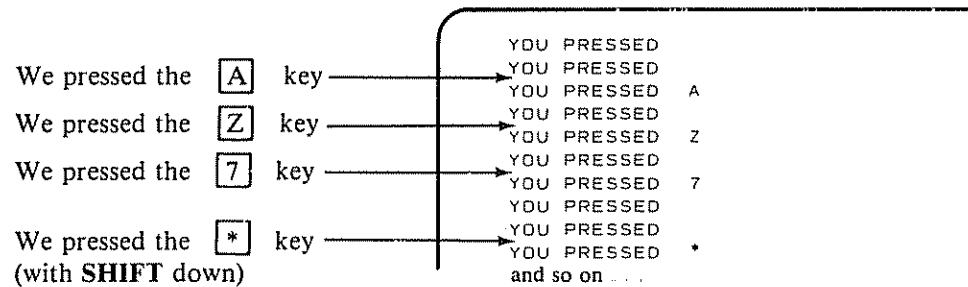
The value of INKEY\$ can be *one*, and only one, character. For this same reason, you cannot use the following:

```
310 IF INKEY$ <> "ENTER" THEN 210
```

22. Experiment. Try this short program, which uses INKEY\$.

```
10 CLS
20 PRINT "YOU PRESSED " INKEY$
30 T = 100
40 FOR Z = 1 TO T : NEXT Z
50 GOTO 20
```


Here is what happened when we ran the program:



If we *don't* press a key, the TRS-80 prints YOU PRESSED followed by an empty space.

If we *do* press a key, the TRS-80 tells us which key we pressed.

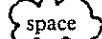
(a) Suppose we press the [/] key. What will the TRS-80 print?

(b) Suppose we hold the **SHIFT** down and press the  key. What will the TRS-80 print?

(c) Suppose we press the space bar. What will the TRS-80 print?

(a) YOU PRESSED /

(b) YOU PRESSED +

(c) YOU PRESSED  but you don't see the space.

23. Increase the value of T in line 30 to 2,000 (30 T = 2000); this will give about a four-second delay. RUN the program again and follow these instructions:

—*Very quickly*, press **A**, then press **B**. The TRS-80 will probably print
YOU PRESSED B.

—*Very quickly*, press **A**, then press **B**, then press **C**. The TRS-80 will probably print YOU PRESSED C.

The value of INKEY\$ will be the *last* key you pressed just a tiny fraction of a second before the TRS-80 executed the INKEY\$ function in line 20.

Try these:

(a) Press two keys at the same time. What happens? _____

(b) Hold a key down for awhile. Does it repeat? _____

(a) Only one key is printed.

(b) No. It prints once. To make a key repeat, you have to press it repeatedly.

Variation. Change line 50 to:

```
50 GOTO 10
```

24. Here is a way to use almost any key to interrupt "Starfall," or any other program.

```
310 IF INKEY$ = "" THEN 210
```

Huh? There is *nothing* between quotation marks—not even a space.



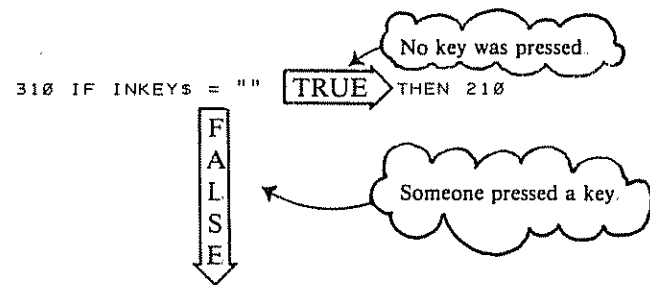
Line 310 says: If the value of INKEY\$ is "nothing," then go to line 210. In other words, if *no* key has been pressed since the last time we used INKEY\$, then go to line 210.

Each time INKEY\$ looks at the keyboard, *it first erases its previous value* . . . it is "empty." Then, if someone has pressed a key, INKEY\$ remembers that key as its value. If someone has pressed a bunch of keys, INKEY\$ remembers the *last key pressed*.

But, if *no* key has been pressed, INKEY\$ remains "empty." In this case, the condition INKEY\$ = "" is *true* and the TRS-80 goes to line 210.

However, if someone has pressed a key (almost any key), then the condition INKEY\$ = "" is *false* because the value of INKEY\$ will now be the key that was pressed. In this case, the TRS-80 moves on to lines 400, 410, etc.

Follow the arrows:



Remember, in line 310, there is *nothing* between quotation marks. Since we use quotation marks to enclose strings, you can think of this ("") as an "empty string."

25. Here again is "Starfall," using line 310 from the previous frame:

```

100 REM***STARFALL, A PROGRAM BY FIREDRAKE THE DRAGON
110 CLS

200 REM***TURN ON A RANDOM STAR
210 OVER = RND(128)-1
220 DOWN = RND(48)-1
230 SET(OVER,DOWN)

300 REM***IF NO KEY IS PRESSED, GO BACK FOR ANOTHER STAR
310 IF INKEY$ = "" THEN 210

400 REM***YOU GET HERE IF A KEY IS PRESSED
410 T = 2000
420 FOR Z = 1 TO T : NEXT Z
430 GOTO 210
  
```

Try it. You can press almost any key to cause the TRS-80 to pause for a few seconds. The only keys that don't do this are **BREAK**, which *stops* the TRS-80 "for good," and **SHIFT**, which does nothing. However, you can use **SHIFT** with another key to cause the TRS-80 to pause. Experiment!

Now *replace* lines 400–430 with the following:

```
400 REM***
410 IF INKEY$ = "" THEN 410 ELSE 210
```

RUN the modified program.

- Press any key (except **BREAK** or **SHIFT**). The TRS-80 will pause while you stargaze.
- Stargaze for as long as you wish. When you are ready for more stars, press any key (except **BREAK** or **SHIFT**). The TRS-80 will continue putting stars on the screen.

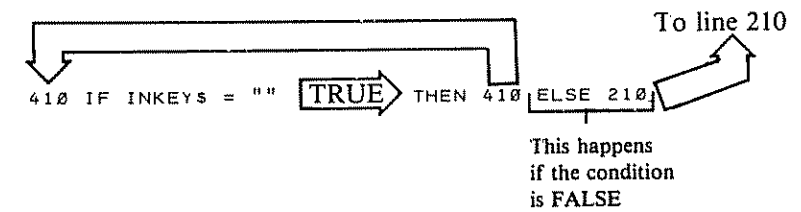
Do the above as often as you wish. Then, read on . . . We will explain IF . . . THEN . . . ELSE . . .

26. The statement:

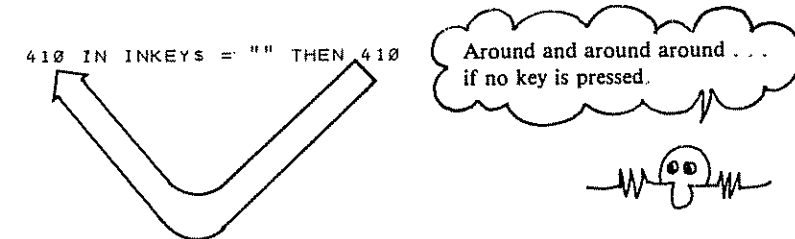
```
410 IF INKEY$ = "" THEN 410 ELSE 210
```

tells the computer: If INKEY\$ is empty (no key has been pressed), then go to line 410. However, if INKEY\$ is not empty (a key has been pressed), then go to line 210.

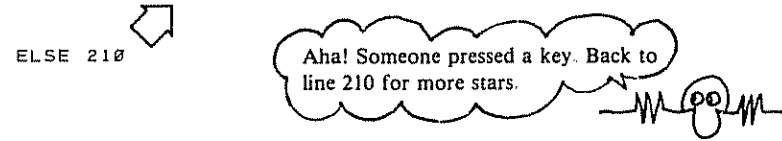
Follow the arrows:



As long as you *don't* press a key, the first part of line 410 will go around and around and around.



But, if you *do* press a key, the computer moves on to the ELSE part of the statement.



Lines 310 and 410 work together to let us stop and restart "Starfall."

```
310 IF INKEY$ = "" THEN 210
410 IF INKEY$ = "" THEN 410 ELSE 210
```

We will use this method frequently . . . so spend some time learning it.

27. The following "Reaction Time" program uses INKEY\$ several times.

```
100 REM***REACTION TIME PROGRAM
200 REM***INSTRUCTIONS TO THE PLAYER
210 CLS
220 PRINT "HOW FAST ARE YOU? I WILL CLEAR THE SCREEN FOR A"
230 PRINT "LITTLE WHILE, THEN COUNT NEAR THE MIDDLE OF THE"
240 PRINT "SCREEN. WHEN I START COUNTING, PRESS THE SPACE BAR"
250 PRINT "AND I WILL STOP. STOP ME QUICKLY, IF YOU CAN!"
260 PRINT : PRINT "WHEN YOU ARE READY, PRESS ANY KEY.."
270 IF INKEY$ = "" THEN 270

300 REM***CLEAR THE SCREEN FOR A RANDOM TIME, T
310 CLS
320 T = RND(2000)
330 FOR Z = 1 TO T : NEXT Z

400 REM***START COUNTING, SPACE BAR STOPS IT.
410 X = 1
420 PRINT @472,X
430 IF INKEY$ <> " " THEN X = X + 1 : GOTO 420

500 REM***PLAYER PRESSED SPACE BAR. PAUSE, THEN PLAY AGAIN.
510 T = 2000
520 FOR Z = 1 TO T : NEXT Z
530 GOTO 210
```

The next few frames refer to the above program. But, before you do them, first enjoy the game! Play it several times. An average of 10 is fast—congratulations! If your average is more than 20, well . . . maybe you are thinking about something else.

Hmmmm . . . we played the game several times and discovered a way to cheat. We can stop the computer with a count of 1 every time! 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 every time, just by pressing the space bar. Later, we will share our discovery with you, then show you how to fix the "bug" in the program so that this kind of cheating can't happen.

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! This error almost escaped our notice—imagine the letters we might have received if we had missed it!

28. Look at line 270:

```
270 IF INKEY$ = "" THEN 270
```

- (a) If no one presses a key, what happens? _____

- (b) If someone presses a key, what happens? _____

- (a) Line 270 repeats itself. If no key is pressed, the value of INKEY\$ is the empty string. Therefore, `INKEY$ = ""` is *true*, and the TRS-80 goes to line 270 again. Around and around and around . . . until someone presses a key.
- (b) The TRS-80 moves on to the lines following line 270. These are lines 300, 310, 320, etc.

29. OK, the screen goes blank for a random time, which depends on the value at T in line 320. Then the computer starts counting—putting numbers one after the other in the center of the screen. This is controlled by lines 410–430.

```
410 X = 1
420 PRINT @470, X
430 IF INKEY$ <> " " THEN X = X + 1 : GOTO 420
```



space

- (a) What is the first number printed? _____
- (b) If the space bar is *not* pressed, what happens (line 430)? _____

- (c) If the space bar is pressed, what happens? _____

- (a) 1 (because line 410 says $X = 1$, which is printed by line 420).
- (b) The computer increases the value of X by 1 and then goes to line 420, where the new value is printed.
- (c) Everything following THEN in line 430 is *not* done. The computer moves on to the lines following line 430. After a time delay of a few seconds, the game starts again.

30. Have you figured out how to beat our "Reaction Time" program and always get a score of 1?

Here's how. When the TRS-80 prints WHEN YOU ARE READY, PRESS ANY KEY, 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 very unlikely that anyone can press the space bar while the computer is going from line 340 to line 410.

Before you begin *reacting* to Chapter 7, try our friendly Self-Test, which follows.

SELF-TEST

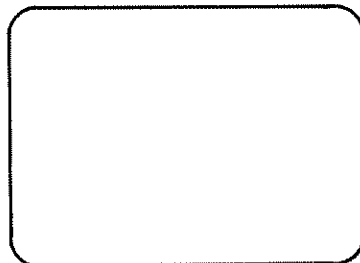
Try this Self-Test, so you can evaluate how much you have learned in the chapter.

1. Which of the following are *not* legal statements?

- (a) SET(47,52) (b) SET(-30,0)
- (c) SET(100,40) (d) SET(127,32)

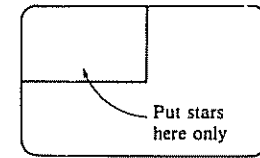
2. Sketch the display for this program.

```
100 CLS
110 SET(24,5) : SET(12,10)
120 SET(12,5) : SET(24,10)
130 GOTO 130
```



3. Complete this program to light stars in the *upper left quarter* of the screen *only*.

```
100 CLS
110 OVER = RND( )-1
120 DOWN = RND( )-1
130      (OVER,DOWN)
140 GOTO 110
```



4. Write a program to connect the following points on the screen so that a rectangle will be formed.

```
OVER 60, DOWN 10      OVER 80, DOWN 10
OVER 80, DOWN 20      OVER 60, DOWN 20
```

5. Examine the following program and answer the questions that follow it.

```

100 CLS
110 DOWN = 10
120 FOR OVER = 10 TO 20
130   SET(OVER,DOWN)
140 NEXT OVER
150 A = RND(11)-1
160 FOR W = 1 TO 10 : NEXT W
170 RESET(A + 10,DOWN)
180 FOR W = 1 TO 10 : NEXT W
190 SET(A + 10,DOWN)
200 GOTO 150

```

- (a) Describe the result of executing lines 120-140. _____

- (b) What is the range of values for A? _____ to _____
- (c) Will the point that is RESET in line 170 be on the original line that was drawn? _____
- (d) Describe the result of executing lines 150-190. _____

6. Here is a portion of a program.

```

200 CLS
210 OVER = RND(128)-1
220 DOWN = RND(48)-1
230 SET(OVER,DOWN)
240 IF INKEY$ <> "S" THEN 210
250 GOTO 200

```

Suppose we enter and RUN this program.

- (a) What happens if we don't press the S key? _____

- (b) What happens if we press the S key? _____

7. Refer back to the "Reaction Time" program in frame 27. We are going to shorten the program and let you complete some lines to do ten reaction tests at a time. We have left out the instructions since they are the same. Remember, though, to use the space bar to stop the count.

```

210 CLS
220 PRINT "WHEN YOU ARE READY, PRESS ANY KEY."
230 IF INKEY$ = "" THEN 230

300 FOR N = _____ TO _____
310 CLS
320 T = RND(2000)
330 FOR Z = 1 TO T : NEXT Z

410 X = 1
420 PRINT @472,X
430 IF INKEY$ <> " " THEN X = X + 1 : GOTO 420

510 FOR Z = 1 TO 2000 : NEXT Z
520 _____

```

Please complete
the FOR-NEXT loop.

8. It would be helpful to have a measure of reaction times for the ten tries in the program of frame 7. Add to that program some lines to:

—Add the reaction time for the tests.

—Compute and print the *average* reaction time for the 10 tests.

Remember to initialize the sum *before* the FOR-NEXT loop. Compute and print the average *after* the FOR-NEXT loop. Show the lines (with line numbers) that you would add to the program of question 7.

9. Study this program and figure out what would happen if you entered it into the computer and ran it. Then answer the questions.

```

100 REM***SCRIBBLE ON THE SCREEN
110 CLS
120 OVER = 64
130 DOWN = 24

200 REM***TURN ON A LIGHT AT OVER,DOWN
210 SET(OVER,DOWN)

300 REM***WAIT FOR SOMEONE TO PRESS A KEY
310 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 BACK TO TURN ON A LIGHT
610 GOTO 210

```

- (a) After you type RUN and press ENTER, what happens first (lines 100-310)? _____
- (b) You press the R key several times. What happens? _____
- (c) You press the U key several times. What happens? _____
- (d) You press the L key several times. What happens? _____
- (e) You press the D key several times. What happens? _____
- (f) You press the key *other than R, L, U, or D*. What happens? _____

Answers to Self-Test

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

- 1. (a) SET(47,52) (b) SET(-30,0) (frame 2)
- 2.  (frames 1-3)
- 3. 110 OVER = RND(64)-1 (gives 0-63)
 120 DOWN = RND(24)-1 (gives 0-23)
 130 SET(OVER,DOWN) (frames 4, 5)
- 4. 100 REM***CONNECT (60,10), (80,10), (80,20), (60,20)
 110 CLS

 200 REM***DRAW HORIZONTAL LINES
 210 DOWN = 10
 220 FOR OVER = 60 TO 80
 230 SET(OVER,DOWN)
 240 SET(OVER,DOWN + 10)
 250 NEXT OVER

 300 REM***DRAW VERTICAL LINES
 310 OVER = 60
 320 FOR DOWN = 10 TO 20
 330 SET(OVER,DOWN)
 340 SET(OVER + 20,DOWN)
 350 NEXT DOWN

 400 REM***WAIT UNTIL SOMEONE PRESSES 'BREAK'
 410 GOTO 410 (frames 6-9)

5. (a) 11 points are set at OVER = 10 through 20 at 10 spaces down.
 (b) 0 through 10
 (c) yes
 (d) a random point on the bar of light goes out and then reappears.
 (frames 11-13)
6. (a) The screen will slowly fill with random stars. Lines 210-240 will be executed again and again and again.
 (b) The computer will execute line 250, which sends it to line 200. The screen will be cleared and then start filling with stars until someone presses the **S** key again.

Note: Lines 240 and 250 could be replaced with a single IF . . . THEN . . . ELSE, as follows:

```
240 IF INKEYS <> "S" THEN 210 ELSE 200
```

(frames 4, 20, 21)

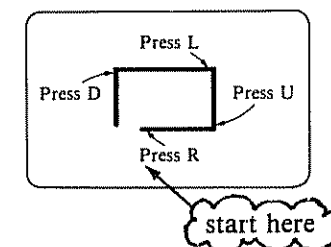
7. 300 FOR N = 1 TO 10
 520 NEXT N (frame 27)
8. Our changes:

```
240 SUM = 0
440 SUM = SUM + X
530 AV = SUM/10
540 PRINT "YOUR AVERAGE IS" : AV
```

9. (a) The screen is cleared, then a single "blip" of light comes on at (64,24), near the center of the screen. Then the computer waits for someone to press a key (line 310).
 (b) The computer "paints" a line to the right, turning on one blip for each time you press **R**.
 (c) The computer "paints" a line upward, turning on one blip for each time you press **U**.
 (d) The computer paints a line to the left, turning on one blip for each time you press **L**.
 (e) The computer paints a line downward, turning on one blip for each time you press **D**.
 (f) Nothing. Only the **R**, **U**, **L**, and **D** keys control the computer.

Note: Following (b), (c), (d), and (e) above, the screen might look something like this:

(frames 17-19, 21-24)



CHAPTER SEVEN

Entering and Displaying Data

In this chapter, you will learn some new ways to enter data and some new ways to display information on the video display. We will explore different methods of assigning values to variables. These new techniques will provide you with more versatility and freedom and improve your programming capabilities.

When you have finished this chapter, you will be able to:

- input data from a READ statement;
- use a DATA statement to provide information to be READ as needed in your program;
- control the data READ statement with a FOR-NEXT loop;
- use the TAB function to control output displays;
- use strings in READ and DATA statements;
- mix strings and numerics in READ and DATA statements;
- use the RESTORE statement to change the data pointer to the beginning of a DATA list;
- use the POS(X) statement to provide a specific number of spaces between bits of information.

1. You have been using LET and INPUT statements to assign values to variables. Another method that can be used combines the READ and DATA statements. The READ statement tells the computer to read a value from a data list. The DATA statement provides the list of data values. The two are always used together.

```
10 CLS
100 READ A
110 PRINT A;
120 GOTO 100
130 DATA 99,33,88,-44,102.3,75,29.95,99999
```

The statement 100 READ A tells the computer to read one value from the data list (given in line 130) and assign that value to the variable A. Every time line 100 is executed, the computer assigns the *next* value in the data list to A until all the data in the list have been used. The computer uses a *pointer* to keep track of the

data item that is to be read. Each time an item of data is used, the pointer moves on to the next item.

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

```

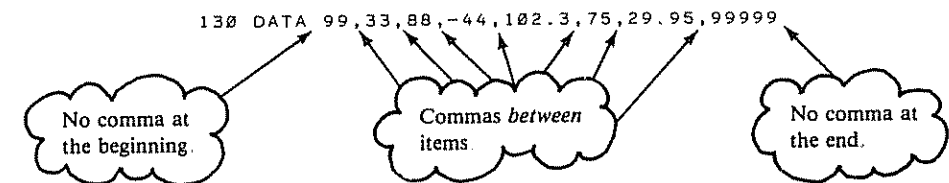
99 33 88 -44 102.3 75 29.95 99999
?OD ERROR IN 100
    
```

The ERROR statement (?OD ERROR IN 100) means "out of data in line 100."

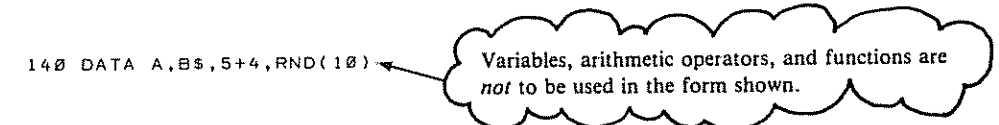
- (a) How many values are in the DATA statement? _____
- (b) Were they all printed? _____
- (c) Why do you suppose the RUN shows an OD error? _____

(a) 8; (b) yes; (c) The end of the data list was reached. The data pointer does not have a new value to point to.

2. Examine the format of the DATA statement in line 130. Commas are used to *separate* the data items in the list.



DATA statements may contain integers, decimal fractions, and numbers in floating point notation. *Do not* use any of the expressions contained in the following example. Each item is *illegal* in the form used.



Write a DATA statement using these values:

| | | |
|------|--------|------|
| 12.6 | 123.45 | .123 |
| -22 | 3105 | .045 |

130 DATA _____

130 DATA 12.6,-22,.123.45,3105,.123,.045

Your line number and order of the values may differ. You may also have spaces that we do not use. Remember, do *not* use commas in long numbers such as 3105. Why not? Because the computer will treat 3,105 as *two* numbers, 3 and 105.

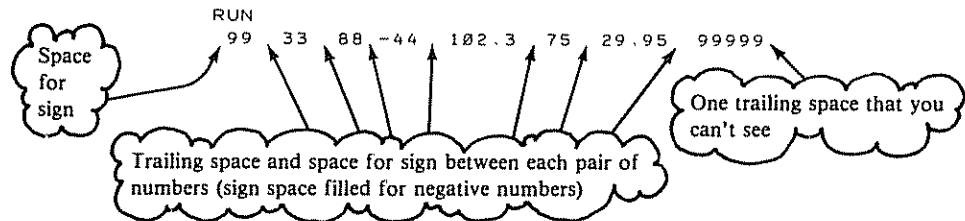
3. Study the following DATA statement and find the errors.

130 DATA, 12,3,-20;14,150,-.04,

The errors are: _____

The errors are: comma following the word DATA
 semicolon between -20 and 14
 comma at end of statement

4. The PRINT statement used in frame 1 grouped the numbers very tightly. The TRS-80 follows a predetermined format in printing numeric values. One space is reserved at the front of a number for its sign (+ or -). However, if a number is positive, the + sign is not printed; instead a space is printed. Then the digits and a decimal point (if one is required) are printed. Last of all, a trailing space is provided. Take another look at the output of the run in frame 1.



For the DATA statement below, show the spacing of the numbers when a program containing this DATA statement is run.

```
130 DATA 12.6,-22,123.45,3105,.123,-.045
```

RUN

```
_____
```

```
_____
```

```
12.6 -22 123.45 3105 .123 -.045
```

5. Let's eliminate the out-of-data error in the program in frame 1 by making use of the last data item (99999) as a *flag** that tells the computer that it is at the end of the data list. Instead of displaying the data, let's turn our computer into an expensive adding machine by changing lines 100 and 130.

```
100 CLS : T = 0
110 READ A
120 IF A = 99999 THEN 160
130 T = T + A : PRINT "TOTAL NOW = " ; T
140 GOTO 110
150 DATA 99,33,88,-44,102.3,75,29.95,99999
160 END
```

The computer now reads each data item and adds it to the sum of the previous values. It then prints the new sum. Note how line 120 avoids the addition of 99999 to the sum.

- (a) How many data items will be read? _____
- (b) How many data items will be added? _____
- (c) How many times is line 140 executed? _____

(a) 8; (b) 7; (c) 7

6. If lines 130 and 120 were interchanged in the program in frame 5:

```
120 T = T + A : PRINT "TOTAL NOW = " ; T
130 IF A = 99999 THEN 160
```

- (a) How many data items would be added? _____
- (b) How many times would line 140 be executed? _____

(a) 8; (b) still only 7

*We covered the flag in Chapter 5, frames 16 and 17.

7. More than one item may be read from a data list by a single READ statement. Commas are then used to separate the items to be read.

```
100 CLS
110 READ A,B
120 IF B = 99999 THEN 160
130 T = A + B : PRINT A "+" B "=" T
140 GOTO 110
150 DATA 99,33,88,-44,102.3,75,29.95,99999
160 END
```

Notice that the *variable B* is tested in line 120 rather than A since the data are read in pairs (the last pair will assign 29.95 to A and 99999 to B).

- (a) How many pairs of data will be READ? _____
- (b) How many sums will be calculated? _____

-
- (a) 4 pairs
 - (b) 3 sums (The last *pair* will not be added because of line 120.)

8. Here is another way to write the program in frame 7:

```
100 CLS
110 READ A,B
120 T = A + B : PRINT A "+" B "=" T
130 IF B <> 99999 THEN 110
140 DATA 99,33,88,-44,102.3,75,29.95,99999
150 END
```

- (a) How many pairs of data will be READ? _____
- (b) How many sums will be calculated and printed? _____

-
- (a) 4 pairs; (b) 4 sums

9. Remember the "Starfall" program by Firedrake the Dragon in Chapter 6? Draco, another dragon, has modified Firedrake's program to use the READ and DATA statements to light up the stars.

```
100 REM***STARFALL, A PROGRAM BY FIREDRAKE THE DRAGON
110 CLS

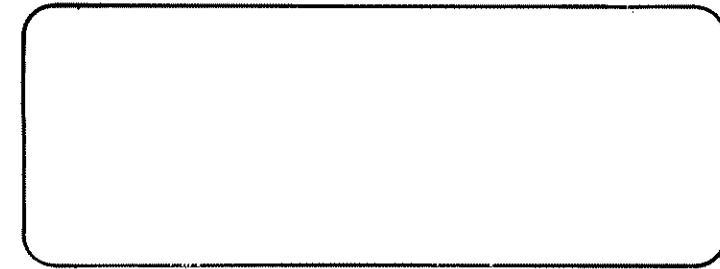
200 REM***DRACO'S MODIFICATION
210 FOR X = 1 TO 14
220 READ OVER,DOWN
230 SET(OVER,DOWN)
240 NEXT X
250 DATA 27,22,29,24,27,25,24,23,40,16,44,15,45,19
260 DATA 39,23,39,26,39,29,41,31,53,30,64,28,69,29

300 GOTO 300
310 REM***THIS IS THE END
```

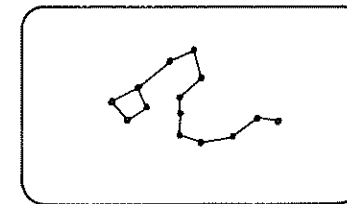
On December 1, 1940, Draco was gazing northward into a clear sky from a spot near his cave in Kelgarth. He saw a similar pattern of stars and gave the shape a name.

Draw a sketch of the pattern displayed on the screen when the above program is run. Then connect the points ("dot-to-dot") in the order in which they were read from the DATA statement. With a little imagination you will know what our dragon named this constellation.

What was the name? _____



The name was DRACO.



10. This program uses a FOR-NEXT statement to READ in and PRINT the data items—three at a time. We do not need a test for the last item since the FOR-NEXT loop runs a definite number of times. Complete the statement in line 110 and write out the printed format of the run on the lines that follow.

```
100 CLS
110 FOR _____
120 READ A,B,C
130 PRINT A;B;C
140 NEXT X
150 DATA 6,-41,2.3E+08,55.3,12345,16,.02,1.5,-43
```

2.3E+08 is a floating point number, described in Appendix C.

RUN

110 FOR X = 1 TO 3

```

RUN
 6 -41 2.3E+08
55.3 12345 16
.02 1.5 -43
READY
>_
    
```

11. Change line 130 to:

130 PRINT A,B,C

Show the output of the modified program. Don't show the READY, prompt, and cursor.

RUN

```

RUN
 6          -41          2.3E+08
55.3       12345        16
.02        1.5         -43
    
```


12. The printed output of the program in frame 10 may be displayed in a more organized way by using the TRS-80 TAB function. This function works almost like the TABs on an ordinary typewriter, allowing you to indent a desired number of spaces in from the left margin. It is used in PRINT statements. Study the format of the TAB function in the following PRINT statement:

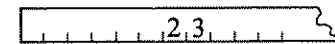
```
130 PRINT TAB(5) A TAB(20) B TAB(35) C
```

The values in parentheses tell the computer the position (on the line being printed) where the value of the variable is to be printed. These values in parentheses may be numbers or numerical expressions.

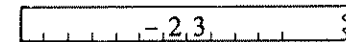
If line 130 above were executed in a program, which value would be printed on the left? _____ How far in would it be indented? _____

The value of A would be printed on the left, 5 spaces in from the left margin. Remember, though, that if the value of A is a positive number or zero, the printed value will begin with a space. If the value of A is negative, the printed value will begin with a minus sign (-). For example:

A = 23



A = -23



13. The number of spaces in a TAB function may range from 0 to 255. Remember, though, that one line on the screen has only 64 printing positions (Chapter 4, frame 14). Usually, we use only 0 through 63 as the number of spaces in a TAB function.

- TAB(0) is the left edge of a line on the screen.
- TAB(31) is about the middle of a line on the screen.
- TAB(63) is the right edge of a line on the screen.

Hmmm . . . what happens if we use a number bigger than 64 in a TAB function? Well, something strange happens. The computer will divide the number by 64, ignore the quotient, and use the remainder as the number of spaces. Huh? OK, suppose you try to TAB(100):

$$64 \overline{) 100} \\ \underline{64} \\ 36$$

The TRS-80 will TAB(36).

- TAB(64) is the same as TAB(0).
- TAB(127) is the same as TAB(63).

- TAB(128) is the same as TAB(0).
- TAB(255) is the same as TAB(63).

Let's experiment. Use this short program on your TRS-80 to learn more about TAB.

```
10 CLS
20 INPUT "HOW MANY SPACES FOR TAB(0 TO 255)"; S
30 CLS
40 PRINT TAB(S) "*";
50 FOR Z = 1 TO 1000: NEXT Z
60 GOTO 10
```

14. Another way to position things on the screen is to use the PRINT @ statement, which we introduced in Chapter 4 (frames 14 and 15). Remember, the screen has 1,024 printing positions numbered from 0 (top left) to 1,023 (bottom right). These positions are arranged in 16 lines, each with 64 printing positions, as follows:

| <i>Line</i> | <i>Positions</i> |
|-------------|------------------|
| 1 | 0-63 |
| 2 | 64-127 |
| 3 | 128-191 |
| 4 | 192-255 |
| 5 | 256-319 |
| 6 | 320-383 |
| 7 | 384-447 |
| 8 | 448-511 |
| 9 | 512-575 |
| 10 | 576-639 |
| 11 | 640-703 |
| 12 | 704-767 |
| 13 | 768-831 |
| 14 | 832-895 |
| 15 | 896-959 |
| 16 | 960-1023 |

The program shown here will print an X on the first line, and the second line of the display tells where the X is located.

```
100 CLS
110 PRINT @31,"X"
120 PRINT @64,"X IS NEAR THE CENTER OF THE FIRST LINE"
```

Complete this program so that an X will be placed near the center of line 8.

```
100 CLS
110 _____
_____
```

110 PRINT @479,"X" (The number should be near 448 + 31 or 479.)

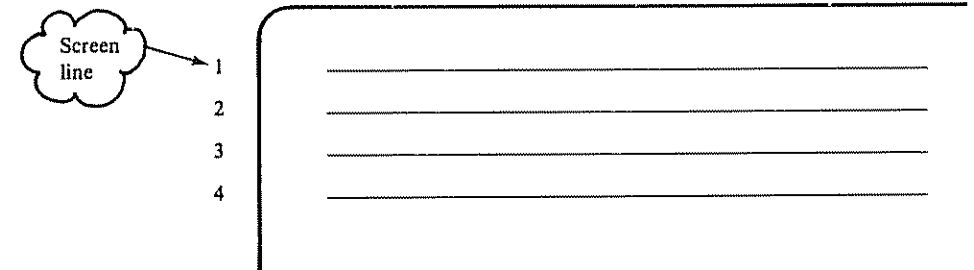
15. The value used following the PRINT@ can be used to "position" printed information anywhere on the screen. The value can even be an arithmetic expression such as X*55, Y + 32, etc.

```

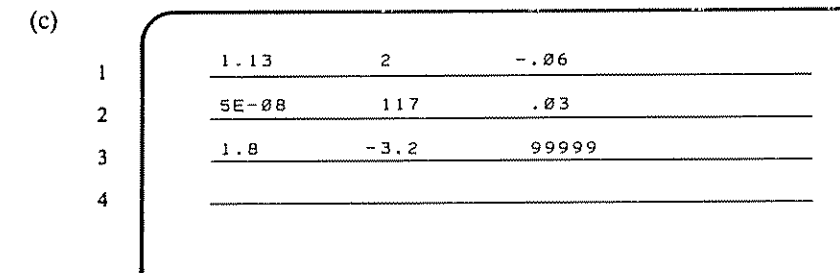
100 CLS
110 FOR X = 1 TO 3
120 READ A,B,C
130 PRINT @X*64, A,B,C
140 NEXT X
150 DATA 1.13,2,-.06,5E-08,117,.03,1.8,-3.2,99999
    
```

Line 130 tells where the information to be displayed will be shown on the screen. The commas between A, B, C provide for the spacing between values.

- (a) When the FOR-NEXT loop is executed for the first time, what is the value of X*64? _____
- (b) When the FOR-NEXT loop is executed the second time, will the values from the data list be printed on the same line? _____
- (c) Show approximately where the data will appear on the screen as the entire program is executed:



(a) 64; (b) no, the next line down



16. DATA statements may contain strings, and READ statements will read them providing the correct format is used. The variable in the READ must contain the usual \$ sign, and the data strings must be in quotes if they contain commas, colons, or leading blank spaces.

```
100 CLS
110 FOR X = 1 TO 4
120 READ A$
130 PRINT A$;
140 NEXT X
150 DATA "READ ", "DATA ", "AND ", "PRINT"
```

Notice that blank spaces are provided within the quotes of the data items for proper spacing in the PRINT statement.

Complete the following:

(a) String data items in a DATA statement sometimes must be in

(b) String variables in a READ statement are followed by what symbol?

(a) quotation marks; (b) \$ (dollar sign)

17. Trailing spaces in DATA statements will cause a blank space to be printed for all except the last item on the line. Change line 150 of the program of frame 16 to:

```
150 DATA READ ,DATA ,AND ,PRINT
```

Do the blank spaces show up when run? _____

Yes. Trailing spaces work OK, but leading spaces do *not* unless enclosed in quotation marks.

18. The spaces provided in the DATA items in frame 16 for the print format can be done in the PRINT statement itself. Then quotation marks and spaces in the DATA statement may be omitted. Alter the program in frame 16 as shown in the two following lines. Then run the program again.

```
130 PRINT A$;" ";
150 DATA READ,DATA,AND,PRINT
```

Did the printed results look the same as frame 17? _____

Yes

19. More than one READ statement may be used in the same program. All of these READ statements use the *same* data list. An item of data is assigned to a READ variable in the order that the computer executes the READ statements.

```

100 CLS
110 FOR X = 1 TO 3
120 READ A,B
130 C = A + B
140 READ D
150 E = C*D
160 PRINT C,E
170 NEXT X
180 DATA 1,2,3,4,5,6,7,8,9

```

Show the printed output:

```

_____
_____
_____
-----

```

Printed output:

| | |
|----|-----|
| 3 | 9 |
| 9 | 54 |
| 15 | 135 |

20. DATA and READ statements may use mixed numeric and string data. You should experiment to find out when quotation marks are needed when using data strings. The following program points this out:

```

100 CLS
110 READ A$
120 PRINT A$
130 READ B$,C,D$,E$
140 PRINT B$;C;D$;E$
150 DATA JOHN IS 8 YEARS OLD.,IN,2,YEARS HE WILL BE
160 DATA 10.

```

Data strings may spill over to a second line. Be sure to include the word "DATA" at the start of the second line.

(a) Show the display when the above program is executed.

```

_____
_____

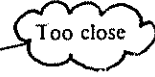
```


(b) Change line 160 (using quotation marks) so that the correct spacing will be used in the printout.

```

160 DATA _____

```

(a) JOHN IS 8 YEARS OLD.
 IN 2 YEARS HE WILL BE 10. 

(b) 160 DATA " 10."


21. Using the program in frame 20, show what each variable contains after the program is RUN.

- (a) A\$ = _____
- (b) B\$ = _____
- (c) D\$ = _____
- (d) E\$ = _____
- (e) C = _____

- (a) A\$ = JOHN IS 8 YEARS OLD.
- (b) B\$ = IN
- (c) D\$ = YEARS HE WILL BE
- (d) E\$ = 10. OR " 10."
- (e) C = 2

22. Study this program. Then answer the questions below. Finally, RUN the program to verify your answers before looking at ours.

```

100 CLS
110 FOR X = 1 TO 6
120   READ A,B
130   PRINT A;" +5 = " ; INPUT C
140   IF C = B PRINT "CORRECT" ; GOTO 180
150   IF C <> B PRINT "TRY AGAIN"
160   FOR W = 1 TO 500 : NEXT W
170   CLS : GOTO 130
180   FOR W = 1 TO 500 : NEXT W
190   CLS
200 NEXT X
210 GOTO 110
220 DATA 8,13,12,17,23,28,27,32,119,124,225,230
    
```

At line 210, the program attempts to repeat.

- (a) How many different problems are printed? _____
- (b) Why wouldn't the program repeat when line 210 was executed?

- (a) 6
 (b) The data had all been used—one pair of values for each time through the loop. The computer then displayed “?OD ERROR IN 120.

23. After all the data items of a list have been used, the data pointer may be moved to the beginning of the list by using the RESTORE statement. This allows you to use all the data in the list again. Each time a program executes a RESTORE statement, the pointer moves back to the beginning of a list.

Add this line to the program in frame 17:

```
205 RESTORE
```

Now run the program again.

- (a) Are the problems repeated? _____
 (b) How many problems are given each time through the FOR-NEXT loop (lines 110-200)? _____
 (c) When will the program stop? _____

- (a) yes
 (b) 6
 (c) Only when you press the **BREAK** key or when you turn the computer off.

24. Study the program below. Then show the output that will be displayed.

```
100 CLS
110 FOR X = 1 TO 4
120   READ A
130   PRINT A;
140   RESTORE
150 NEXT X
160 DATA 11,22,33,44
```

 11 11 11 11

(The data pointer is moved to the beginning of the line each time a data item is printed. Hence, only the first item is displayed.)

25. It's time for Draco and the dragon constellation again. We lit Draco's stars in frame 9. Here's the old program with the RESTORE statement and a few other additions:

```

100 REM***STARFALL - ORIGINALLY BY FIREDRAKE
110 CLS

200 REM***DRACO MODIFIES AGAIN
210 FOR X = 1 TO 14
220   READ OVER,DOWN
230   SET(OVER,DOWN)
240 NEXT X
250 PRINT @354,"DRACO";
260 FOR W = 1 TO 200 : NEXT W

300 REM***RESTORE DATA AND RESET STARS
310 RESTORE
320 FOR X = 1 TO 14
330   READ OVER,DOWN
340   RESET(OVER,DOWN)
350 NEXT X
360 FOR W = 1 TO 10 : NEXT W

400 REM***RESTORE AND START OVER
410 RESTORE
420 GOTO 210

500 REM***DATA LIST FOR STARS
510 DATA 27,22,29,24,27,25,24,23,40,16,44,15,45,19
520 DATA 39,23,39,26,39,29,41,31,53,30,64,28,69,29
    
```

(a) Describe how the display resulting from a run of this program differs from the display in frame 9. _____

(b) How many times will the data list be read *before* line 360 is executed?

- (a) The stars displayed in this program blink on and off. In frame 9, the stars stayed on.
- (b) twice (once to SET the points and once to RESET the points)

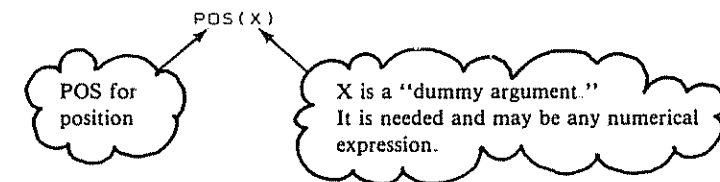
26. Instead of resetting each individual star, we could have made the program in frame 25 much shorter by taking advantage of the CLS statement in line 110. All of lines 320-420 could have been eliminated by a simple GOTO 110 statement. Rewrite the program so that this is done.

This is how we did it.

```
100 REM***STARFALL
110 CLS : FOR W = 1 TO 10

200 REM***DRACO MODIFIES AGAIN
210 FOR X = 1 TO 14
220   READ OVER,DOWN
230   SET(OVER,DOWN)
240 NEXT X
250 PRINT @354,"DRACO";
260 FOR W = 1 TO 200 : NEXT W
270 RESTORE : GOTO 110
```

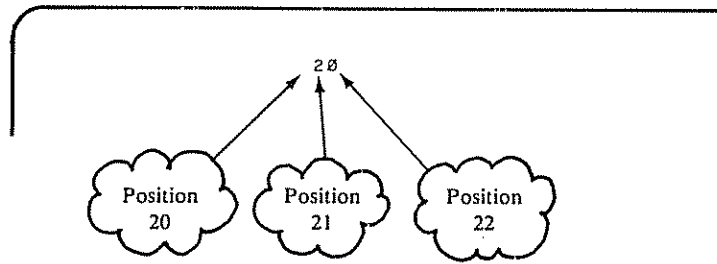
27. One more PRINT feature should be considered. It reveals where the cursor is located on a given line. Remember, the positions on a line are numbered from 0 through 63. The format for our new position locator is:



POS(X) can be used effectively with the TAB function. Take a look at these two lines:

```
100 CLS
110 PRINT TAB(20)POS(0)
```

When these two lines are executed, the number 20 will be printed at screen position 20. (Actually the 2 would be printed at position 21 and the zero at position 22 due to the leading space inserted to accommodate the sign of the number.)



If the statements below were executed, _____ would be printed at position _____.

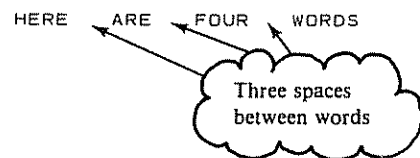
```
100 CLS
110 PRINT TAB(52)POS(0)
```

52 would be printed at position 52 (or 53 and 54 if you want to be technical)

28. POS(X) can be used to provide spacing as desired on a given line. Suppose you want to space four words so that there were three blanks between the end of each word and the start of the next word.

```
100 PRINT "HERE" TAB(POS(0)+3) "ARE" TAB(POS(0)+3);
110 PRINT "FOUR" TAB(POS(0)+3) "WORDS"
```

If those two lines are executed, you will see:



- (a) Alter line 100 so that there will be two spaces between the words "HERE" and "ARE."

```
100 _____
```

- (b) Alter line 110 so that there will be four spaces between the words "FOUR" and "WORDS."

```
110 _____
```

- (c) Show the placement of the words on the screen using lines 100 and 110 from (a) and (b).

```
-----
0 1 2 3 4 5 6                               24
-----
```


- (a) On which line does the dog appear? _____
 (b) For what values of N will the dog move on to the second line?

(a) first; (b) 9 and 10

32. Now for the GRAND FINALE! It's time for the annual "Frog Jumping Contest." Your frog will make giant leaps according to your input, J. The program counts the number of jumps for the frog to get off the screen. It's a wet day—so the frog leaves his imprint in the mud.

```
100 CLS
110 AS = "FRDG"
120 INPUT "JUMP = " ; J
130 P = 3 ; C = 0
140 CLS ; PRINT AS;
150 FOR X = 1 TO 500 : NEXT X
160 IF P >= 60 GOTO 210
170 PRINT TAB(POS(2)+J) AS;
180 FOR X = 1 TO 200 : NEXT X
190 P = P + 4 + J ; C = C + 1
200 GOTO 160
210 FOR X = 1 TO 300 : NEXT X
220 CLS ; PRINT "THE FROG JUMPED OFF THE SCREEN IN";C;"JUMPS"
230 PRINT "HIS LEAP WAS";J;"SPACES"
```

- (a) For each run that you make, the space between each frog imprint will depend on what? _____
 (b) Complete this table showing how many jumps it takes for the frog to clear the screen:

| <i>J Input</i> | <i>Total Jumps</i> |
|----------------|--------------------|
| 2 | _____ |
| 4 | _____ |
| 8 | _____ |
| 10 | _____ |
| 20 | _____ |
| 35 | _____ |

- (c) Change line 120 so that a random-length jump will be made.

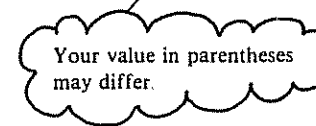
120 _____

(a) the value of J, your input

(b) *J Input* *Total Jumps*

| | |
|----|----|
| 2 | 11 |
| 4 | 8 |
| 8 | 6 |
| 10 | 5 |
| 20 | 3 |
| 35 | 2 |

(c) 120 J = RND(57)



Jump next to the Self-Test that follows, so you can test your knowledge of how to enter and display data.

SELF-TEST

Here is another chance to show how much you have learned.

1. Whenever a READ statement is used, there must be a corresponding _____ statement.

2. The following program is executed:

```
100 READ A
110 PRINT A
120 GOTO 100
130 DATA 55,122,66,144
```

What is printed after line 120 is executed for the fourth time?

3. Find errors in statement 130. Assume the program of test question 2 is being run with this modification:

```
130 DATA 55;103,A+5,99
```

Errors: _____

4. Complete line 110 so that a flag will avoid an out-of-data error. Six items are to be printed.

```
100 READ C
110 IF _____
120 PRINT C;
130 GOTO 100
140 DATA 11,22,33,44,66,77,-9999
150 END
```

5. Complete the FOR-NEXT statements so that this program will read and print all data provided.

```
100 FOR X = _____
110 READ A,B
120 PRINT A;B
130 _____
140 DATA 1,2,3,4,5,6,7,8
```

6. Write a modification of line 120 in the program of question 5 using the TAB function. A is to be printed 8 spaces in from the left margin. B is to be printed 16 spaces in.

120 _____

7. Print positions used with the TAB function may be integers from _____ through _____.

8. The value or expression following PRINT@ may be from _____ through _____.

9. If the information in line 150 below is to be read from a DATA statement, what kind of variable must be used in the corresponding READ statement?

```
150 DATA "READ", "THIS LINE"
```

10. If two READ statements are used in a program, may the same data list be used for both READ statements? _____

11. What happens to the data pointer when all data items of a list have been used and a RESTORE statement is executed?

12. This program is illegal:

```
100 CLS
110 A = 5
120 PRINT TAB(POS(A)+8)
```

- (a) Which line should be deleted? _____
 (b) Rewrite the line that should be changed.

13. Complete these lines so that 4 spaces would be provided *between* the words:

```
100 PRINT "FOUR" TAB(      ) "SPACES";
110 PRINT TAB(      ) "BETWEEN";
150 PRINT TAB(      ) "WORDS"
```

14. Show the screen display for this program:

```
100 CLS
110 PRINT @64, "START" TAB(POS(3)+4) "END"
```

1st line

2nd line

3rd line

Answers to Self-Test

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

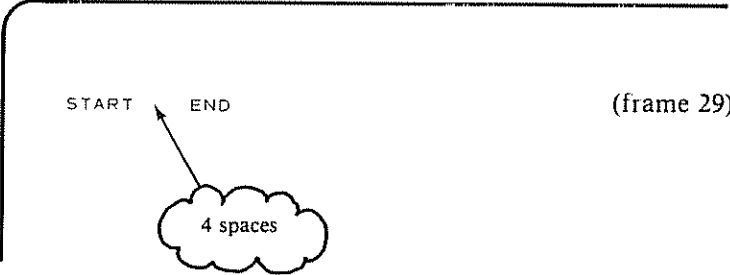
1. DATA statement (frame 1)
2. ?OD ERROR IN 100 (frame 1)
3. Errors: semicolon after 55
A + 5 (no variables or arithmetic operators) (frames 2, 3)
4. 110 IF C = -9999 THEN 150 OR
110 IF C = -9999 GOTO 150 OR other forms of the same (frame 5)
5. 100 FOR X = 1 TO 4
130 NEXT X (frame 9)
6. 120 PRINT TAB(8) A; TAB(16) B (frame 12)
7. from 0 through 63 (frame 13)
8. from 0 through 1023 (frame 14)
9. string variable (frame 16)
10. yes (frame 19)
11. the pointer moves back to the beginning of the DATA list (frame 23)
12. (a) delete line 110
(b) 120 PRINT TAB(POS(5)+8) (frames 27, 31)

```
13. 100 PRINT "FOUR" TAB(POS(1)+4) "SPACES";  
    110 PRINT TAB(POS(1)+4) "BETWEEN";  
    120 PRINT TAB(POS(1)+4) "WORDS"
```

Any numerical
expression is OK.

(frames 27, 28, 31)

14. 1st line
 2nd line
 3rd line



(frame 29)

CHAPTER EIGHT

Strings

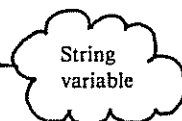
You have used strings in earlier chapters in INPUT and PRINT statements such as:

```
40 INPUT "WHAT DIAMETER IS THE WHEEL"; D
50 C = 3.14159*D
60 PRINT "THE CIRCUMFERENCE OF THE WHEEL IS"; C
```



OR

```
20 INPUT "WHAT IS YOUR NAME"; NS
30 INPUT "HELLO "; NS
```



String comparisons have also been used. For example:

```
250 IF AS = "YES" THEN GOTO 100
```

In this chapter, we will attempt to untie some of the "knotty" string experiences that will face you in the future.

After completing this chapter, you will be able to:

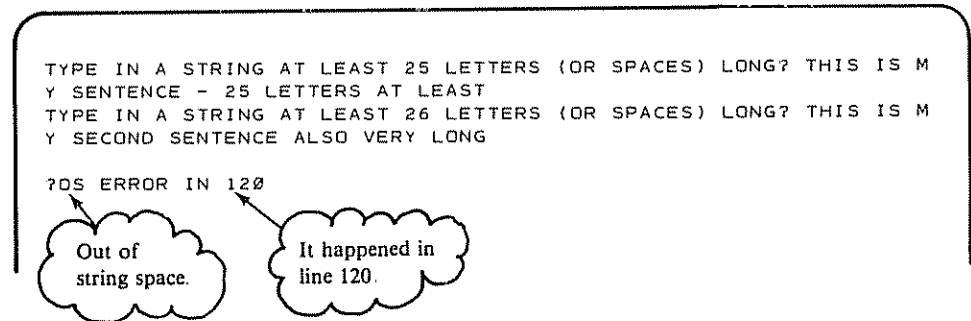
- set aside memory space for strings;
- concatenate (or join) strings together;
- have the computer calculate the length of strings;
- find a substring contained in a string;
- compare strings for precedence (which one comes first alphabetically);
- display ASCII codes for keyboard characters;
- use inequalities with strings;
- display keyboard characters from ASCII codes.

1. When the TRS-80 is turned on, 50 characters of memory are automatically set aside for string space. If you are using several strings, or a few long strings,

50 characters may not be enough. When you try to run your program, you may get an OS ERROR IN LINE XX (Out of String Space). For example:

```
100 CLS
110 INPUT "TYPE IN A STRING AT LEAST 25 LETTERS (OR SPACES)
    LONG"; A$
120 INPUT "TYPE IN A STRING AT LEAST 26 LETTERS (OR SPACES)
    LONG"; B$
130 PRINT A$
140 PRINT B$
```

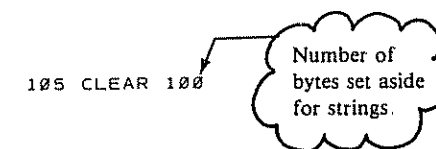
This is our run. Notice how the word "MY" is broken at the end of the line:



- (a) How many characters of string space are available when the TRS-80 is turned on? _____
- (b) I counted 41 characters in the A\$ input. How many characters can we have in B\$ without getting an OS ERROR? _____

(a) 50; (b) 9 (A\$ used up 41 characters out of 50.)

2. To correct the situation in frame 1, we can save more string space in memory by using the CLEAR N statement (see Chapter 3, frame 20). The total number of string characters needed (one for each character and blank space) is used in place of N. It is better to save too many than too few. But when you see the OS ERROR in the future, you will know what to do. For example:



- (a) If we add the above statement to the program in frame 1, will we get an OS ERROR when we run it? _____

- (b) Complete the screen display below to show the results of a run with line 105 added.

```

TYPE IN A STRING AT LEAST 25 LETTERS (OR SPACES) LONG? THIS IS M
Y SENTENCE - 25 LETTERS AT LEAST
TYPE IN A STRING AT LEAST 26 LETTERS (OR SPACES) LONG? THIS IS M
Y SECOND SENTENCE ALSO VERY LONG
_____
_____

```

- (a) no
(b)

```

TYPE IN A STRING AT LEAST 25 LETTERS (OR SPACES) LONG? THIS IS M
Y SENTENCE - 25 LETTERS AT LEAST
TYPE IN A STRING AT LEAST 26 LETTERS (OR SPACES) LONG? THIS IS M
Y SECOND SENTENCE ALSO VERY LONG
THIS IS MY SENTENCE - 25 LETTERS AT LEAST
THIS IS MY SECOND SENTENCE ALSO VERY LONG

```

Now it is OK—
no OS ERROR.

3. There is an operation that can be performed on strings called *catenation*. When you concatenate two or more strings, you join them together (or unite them). The operator for joining strings together is the + sign. In arithmetic the + sign is used for adding two quantities, but when used with two strings it joins them together. For example:

```

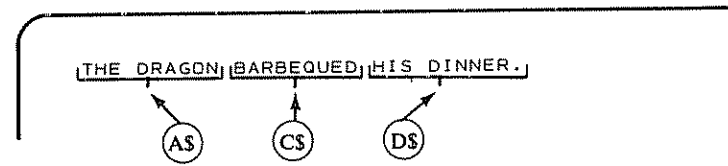
10 CLS
20 CLEAR 100
30 A$ = "THE DRAGON"
40 B$ = " SLEW "
50 C$ = " BARBEQUED"
60 D$ = " HIS DINNER."
70 E$ = "THE BLACK KNIGHT"
80 PRINT A$ + C$ + D$

```

Be sure to include spaces.

Join them together.

When run:

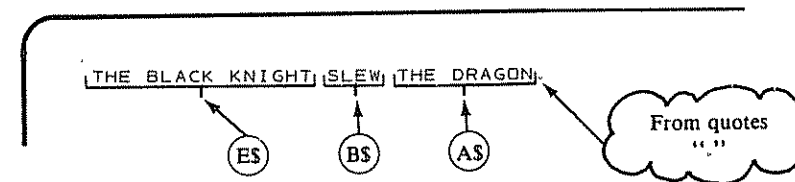


Now change line 70 to:

```
70 PRINT E$ + B$ + A$ + "."
```

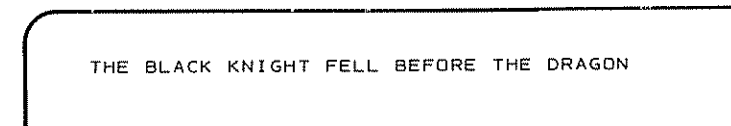
This joins a period to the rest.

When run:



Of course, there are many combinations that you could make by concatenating A\$, B\$, C\$, D\$, E\$ + anything else in "quotes."

Write a PRINT statement to display:



```
80 PRINT
```

```
80 PRINT E$+" FELL BEFORE "+A$
```

Spaces

4. You may also enter strings and then use them in the catenation operation.
For example:

```
100 CLS
110 CLEAR 200
120 INPUT "TYPE IN AN ADJECTIVE"; A$
130 INPUT "TYPE IN A VERB (PAST TENSE)"; B$
140 C$ = "THE WHITE KNIGHT "
150 D$ = " DAMSEL NAMED KIT."
160 PRINT
170 PRINT C$ + B$ + " THE " + A$ + D$
```

Notice increased
string space reserved.

The run:

```
TYPE IN AN ADJECTIVE? BEAUTIFUL
TYPE IN A VERB (PAST TENSE)? KISSED

THE WHITE KNIGHT KISSED THE BEAUTIFUL DAMSEL NAMED KIT.
```

(a) How many bytes of memory does the program set aside for string space?

(b) Would 100 bytes have been enough? _____

(a) 200

(b) It depends on the length of the A\$ and B\$ strings that *you* type in.

5. A string variable may be assigned to the unification (catenation) of other strings. For example:

```
T$ = A$ + B$
```

(a) Rewrite line 170 of the program in frame 4 to assign the catenation in the PRINT statement to the variable T\$.

```
170 _____
```

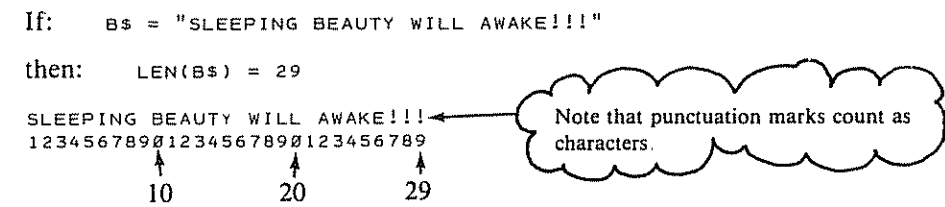
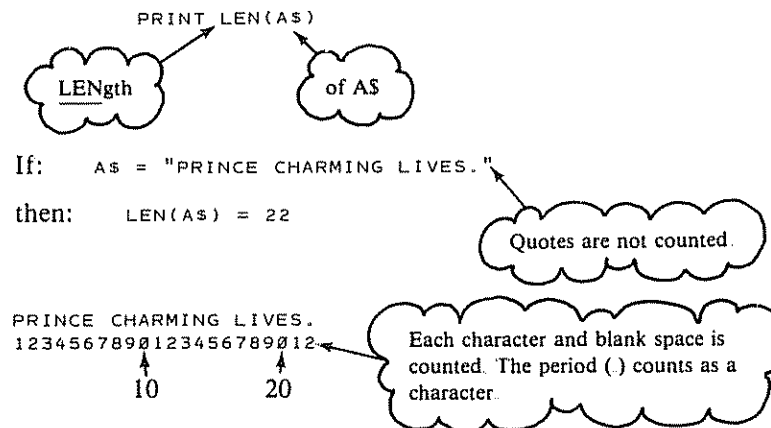
(b) Write a new line 180 to print T\$ on the eighth line of the screen.

```
180 _____
```

(a) 170 T\$ = C\$ + B\$ + " THE " + A\$ + D\$

(b) 180 PRINT @448, T\$

6. We can determine the length of any string with a PRINT statement such as:



Now it's your turn. Give the length of each.

- A\$ = "HAPPY HOLIDAYS!" LEN(A\$) = _____
- B\$ = "HOW LONG AM I?" LEN(B\$) = _____
- C\$ = "" LEN(C\$) = _____
- D\$ = "FUN, ISN'T IT? YES?" LEN(D\$) = _____
- E\$ = "EIGHT" LEN(E\$) = _____
- F\$ = "FOUR" LEN(F\$) = _____

```

LEN(A$) = 15
LEN(B$) = 14
LEN(C$) = 0
LEN(D$) = 20
LEN(E$) = 5
LEN(F$) = 4
('' is an empty string, with no
characters.)
    
```

7. We could use this feature to determine the length of each of the strings that we used in the program of frame 4 (as modified in frame 5). We would add these lines to the program.

```

200 PRINT LEN(A$);LEN(B$);LEN(C$);LEN(D$);LEN(" THE ")
210 PRINT LEN(T$)
    
```

Give the values that you would expect to be printed.

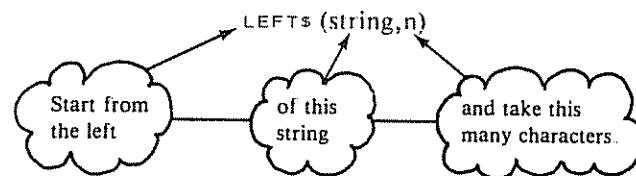
```
LEN(A$) = _____
LEN(B$) = _____
LEN(C$) = _____
LEN(D$) = _____
LEN(" THE ") = _____
LEN(T$) = _____
```

```
-----
LEN(A$) = 9
LEN(B$) = 6
LEN(C$) = 17
LEN(D$) = 18
LEN(" THE ") = 5
LEN(T$) = 55 (This is the sum of the lengths of the strings which are
               catenated to form T$.)
```

8. A string has a beginning and an end. We know now that a string also has length. Since we normally read from left to right, the beginning of a string naturally starts on the left.

```
A$ = "THE LEFT OF A STRING IS ITS BEGINNING.."
```

The string features of the TRS-80 are many and magnificent. If you don't want to use all of A\$, there is a statement that will let you use only the left part of it. You can take as much as you want.



If we enter this:

```
100 CLS
110 A$ = "THE LEFT OF A STRING IS ITS BEGINNING.."
120 PRINT LEFT$(A$,8)
RUN_
```


When we press the **ENTER** key we see:

```
THE LEFT
```

- (a) If line 120 said: 120 PRINT LEFT\$(A\$,11), what would the screen show when run?

- (b) If 120 said: 120 PRINT LEFT\$(A\$,20), what would the screen show?

- (a)

```
THE LEFT OF
```

- (b)

```
THE LEFT OF A STRING
```

9. We can assign a part of a string to another string variable—such as: B\$ = LEFT\$(A\$,8). The following program defines several such “substrings” of A\$:

```
100 CLS : CLEAR 150
110 A$ = "THE LEFT OF A STRING IS ITS BEGINNING."
120 B$ = LEFT$(A$,8)
130 C$ = LEFT$(A$,11)
140 D$ = LEFT$(A$,20)
150 E$ = LEFT$(A$,23)
160 F$ = LEFT$(A$,30)
170 PRINT B$ : PRINT C$ : PRINT D$ : PRINT E$ : PRINT F$
```

Show how the display will look after running the program.

```
THE LEFT
THE LEFT OF
THE LEFT OF A STRING
THE LEFT OF A STRING IS
THE LEFT OF A STRING IS ITS BEGINNING.
```

10. In Chapter 7, frame 25, the constellation Draco was shown in the sky. It turned on and off but not very realistically. The program below randomly blinks the stars in the constellation. Note the use of LEFT\$ in line 520.

```
100 REM***DRACO BLINKS
110 CLS
120 RESTORE

210 FOR X = 1 TO 14
220 READ OVER,DOWN
230 SET(OVER,DOWN)
240 NEXT X
250 PRINT @354, "DRACO BLINKS"
260 FOR W = 1 TO 200 : NEXT W

300 FOR N = 1 TO 10
310 RESTORE
320 FOR X = 1 TO 14
330 READ OVER,DOWN
340 Y = RND(2)
350 IF Y = 1 THEN RESET(OVER,DOWN)
360 FOR W = 1 TO 10 : NEXT W
370 SET(OVER,DOWN)
380 FOR W = 1 TO 10 : NEXT W
390 NEXT X
400 NEXT N

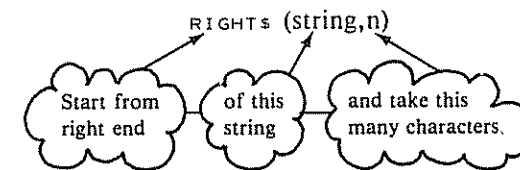
500 CLS
510 INPUT "DO YOU WANT TO WATCH SOME MORE (YES OR NO)"; AS
520 IF LEFT$(AS,1) = "Y" GOTO 110
530 PRINT "THANK YOU FOR WATCHING"
540 PRINT : PRINT "GOODBYE" : PRINT

600 DATA 27,22,29,24,27,25,24,23,40,16,44,15,45,19
610 DATA 39,23,39,26,39,29,41,31,53,30,64,28,69,29
```

- (a) A player types just the letter Y and presses **ENTER** when the program reaches line 510. When line 520 is executed, will the program go to line 110 or line 530? _____
- (b) In line 520, only the first letter of the input is checked. Change line 520 so that it looks at the first three letters YES. _____
- (c) With the change to line 520, if a player types just the letter Y and presses **ENTER**, will the program now go to line 110 or to line 530? _____

- (a) 110
 (b) 130 IF LEFT\$(A\$,3) = "YES" GOTO 110
 (c) 530

11. Just as a string has a left beginning, so it must have a right ending. As you would suspect, the TRS-80 can also take substrings off the right end of a string. You probably have already guessed what the statement is.



Let:

Z\$ = "THE END OF A STRING IS ON THE RIGHT."

If we enter this program:

```
100 CLS
110 Z$ = "THE END OF A STRING IS ON THE RIGHT."
120 PRINT RIGHT$(Z$,6)
RUN_
```

when we press the **ENTER** key, we see:

```
RIGHT.
```

(a) Change line 120 so that the screen will show:

```

ON THE RIGHT.
120 _____
    
```

(b) Change line 120 so that the screen will show:

```

A STRING IS ON THE RIGHT.
120 _____
    
```

(a) 120 PRINT RIGHT\$(Z\$,13)

(b) 120 PRINT RIGHT\$(Z\$,25)

12. Complete the following program, which is similar to that of frame 11. Use string variables for substrings to produce this display from \$Z = "THE END OF A STRING IS ON THE RIGHT."

```

THE RIGHT.
RING ON THE RIGHT.
END OF A STRING IS ON THE RIGHT.
    
```

The program:

```

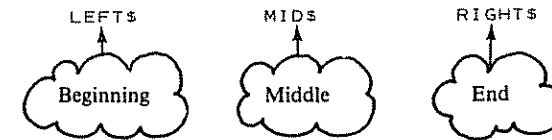
100 CLS : CLEAR 150
110 Z$ = "THE END OF A STRING IS ON THE RIGHT."
120 A$ = _____
130 B$ = _____
140 C$ = _____
150 _____
    
```

A MULTIPLE
statement line

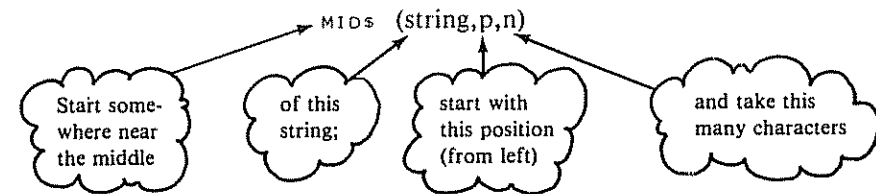
```

120 A$ = RIGHTS(Z$,10)
130 B$ = RIGHTS(Z$,21)
140 C$ = RIGHTS(Z$,32)
150 PRINT A$ : PRINT B$ : PRINT C$
    
```

13. Now we come to the most magnificent string feature of all, the MID\$ function. Strings have a beginning on the left, and an end on the right. And they have a middle that joins the beginning and the end.



To pick a substring out of the middle of a string, we must decide where to start and how many characters to take. That's just what the MID\$ function provides.



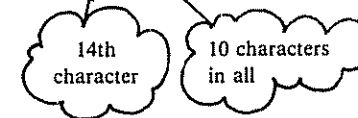
Let's see how it works.

```

Z$ = "THE HEART IS THE MIDDLE OF A STRING."
    123456789012345678901234567890123456
    (10)  (20)  (30)
    
```

```

MID$(Z$,14,10) = THE MIDDLE
    
```



Show these substrings of Z\$:

(a) MID\$(Z\$,5,19) = _____

(b) MID\$(Z\$,18,18) = _____

(a) MID\$(Z\$,5,19) = HEART IS THE MIDDLE



(b) MID\$(Z\$,18,18) = MIDDLE OF A STRING

14. Here is a program that uses Z\$ but completely scrambles the sentence:

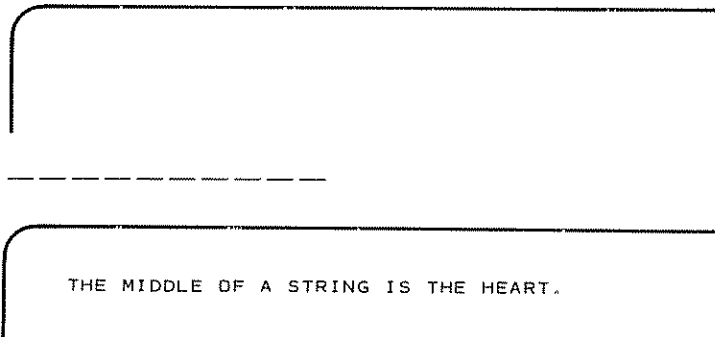
```

100 CLS
110 Z$ = "THE HEART IS THE MIDDLE OF A STRING."
120 A$ = MID$(Z$,14,11)
130 B$ = MID$(Z$,25,11)
140 C$ = MID$(Z$,10,4)
150 D$ = MID$(Z$,1,9)
160 E$ = MID$(Z$,36,1)
170 PRINT A$ + B$ + C$ + D$ + E$
    
```

Pick Z\$ apart.

Join the pieces together again—but in a different order.

Show the display when the program is run.



15. Sometimes the same result can be achieved by using either MID\$ or LEFT\$. Sometimes the same result can be achieved by using either MID\$ or RIGHT\$. Study lines 150 and 160 of the program in frame 14.

Replace each line using either LEFT\$ or RIGHT\$.

(a) 150 _____

(b) 160 _____

(a) 150 D\$ = LEFT\$(Z\$,9)

(b) 160 E\$ = RIGHT\$(Z\$,1)

16. Earlier in the book you made string comparisons to see whether or not two strings were equal. For example:

```

30 IF A$ = "YES" THEN GOTO 100
    
```

Strings are compared by checking each character from left to right. For example:

A\$ = "CAB" and B\$ = "CAN"

1. C EQUALS C
2. A EQUALS A
- but 3. B DOES NOT EQUAL N

Therefore A\$ is not equal to B\$.

Deep down inside the computer, characters are stored (in memory) as ASCII codes (ASCII means *American Standard Code for Information Interchange*). The ASCII code for A is 65, the ASCII code for B is 66, and the ASCII code for C is 67.

(a) Guess the ASCII code for the letter D. _____

(b) Guess the ASCII code for the letter Z. _____

(a) 68; (b) 90

ASCII codes for letters run from 65 (A) to 90 (Z).

17. We can use the ASC function to get the ASCII code for any keyboard character.

We type: PRINT ASC("A")

It prints: 65

We type: PRINT ASC("B")

It prints: 66

We type: PRINT ASC("Z")

It prints: 90

We type: PRINT ASC("*")

It prints: 42

Appendix G is a list of ASCII codes for all characters used in TRS-80 BASIC. Use Appendix G to help you complete the following:

(a) You type: PRINT ASC("C") (b) You type: PRINT ASC("+")

It prints: _____

It prints: _____

(c) You type: PRINT ASC(";")

(d) You type: PRINT ASC(" ")

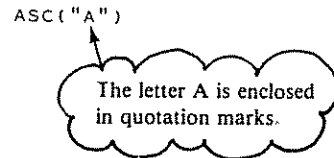
It prints: _____

It prints: _____

Space

- (a) 67
- (b) 43
- (c) 59
- (d) 32 (Yes, even a space has an ASCII code!)

18. The `ASC` function delivers the ASCII code of a keyboard character, enclosed in quotation marks.



We can also use `ASC` to compute the ASCII code for the value of a string variable.

We type: `z$ = "A"`

We type: `PRINT ASC(z$)`

It prints: 65

We type: `x$ = "M"`

We type: `PRINT ASC(x$)`

It prints: 77

Your turn. Complete the following:

- (a) You type: `b$ = "+"`
You type: `PRINT ASC(b$)`
It prints: _____
- (b) You type: `k$ = "Z"`
You type: `PRINT k$`
It prints: _____
- (c) This one is tricky. Go ahead—guess!
You type: `n$ = "ABC"`
You type: `PRINT n$`
It prints: _____

- (a) 43
- (b) 90
- (c) 65 (The `ASC` function computes the ASCII code of the *first* character in the value of `n$`.)

19. It's play time now. How about a game of "Guess My Letter"? In this game the computer will think of a letter from A to Z. You will then try to guess what

letter it's thinking of. If you guess wrong, it will give you a clue. Here is the first part of the program:

```
100 REM***GUESS MY LETTER GAME
110 REM***EXPLAIN GAME TO PLAYER
120 CLS
130 PRINT "I'M THINKING A LETTER FROM A TO Z."
140 PRINT "GUESS MY LETTER!!!"

200 REM***X = ASCII CODE FOR SECRET LETTER
210 X = RND(26) + 64
```

Your turn—answer these questions:

- (a) What values are possible for RND(26)? _____
- (b) When 64 is added to your answers in part (a), you have the range of values for X. What is this range? _____ to _____
- (c) What is the possible range of values for the ASCII codes of the alphabetic characters? _____ to _____

-
- (a) integers from 1 through 26 inclusive
- (b) integers from 65 through 90 inclusive
- (c) also integers 65 through 90 inclusive

Aha! RND(26) + 64 is an integer in the range 65 to 90. Therefore, it is the ASCII code for a letter in the range A to Z.

20. Here is the next piece of the program:

```
300 REM***GET GUESS (G$). IS IT A LETTER?
310 PRINT : INPUT "YOUR GUESS (A TO Z)"; G$
320 IF ASC(G$) < 65 THEN PRINT "GUESS A LETTER!" : GOTO 310
330 IF ASC(G$) > 90 THEN PRINT "GUESS A LETTER!" : GOTO 310
```

Lines 320 and 330 compare the ASCII code of the guess (G\$) with the ASCII codes for A and Z.

- (a) If the code for G\$ is less than the code for A or more than the code for Z, what happens? _____
- _____
- (b) If the code for G\$ is between 65 and 90, inclusive, what happens? _____
- _____

-
- (a) The computer prints GUESS A LETTER! then goes to line 310 and asks for another guess.
- (b) The computer moves on to the next line after line 330, which you will see in the next frame.

21. Next we compare the ASCII code of the guess with the ASCII code of the secret letter.

```
400 REM***COMPARE GUESS WITH SECRET LETTER
410 IF ASC(G$) = X THEN GOSUB 510 : GOTO 120
420 IF ASC(G$) < X THEN GOSUB 610
430 IF ASC(G$) > X THEN GOSUB 710
440 GOTO 310
```

Remember, the value of X is the ASCII code of the computer's secret letter.

If the player guesses the secret letter, the condition in line 410 is *true*. The computer will go to line 510. The section of the program beginning at line 510 will tell the player he or she has guessed the secret letter.

(a) Suppose the guess is "earlier" in the alphabet than the secret letter. What happens? _____

(b) Suppose the guess is "later" in the alphabet than the secret letter. What happens? _____

(a) The condition in line 420 is *true*. The computer will call (use) the subroutine beginning in line 610.

(b) The condition in line 430 is *true*. The computer will call (use) the subroutine beginning in line 710.

22. Your patience is rewarded! Here is the rest of the program. These are the messages to the player:

```
500 REM***PLAYER HAS GUESSED THE SECRET LETTER
510 PRINT "YOU GUESSED IT!!!"
520 FOR Z = 1 TO 1000 : NEXT Z
530 RETURN

600 REM***GUESS IS 'LESS THAN' SECRET LETTER
610 PRINT "TRY CLOSER TO THE END OF THE ALPHABET"
620 RETURN

700 REM***GUESS IS 'MORE THAN' SECRET LETTER
710 PRINT "TRY CLOSER TO THE BEGINNING OF THE ALPHABET"
720 RETURN

999 END
```

We did say that it's play time. So . . . play!

A gentle warning: When you play this game, pressing certain keys may cause an FC ERROR to occur. For example, this may happen if you enter a

space without quotation marks or a left arrow (←) or a down arrow (↓). We are not going to try to explain why, but we will offer a way to fix the problem.

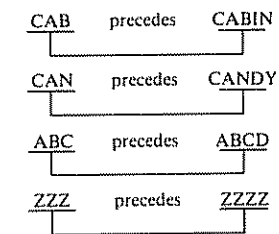
Replace line 310 with the following bunch of lines.

```
310 PRINT : PRINT "YOUR GUESS (A TO Z)?":
312 G$ = INKEY$ : IF G$ = "" THEN 312
314 PRINT G$
```

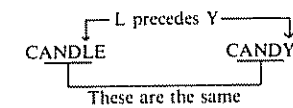
Watch for our next TRS-80 Self-Teaching Guide, in which we try to explain our way out of the above idiosyncrasy of the TRS-80!

23. When two strings are compared, the character for the lower code is considered to precede the other character. For example, CAB precedes (comes before or is "less than") CAN because the ASCII code for B is 66, which is less than the ASCII code for N, which is 78.

CAB precedes CABIN and CAN precedes CANDY. If a word is the left-most part of a longer word, then the shorter word precedes the longer word. Got that?



CANDLE, of course, precedes CANDY.



Hmmm . . . it is very much like the way things are organized in a dictionary!

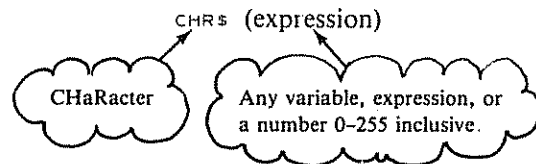
Your turn. For each pair of words or other bunches of characters, complete the sentence about precedence. If necessary, use the ASCII codes in Appendix G.

| First "Word" | Second "Word" | Precedence Sentence |
|--------------|---------------|------------------------|
| (a) A | Z | _____ precedes _____ . |
| (b) ABD | ABC | _____ precedes _____ . |
| (c) ZZZQ | ZZZPP | _____ precedes _____ . |
| (d) TRS 80 | TRS-80 | _____ precedes _____ . |

-
- (a) A; Z How about that for an easy one!
 (b) ABC; ABD Still pretty easy.

- (c) ZZZPP; ZZZQZ Did you get this one?
 (d) TRS 80; TRS-80 Ha! Bet we got you on this one. Since the ASCII code for space is 32 and the ASCII code for the hyphen is 95, TRS 80 precedes TRS-80.

24. 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, we'll let you in on another BASIC feature, the CHR\$ function.



The CHR\$ function gives the character that corresponds to the ASCII code enclosed in parentheses.

We type: PRINT CHR\$(65)

It prints: A

We type: PRINT CHR\$(92)

It prints: †

Your turn. Use Appendix G or your TRS-80 to help you complete the following:

(a) You type: PRINT CHR\$(66)

It prints: _____

(b) You type: PRINT CHR\$(35)

It prints: _____

(a) B

(b) #

25. Experiment. Use the following program to explore the characters that correspond to ASCII codes:

```

100 CLS
110 INPUT "FIRST ASCII CODE" ; FIRST
120 INPUT "LAST ASCII CODE" ; LAST

130 FOR CODE = FIRST TO LAST
140   PRINT CODE "=" CHR$(CODE),
150 NEXT CODE

160 GOTO 160
  
```

Comma

Here is what happened when we ran the program using 32 for FIRST and 63 for LAST. Compare the information on the screen with Appendix G.

```
FIRST ASCII CODE? 32
LAST ASCII CODE? 63
```

| | | | |
|---------|--------|--------|--------|
| 32 = | 33 = ! | 34 = " | 35 = # |
| 36 = \$ | 37 = % | 38 = & | 39 = ' |
| 40 = (| 41 =) | 42 = * | 43 = + |
| 44 = , | 45 = - | 46 = . | 47 = / |
| 48 = 0 | 49 = 1 | 50 = 2 | 51 = 3 |
| 52 = 4 | 53 = 5 | 54 = 6 | 55 = 7 |
| 56 = 8 | 57 = 9 | 58 = : | 59 = ; |
| 60 = < | 61 = = | 62 = ? | |

Suppose you want to run the program and show the codes and characters for the letters A through Z.

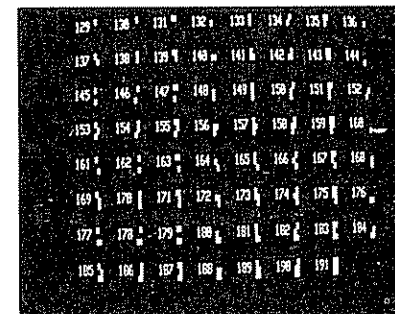
- (a) What value will you enter for FIRST? _____
 (b) What value will you enter for LAST? _____

- (a) 65
 (b) 90

26. The ASCII codes 129 to 191 represent graphics "characters." RUN the program from the preceding frame and enter 129 at the request for the first ASCII code; 191 at the request for the last code. The screen fills with "characters." Let's try another program before we attempt to explain any of this. The following program will show the graphics characters more clearly:

```
100 CLS
110 FOR CODE = 129 TO 191
120   PRINT CODE CHR$(CODE) CHR$(32) CHR$(32);
130   IF POS(123) = 0 THEN PRINT
140 NEXT CODE
150 GOTO 150
```

RUN the program on your TRS-80. The screen should look like this:



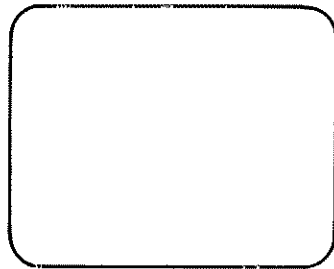
Yes, yes—we know that the program is a little mysterious. But it showed the graphics characters (didn't it??). Read on. We will try to explain the program.

27. Ha! We tricked you. Instead of explaining the program, we will do something with the graphics characters.

Try this program. (Have patience—we will explain the program soon.)

```
10 CLS
20 PRINT CHR$(32) CHR$(32) CHR$(151)
30 PRINT CHR$(143) CHR$(143) CHR$(143) CHR$(143) CHR$(131)
40 GOTO 40
```

Before you RUN this program, look at the graphics characters for codes 151, 143, and 131. Then sketch the "picture" that will appear on the screen when the above program is run.



Put your sketch on the screen in the place where it will appear.



Got the picture? (Oops, sorry.) You can use the graphics characters as building blocks to make pictures on the screen.

28. OK, OK, we will try to explain the program in frame 26. Here it is again:

```
100 CLS
110 FOR CODE = 129 TO 191
120   PRINT CODE CHR$(CODE) CHR$(32) CHR$(32);
130   IF POS(0) = 0 THEN PRINT
140 NEXT CODE
150 GOTO 150
```

We suspect that you may have some difficulty understanding lines 120 and 130. So, here we go.

```

120 PRINT CODE CHR$(CODE) CHR$(32) CHR$(32)

```

This occupies five
print positions
(space, three digits,
space).

This occupies
one print
position.

Two spaces.

Each time line 120 is executed, the information printed uses exactly eight print positions. Therefore, after line 120 has been executed eight times, the end of a line is reached as the computer moves to the beginning of a new line. Aha! Look at line 130.

```

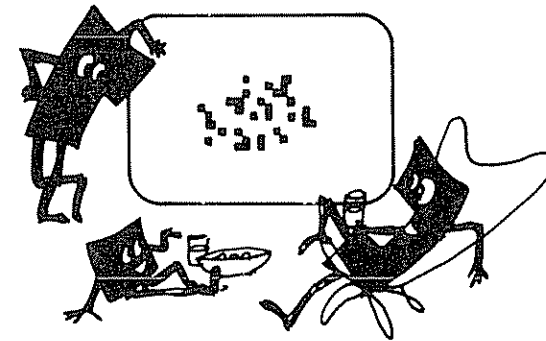
130 IF POS(0) = 0 THEN PRINT

```

If the cursor is at the left side of a new line— $POS(0) = 0$ is *true*—the TRS-80 prints an empty line. That's how we get our "double spacing."

Remember, for printing characters, the screen is 64 characters wide. So, if we print eight characters across the screen eight times, we get to the end of the screen. The computer moves on to the zero (0) position of the *next line*. Then, line 130 causes an empty line to be printed, giving us our desired double spacing.

29. Pick any graphics character. Write a program to put 100 characters on the screen in random positions. (For review, see frame 8, Chapter 5.)



We picked the character whose ASCII code is 167.

```
100 CLS
110 FOR K = 1 TO 100
120   PRINT @RND(1023), CHR$(167) ;
130 NEXT K
140 GOTO 140
```

30. Now modify your program in the previous frame so that each character is also picked at random.

Here are the changes to our program:

```
115 X = RND(63) + 128
120 PRINT @RND(1023), CHR$(X) ;
```

31. There is a nice shortcut for printing a string of repeating characters on the screen. This statement will do the trick:

`PRINT STRING$(n, character or number)`

Number of times you want the character printed, or length of the string.

Any number 0-255 or a character that will be treated as ASCII, control, or graphics code.

We type: `PRINT STRING$(5,65)`

It prints: AAAAA

We type: `PRINT STRING$(10, "*")`

It prints: *****

We type: `A$ = "+"`

We type: `PRINT STRING$(7,A$)`


It prints: ++++++

(a) The ASCII code for the letter X is 88. Write a statement using `STRING$` to print 13 X's. _____

(b) Show what this program will display when run.

```
10 CLS
20 PRINT STRING$(8,"*")
30 PRINT STRING$(2,"0") " " STRING$(2,"0")
40 PRINT STRING$(8,"*")
```

4 spaces



(a) PRINT STRING\$(13,88) OR PRINT STRING\$(13,"X")

(b) *****
00 00

32. A numeric variable may be used for the ASCII code in the STRING\$ function, provided that the value of the variable is in the ASCII code range 0 to 255.

We type: C = 90
We type: PRINT STRING\$(7,C)
It prints: ZZZZZZZ

We type: C = 256
We type: PRINT STRING\$(7,C)
It prints: ?FC ERROR

The FC ERROR occurred because the value of C was more than 255.

Complete the following:

(a) You type: C = 65
You type: PRINT STRING\$(3,C)
It prints: _____

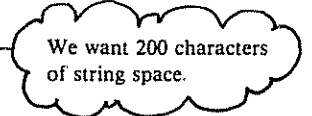
(b) You type: N = 7
You type: Z = 42
You type: PRINT STRING\$(N,Z)
It prints: _____

(a) AAA

(b) *****

33. And now, for our finale, let's put fancy lines across the top and bottom of the screen. We will do this by using the STRING\$ function with two ASCII codes of our choice.

```
100 CLS
110 CLEAR 200
120 PRINT @0, STRING$(64,134);
130 PRINT @960, STRING$(63,182);
140 GOTO 140
```



We want 200 characters
of string space.

Why did we print only 63 characters in line 130? Try it with 64 and find out what happens.

To summarize, there are other fancy things that you can do with strings, but you've learned enough in this chapter to use them effectively. Here is a review of what we covered:

- String space is reserved by the CLEAR statement.
- Strings may be joined (catenated) with the + sign.
- The length of a string may be determined by the LEN statement.
- The LEFT\$ statement picks a substring from a string starting from the left.
- The RIGHT\$ statement picks a substring from a string starting from the right.
- The MID\$ statement picks a substring from a string starting from any specified character.
- String comparisons may be made for equality or inequality.
- The ASC statement returns the ASCII code for the first character of a string.
- The CHR\$ statement returns a character when the character's ASCII code is specified.
- The STRING\$ statement repeats a specified character a specified number of times.

Now try our Self-Test, with no strings attached.

SELF-TEST

Congratulations! You have made it through another episode. Now we give you another chance to show how much you have learned.

1. When your computer displays: ?OS ERROR IN 150

what is the problem? _____

2. What can be done about the situation arising in exercise 1?

3. How does the operation A\$ + B\$ differ from A + B? _____

4. Given: A = 15.2 B = 7.3 A\$ = "BIMBO" B\$ = " THE BLIMP"

A + B = _____

A\$ + B\$ = _____

One space

5. If B\$ equals the string shown in exercise 4, what is the value of LEN(B\$)?

6. If A\$ equals the string shown in exercise 4, give the values of the following:

(a) LEFT\$(A\$, 3) = _____

(b) RIGHT\$(A\$, 4) = _____

(c) MID\$(A\$, 2, 3) = _____

7. Complete the following inequalities with the symbols < or >:

A\$ = "ARTISTIC" B\$ = "ARITHMETIC"

C\$ = "BASIC" D\$ = "BASICALLY"

(a) A\$ _____ B\$

(b) B\$ _____ C\$

(c) C\$ _____ D\$

8. Using the strings in exercise 7, complete these inequalities (or equalities):

(a) ASC(A\$) _____ ASC(B\$)

(b) ASC(B\$) _____ ASC(C\$)

(c) ASC(D\$) _____ ASC("ALPHA")

9. If C\$ = "AZBYC":

ASC(MID\$(A\$, 3, 1)) = _____

10. Show the display resulting from a run of this program:

```
100 CLS
110 A$ = "FUNDAMENTAL"
120 B = ASC(MID$(A$, 5, 1))
130 PRINT B
140 PRINT CHR$(B)
```

11. Complete this program, which will display the first thirteen letters of the alphabet with their ASCII codes:

```

100 CLS
110 B = _____
120 E = _____
130 FOR L = B TO E
140 PRINT _____ " = " L
150 NEXT _____

```

12. Show the display resulting from a run of this program:

```

100 CLS
110 A$ = "*" : B$ = "0"
120 PRINT STRING$(8,A$)
130 PRINT
140 PRINT STRING$(4,B$)
150 PRINT @192,STRING$(8,A$) STRING$(4,B$)

```

```

_____ ← 1st line
_____ ← 2nd line
_____ ← 3rd line
_____ ← 4th line

```

Answers to Self-Test

1. Not enough memory space has been reserved for the strings used in the program. (frame 1)
2. Use the statement `CLEAR XXX` where `XXX` reserves enough string space for your program. (frame 2)
3. `A$+B$` joins (or catenates) the strings named by the string variables `A$` and `B$`. `A+B` adds the values of the numeric variables `A` and `B`. (frame 3)

4. A + B = 22.5
A\$ + B\$ = BIMBO THE BLIMP (frame 3)

5. LEN(B\$) = 10 (frame 6)

6. (a) LEFT\$(A\$,3) = BIM (frame 8)

(b) RIGHT\$(A\$,4) = IMBO (frame 11)

(c) MID\$(A\$,2,3) = IMB (frame 13)

7. (a) A\$ > B\$

(b) B\$ < C\$

(c) C\$ < D\$ (frame 23)

8. (a) ASC(A\$) = ASC(B\$)

(b) ASC(B\$) < ASC(C\$)

(c) ASC(D\$) > ASC("ALPHA") (frame 23)

9. ASC(MID\$(A\$,3,1)) = 66 (frames 13, 16-18)

10.

```
65
A
```

(frames 13, 16-18)

11. 110 B = 65
120 E = 77

```
140 PRINT CHR$(L) " = " L
150 NEXT L
```

(frame 24)

12.

```
***** ← 1st line
        ← 2nd line
0000    ← 3rd line
*****0000 ← 4th line
```

(frame 31)

CHAPTER NINE

One-Dimensional Arrays

In this chapter, you will learn to use subscripted variables and one-dimensional arrays. You will learn to INPUT data and store it in an ordered list (an array). You will also learn to READ data from a data list and store it in an array, where it is held for you to use when needed. Each item in the array is numbered so that it can easily be found. You will also become well acquainted with the DIMension statement, which is used to reserve enough memory space to hold your array.

After completing this chapter, you will be able to:

- use the DIMension statement to tell the computer how much space to save for your arrays;
- recognize a subscripted variable;
- assign values to subscripted variables;
- use subscripted variables in your programs;
- use one-dimensional arrays to store the values assigned to subscripted variables;
- recall the values of subscripted variables from memory for use in your programs.

1. Until now, you have used only simple variables called *numeric* or *string variables*. Quite often you have used only one letter, such as:

A K D

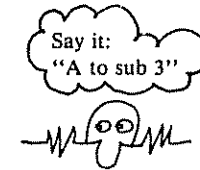
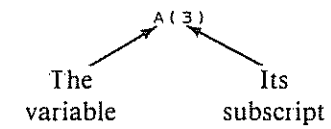
Sometimes you have used a complete word, such as:

OVER DOWN CODE

Other times you have used a letter and a \$ sign for string variables, such as:

A\$ B\$ C\$

Here you'll meet and learn to use a new type of variable called the *subscripted variable*, which consists of a letter followed by a number in parentheses.



The parentheses tell the computer that this is a subscripted variable. This distinguishes it from a simple variable. To review:

- A(3) is a subscripted variable.
- A3 is *not* a subscripted variable. It is a simple variable.
- X(5) is a subscripted variable; X5 is *not* a subscripted variable.
- N\$(7) is a subscripted variable; N\$ is *not* a subscripted variable.

Tell whether the following are subscripted variables or simple variables.

| | | | |
|---------|-------|-----------|-------|
| A (3) | _____ | Y (2 5) | _____ |
| A 3 | _____ | Z \$ | _____ |
| Y | _____ | Z \$(7) | _____ |
| Y 5 | _____ | X 7 | _____ |

| | |
|------------------|--------------------|
| A(3) subscripted | Y(25) subscripted |
| A3 simple | Z\$ simple |
| Y simple | Z\$(7) subscripted |
| Y5 simple | X7 simple |

2. An array is a list of subscripted variables. The variable (such as A or Z\$) names the array (or list) of data, and the number in parentheses (the subscript) labels one element in the array. Thus the computer can store and find one particular element in the array when it is given both the variable and its subscript.

A (0) Each one of these three
 A (1) subscripted variables is an
 A (2) *element* of array A.

If you know that:

A (0) = 5
 A (1) = 4
 A (2) = 3

- (a) What are the three subscripts? _____
- (b) Which element of array A holds the highest value? _____

- (a) 0, 1, and 2
- (b) A(0) (the value is 5)

3. Once again, to understand how a computer stores information, think of the computer's memory as a series of boxes. Array A, below, is held in the computer's memory in the order of its subscripts, 0, 1, 2, 3, 4, 5.

| | |
|------|--|
| A(0) | |
| A(1) | |
| A(2) | |
| A(3) | |
| A(4) | |
| A(5) | |

← This array holds six subscripted variables (elements). Each one is designated by its subscript number in parentheses.

Fill in the above memory locations with the values assigned:

$$A(0) = 32, A(1) = 25, A(5) = 37, A(2) = A(1) + A(0)$$

$$A(3) = A(0) - A(1), A(4) = A(3) + 2$$

| | |
|------|----|
| A(0) | 32 |
| A(1) | 25 |
| A(2) | 57 |
| A(3) | 7 |
| A(4) | 9 |
| A(5) | 37 |

(32 + 25)

(32 - 25)

(7 + 2)

4. Subscripted variables can also have variables as subscripts (the material within parentheses). For example, the subscripted variable D(X) has the variable X as its subscript. Because X is a variable, it can be assigned different values.

If X = 1, then D(X) is D(1).
 If X = 2, then D(X) is D(2).
 (And so on . . .)

Assume that values have been assigned to variables as shown in the following boxes. Note that both simple and subscripted variables are shown here.

| | | | | | |
|------|----|------|-----|---|---|
| D(1) | 10 | P(1) | 358 | X | 1 |
| D(2) | 25 | P(2) | 406 | A | 2 |
| D(3) | -8 | N(1) | 12 | B | 3 |
| D(4) | 37 | N(2) | 3 | C | 4 |

Write the value of each variable below.

D(1) = ____ X = ____ D(B) = ____
 D(4) = ____ N(X) = ____ D(X) = ____
 P(1) = ____ A = ____ D(C) = ____
 N(2) = ____ P(A) = ____ D(A) = ____

 10 1 -8
 37 12 10
 358 2 37
 3 406 25

5. One way to enter values into an array in a program is by using a READ statement and a data list. Although we have labeled elements in arrays in frames 2 and 3 starting with the zero-subscripted variable, A(0), not everyone likes to start with zero, and you don't have to. In the programs in this chapter, we will *not* use the zero-subscripted variable but will start with 1. For example:

```
10 FOR X = 1 TO 8
20   READ D(X)
30 NEXT X
40 FOR X = 1 TO 8
50   PRINT "D(" X ")=" D(X)
60 NEXT X
90 DATA 29,0,-14,75,43,1.5,14,22
```

← This reads eight values into the array D.

← This prints the values of array D.

What subscripts will be used for D(X)? _____

 1, 2, 3, 4, 5, 6, 7, 8

6. The READ statement at line 20 in the program of frame 5 reads from the data list and assigns values to the elements in the array D(X). Note that the array starts from D(1), not D(0). Fill in the values that would be printed if the program was run.

D(1) _____ D(2) _____
 D(3) _____ D(4) _____
 D(5) _____ D(6) _____
 D(7) _____ D(8) _____

 D(1) = 29 D(2) = 0
 D(3) = -14 D(4) = 75
 D(5) = 43 D(6) = 1.5
 D(7) = 14 D(8) = 22

Now RUN the program and check to see if you get these results.

7. Now, let's try a program that will turn on some stars. To do this, we will first READ ten numbers from a data list and assign them as values of the elements in the array S(K). These numbers will be the screen positions for ten stars.

```
100 REM***READ STAR POSITIONS INTO ARRAY S
110 FOR K = 1 TO 10
120   READ S(K)
130 NEXT K
140 DATA 0,111,222,333,444,555,666,777,888,999
```

The array of S(1) through S(10) is now filled with numbers: S(1) = 0, S(2) = 111, S(3) = 222, and so on up to S(10) = 999. Note that ten (10) is the largest subscript that the computer will accept for S(K).

Next, let's turn on the stars using the numbers in S(1), S(2), and so on, as printing positions on the screen. We will also include a time delay so that you can see them coming on, one at a time.

```
200 REM***CLEAR SCREEN, THEN TURN ON STARS
210 CLS
220 FOR K = 1 TO 10
230   PRINT @S(K), "*"
240   FOR Z = 1 TO 500 : NEXT Z
250 NEXT K
```

What next? Let's include a fairly long delay, and then a repeat from line 210, like this:

```
300 REM***TIME DELAY, THEN TURN ON STARS AGAIN
310 FOR Z = 1 TO 2000 : NEXT Z
320 GOTO 210
```

Now it's your turn. Rewrite line 220 (and *only* 220!) so that the stars come on in reverse order. That is, we want the star in position S(10) first, then the star in position S(9), then S(8), and so on. How about it?

220 _____

220 FOR K = 10 TO 1 STEP -1

8. Now, try something else. OK? Change the number and positions of the stars. In other words, in the previous frame, rewrite lines 110, 140, and 220 so that stars come on in the following printing positions on the screen.

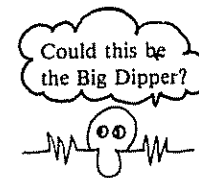
Positions? 392, 337, 411, 547, 559, 748, 740

110 _____

140 _____

220 _____

110 FOR K = 1 TO 7
140 DATA 392,337,411,547,559,748,740
220 FOR K = 1 TO 7



9. We've seen that data can be entered in an array using a READ statement. Data for an array can also be entered by means of an INPUT statement. (We *know* that you already know how to INPUT to arrays. Just be patient.) For now, let's see how we'd use an INPUT statement to see some stars.

```
100 REM***ENTER NUMBER OF STARS AND STAR POSITIONS
110 CLS
120 INPUT "HOW MANY STARS" ; N
130 FOR K = 1 TO N
140   INPUT S(K)
150 NEXT K

200 REM***CLEAR SCREEN, THEN TURN ON STARS
210 CLS
220 FOR K = 1 TO N
230   PRINT @S(K), "*" ;
240   FOR Z = 1 TO 500 : NEXT Z
250 NEXT K

300 REM***TIME DELAY, THEN TURN ON STARS AGAIN
310 FOR Z = 1 TO 2000 : NEXT Z
320 GOTO 210
```

N appears in lines
120, 130, and 220.

We ran the program for the seven star positions used in the previous frame. Here is the first part of the run:

```

HOW MANY STARS?7
?392 ← Value of S(1)
?337 ← Value of S(2)
?411
?547
?559
?748
?740 ← Value of S(7)

```

After typing the seventh star position (740), we pressed **ENTER**. What happened next? _____

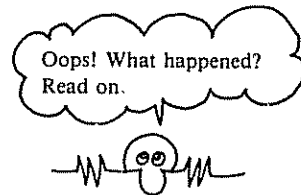
The TRS-80 cleared the screen, then turned on the seven stars, one at a time.

10. Do you want a galactic encore now? We are pleased to present, for your viewing pleasure, a small spiral galaxy. (Very small—only twelve stars.) Let's say we use the program of the preceding frame. Here's the RUN:

```

HOW MANY STARS?12
?480
?547
?485
?419
?353
?350
?411
?538
?605
?673
?677
?BS ERROR IN 140

```



Remember, ten (10) is the largest subscript that the computer will accept for S(K). We got a Bad Subscript (BS) error because we tried to enter a value for S(11). To increase the size of the array to include a subscript beyond the element S(10), we must use a DIMENSION statement as described in the next frame. The TRS-80 reserves space for the element S(0) even when we don't use it. So, for example, when we label elements in an array S(1) through S(10), the computer saves memory space for eleven elements, because it also includes S(0).

- (a) What is the largest subscript in the program in frame 7 that may be used in an array? _____

(b) What is the highest number of elements the computer can hold in this program? _____

- (a) Ten (10)
 (b) Eleven: S(0)-S(10)

11. Let's see how we use the DIMension statement to increase the size of our array. To lengthen our array to fit the galaxy program, use this DIMension statement:

DIM S(12)

This means: Save memory for 13 elements in the array called "S," S(0) through S(12).

Remember, if we didn't DIMension the array in frame 9, we could only have subscripts 0-10 (eleven elements). Since we need twelve stars for our galaxy, though, we need to increase the number of subscripts allowed by the computer to twelve.

Note that the DIMension statement must appear in a program *before* the values of an array are assigned. In fact, it is generally good practice to put all DIMension statements at the *beginning* of a program.

If you add: 105 DIM S(12) to the program of frame 9, will it then assign twelve values to the elements of S? _____

Yes! The twelve values will be put into S(1) through S(12).

This was a tricky question. Remember that we aren't using S(0) in our program. The TRS-80 still has memory space for thirteen elements, S(0) through S(12), though.

12. If we wish to use an array with subscripts larger than 10, we must tell the computer the *largest* subscript it is to permit. To show you how to do this, we'll add a DIM statement to our stars program in frame 9, telling the computer to allow up to 100 star positions in the array S.

Add this statement: 105 DIM S(100)

↑
Maximum subscript permitted in array S.

Now we can show you our small spiral galaxy. Just type RUN and . . .

```

HOW MANY STARS?12
?480
?547
?485
?419
?353
?350
?411
?538
?605
?673
?677
?616

```

No error this time.
 After ENTER is pressed, the screen clears and the spiral galaxy appears, star by star.

13. Hmmmm . . . shouldn't the stars twinkle? Here is the first part of our stars program, which will do this:

```

100 REM***ENTER NUMBER OF STARS AND STAR POSITIONS
105 DIM S(100)
110 CLS
120 INPUT "HOW MANY STARS" ; N
130 FOR K = 1 TO N
140   INPUT S(K)
150 NEXT K

200 REM***CLEAR SCREEN. TURN ON STARS
210 CLS
220 FOR K = 1 TO N
230   PRINT @S(K), "*" ;
240 NEXT K

```

Now it's your turn. Complete the following part of the program to "twinkle" a star picked at random. To do this, the computer must pick a star position S(1), S(2), and so on, up to S(N). Then the computer should turn the star off, and then on . . .

```

300 REM***TWINKLE A RANDOM STAR
310 T = RND(N)
320 _____
330 FOR Z = 1 TO 50 : NEXT Z
340 _____
350 GOTO 310

```

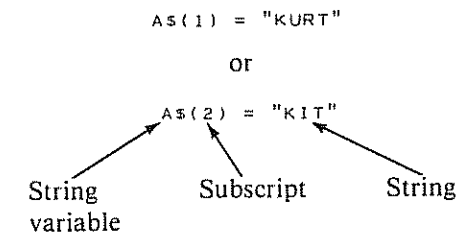
```

-----
320 PRINT @S(T), " ";
340 PRINT @S(T), "*";

```

Line 320 turns the star located at position S(T) *off*. Line 340 turns the same star *on*. Try this program with the Big Dipper star positions in frame 9.

14. In addition to READ and DIMension statements, strings may also be used in arrays as long as they are matched with string variables.



You can use a string array in conjunction with a numeric array. This time we read both with the same READ statement.

```
100 Y$ = "DOLLARS IN THE BANK"
110 FOR X = 1 TO 4
120   READ A$(X),N(X)
130 NEXT X
```

String and numeric
read in pairs.

```
140 CLS
150 FOR X = 1 TO 4
160   PRINT A$(X) " HAS" N(X) Y$
170 NEXT X
```

Strings and numerics
alternated in DATA.

```
180 DATA "KURT",250,"KIT",150,"PHIL",500,"MANDY",175
```

This program would print out the amount that each person has in the bank. Notice that string and numeric values were read in pairs. Therefore the data alternated string and numeric values.

What pair of values is read:

- (a) in the first pass through the loop (lines 110-130)? _____
- (b) in the second pass through the loop? _____
- (c) in the third pass through the loop? _____
- (d) in the fourth pass through the loop? _____

-
- (a) KURT 250
 (b) KIT 150
 (c) PHIL 500
 (d) MANDY 175

15. The program could have read all the names in first and then the numeric values. This would require two READ loops. However, you could have the numeric values read in just before the PRINT statement in the loop in lines 150-170.

Rewrite the program in frame 14 so that it will read in all the names *first*.
Read in the numeric values in the loop (lines 150-170) that contains the PRINT statement.

```

100 Y$ = "DOLLARS IN THE BANK"
110 _____
120 _____
130 _____
140 CLS
150 FOR X = 1 TO 4
160   READ _____
170   PRINT A$(X) " HAS" N(X) Y$
180 NEXT X
190 DATA _____
    
```

```

110 FOR X = 1 TO 4
120 READ A$(X)
130 NEXT X
160 READ N(X)
190 DATA "KURT", "KIT", "PHIL", "MANDY", 250, 150, 500, 175
    
```

16. We could have a running file, or record, of each person's bank account by enlarging the program of frame 14. We will make provisions for deposits and withdrawals from each account.

```

100 CLS
110 FOR X = 1 TO 4
120   READ A$(X), N(X)
130 NEXT X
140 FOR X = 1 TO 4
150   PRINT : PRINT A$(X)
160   INPUT "HOW MUCH (+ FOR DEPOSIT, - FOR WITHDRAW)"; M
170   N(X) = N(X) + M
180 NEXT X
190 FOR W = 1 TO 1000 : NEXT W : CLS
200 FOR X = 1 TO 4
210   PRINT A$(X) " NOW HAS $" N(X)
220 NEXT X
230 FOR W = 1 TO 1000 : NEXT W
240 GOTO 140
300 DATA "KURT", 250, "KIT", 150, "PHIL", 500, "MANDY", 175
    
```

Read in names and initial amounts.

New transaction.

Time delay and clear screen.

Print latest results.

Time delay—then go back.

Quotes are optional in DATA statements.

If the following inputs are made, show the final printed accounts.

| 1st Round | 2nd Round | 3rd Round |
|-----------|-----------|-----------|
| 149.10 | -125 | -50 |
| 25.00 | 75 | 25 |
| 17.23 | -320 | 14 |
| -20.10 | 45.20 | -35 |

KURT \$ _____

PHIL \$ _____

KIT \$ _____

MANDY \$ _____

KURT \$224.1 PHIL \$211.23
KIT \$275 MANDY \$165.10

17. The program of frame 16 allows only four accounts to be used. It should be more general so that you can change the DATA statement for any number of people that you want. You can use an INPUT statement to tell the computer how many accounts are being handled.

```
105 INPUT "HOW MANY ACCOUNTS"; N
```

(a) What lines in the program would have to be changed if line 105 is added?

(b) Show how you would change them:

(a) lines 110, 140, and 200

(b) 110 FOR X = 1 TO N
140 FOR X = 1 TO N
200 FOR X = 1 TO N

18. In the last few frames (14-17), we used string and numeric variables with READ and DATA statements. Now here's your chance to use them, too! You, the chief programmer aboard the starship Horadu VII, are looking at the long-range scanner and see three star systems: Deco, Mala, and Pida. You also see three starbases: Wora, Nuru, and Seda. The distances between star systems and starbases (in warp values) are:

| | <i>Distances (Warp Values)</i> | | |
|------|--------------------------------|-------------|-------------|
| | <i>Wora</i> | <i>Nuru</i> | <i>Seda</i> |
| Deco | 2 | 3 | 5 |
| Mala | 1 | 5 | 7 |
| Pida | 8 | 6 | 2 |

Write a program to:

- READ the name of the star systems into an array SS\$.
 - READ the name of the starbases into an array SB\$.
 - READ the distances into a one-dimensional array DIST.
 - PRINT a table of distances on the screen like the table shown above.
-

Here is a program to perform this task. Unfortunately, portions of the program were erased by overexposure to *zoluta* radiation from the star system Mala. Since you are the chief programmer of Horadu VII, the captain has assigned to you the task of completing the program. So, put up the *zoluta* shields and fix the program.

```

100 REM***WARP VALUE DISPLAY
110 REM***FOR STARSHIP HORADU VII
120 REM***READ STAR SYSTEM NAMES

130 FOR N = 1 TO 3
→140  READ

150 NEXT N
160 REM***READ STAR BASE NAMES
170 FOR N = 1 TO 3

→180  READ
190 NEXT N
200 REM***READ NINE WARP VALUES
210 FOR N = 1 TO 9
→220  READ
230 NEXT N
240 REM***DISPLAY THE TABLE
250 CLS
260 PRINT TAB(4) "DISTANCES(WARP VALUES)"
270 PRINT TAB(4) "*****"

→280 PRINT TAB(7)      $(1) TAB(13)      $(2) TAB(19)      $(3)
290 PRINT TAB(7) "*****" TAB(13) "*****" TAB(19) "*****"
300 FOR N = 1 TO 3

→310  PRINT $(N) TAB(8) (3*N - 2) TAB(14) (3*N - 1) TAB(20) (3*N)
320  PRINT
330 NEXT N
340 END
1000 DATA DECO, MALA, PIDA
1010 DATA WDRA, NURA, SEDA
1020 DATA 2,3,5,1,5,7,8,6,2

```

→ indicates where the *zoluta* radiation damage has occurred.

```

140  READ SS$(N)
180  READ SB$(N)
220  READ DIST(N)
280 PRINT TAB(8) SB$(1) TAB(13) SB$(2) TAB(19) SB$(3)
310 PRINT SS$(N) TAB(8) DIST(3*N - 2) TAB(14) DIST(3*N - 1) TAB(20) DIST(3*N)

```

Your program may be different, and that is OK! What's your next warp vector? Pida! Good, we'll see you back at Nura in fifteen stardates.

19. There may be times when you will want to set all the elements of an array equal to zero. In other words, you may want to use the same array more than once in a program. You could clear out any old values that may be left from a previous solution with three program lines similar to these:

```

400 FOR K = 0 TO 15
410  LET A(K) = 0
420 NEXT K

```

} Clears the elements
A(0) through A(15).

You might also want to clear only certain elements in your array. Suppose we want to clear only the even subscripted elements [A(0), A(2), A(4), etc.] in the above array A(K). Replace line 400 above to clear only the even elements of A(K).

400 _____

400 FOR K = 0 TO 15 STEP 2 (This is the easiest way we can think of.)

20. The following program assigns the values 1 through 15 to fifteen elements of the array A. It prints the array across the top of the screen in a horizontal row, starting with A(1) on the left and ending with A(15) on the right.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

A time delay is used to let you see the results. Then the program puts a zero in each even subscripted element of the array.

1 0 3 0 5 0 7 0 9 0 11 0 13 0 15

The program:

```

100 CLS
105 DIM A(15)
110 FOR X = 1 TO 15
120   A(X) = X
130   PRINT 30 + 3*X, A(X),           Assign and print the array.
140 NEXT X

150 FOR W = 1 TO 1000 : NEXT X

160 FOR K = 1 TO 15 STEP 2
170   LET A(K) = 0
180 NEXT K

190 FOR X = 1 TO 15
200   PRINT 30 + 3*X, A(X)           Print the new array.
210 NEXT X

```

The program prints the original array once and then zeros the even elements once. Can you add one line to make it repeat this alternating cycle over and over again? BE CAREFUL! Remember that you can dimension an array only *once*.

What line would you add and what would it contain? _____

220 GOTO 110 (This is one way. If yours works, HURRAY!)

21. Let's now suppose that you are the treasurer of a computer club that has monthly membership dues of \$1.50. You are using your TRS-80 to review the payment of members. First you write this program:

```

100 DIM A$(20),D(20); D(0) = 0
110 FOR M = 1 TO 20
120   READ A$(M), D(M)
130   D(0) = D(0) + D(M)
140 NEXT M

150 PRINT "TOTAL ASSESSED = $" 20*12*1.50
160 PRINT "TOTAL COLLECTED = $" D(0)
170 PRINT "TOTAL UNCOLLECTED = $" 20*12*1.50 - D(0)

500 DATA J.ABLE ,15, A.BAKER ,18, B.CABLE ,13.50, C.DUNKS ,12
510 DATA D.DUNKS ,3, E.EVERS ,75.0, F.FINK ,16.50, G.GOOD ,18
520 DATA H.HENRY ,18, I.IRVING ,16.50, J.JONES ,18, K.KANE ,16.50
530 DATA L.LEMON ,4.50, M.MARKS ,15, N.NOBLE ,18, O.OTTO ,10.50
540 DATA P.PARKS ,16.50, R.RIOT ,15, S.SINGER ,12, T.TREND ,13
    
```

Read paid dues
and calculate total.

When the program is run, you find out:

```

TOTAL ASSESSED = $ 360
TOTAL COLLECTED = $ 277.5
TOTAL UNCOLLECTED = $ 82.5
    
```

Trailing zeros
are dropped.

Determined to rectify the missing money, you decide to add to the program so that it will print out the amount that each person owes. Complete the lines below that would do this:

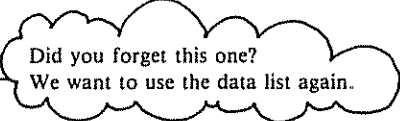
```

180 RESTORE
190 FOR M = _____ TO _____
200   READ A$(M), D(M)

210   OWE = _____

220   PRINT A$(M); " OWES" _____

230 NEXT M
    
```



```

-----
190 FOR M = 1 TO 20
210   OWE = 18 - D(M)
220   PRINT A$(M); " OWES"; OWE
    
```

22. You may use a variable in a DIMension statement to assign the reserved memory for an array.

```

DIM A(N)           DIM A$(M)           DIM B(K + 2)
    
```

If you do this, the DIM statement cannot be at the beginning of the program. The variable *must* be assigned a value *before* the DIMension can be assigned. For example:

```

This will work:  100 INPUT N
                  110 DIM A(N)
This will not work: 100 DIM A(N)
                    110 INPUT N
    
```

The value for N
must be given first.

This program will work:

```

100 CLS
110 INPUT "N = "; N ← Give N first.
120 DIM A(N) ← Then DIM A(N).
130 A(0) = 0

140 FOR X = 1 TO N
150   INPUT A(X)
160   A(0) = A(0) + A(X)
170 NEXT X

180 PRINT "THE SUM OF" N "ELEMENTS IS" A(0)

```

If 25 is entered at line 110, what are the values that can be used for the subscripts of A(N)? _____ through _____

 0 through 25 (26 elements all together)

23. The program in frame 22 allows you to choose how many numbers you wish to add and then sets that value equal to N. It next dimensions array A for the value of N. The numbers to be added are stored in elements A(1) through A(N). Element A(0) is used to hold the sum of the inputs.

Run the program three times with these inputs:

Run 1: N = 9 A(1) through A(N): 1,2,3,4,5,6,7,8,9

Run 2: N = 5 A(1) through A(N): 5,10,15,20,25

Run 3: N = 25 A(1) through A(N): 1,2,3,4,5, . . . 22,23,24,25

What totals did you get for:

(a) Run 1 _____

(b) Run 2 _____

(c) Run 3 _____

 (a) 45; (b) 75; (c) 325

24. Now, add a FOR-NEXT loop to the program of frame 22 that will print the complete array. Then rerun the program with whatever data you desire.

```

190 FOR X = _____
200 _____
210 _____

```

```

190 FOR X = 1 TO N      (or 0 to N)
200   PRINT A(X)      (Format the PRINT statement any way you wish.)
210 NEXT X

```

25. Let's try another program. Our computer club is having its annual open house and is selling coffee, pop, and donuts. You, our club treasurer, are on the job again. You want to program your TRS-80 to keep track of sales. You decide to use arrays to keep track of the sales—coffee sells for 40¢, pop for 35¢, and donuts for 25¢—using C (for coffee), D (for donuts), and P (for pop). You expect sales to stay under 100 for each item. Complete this program:

```

100 CLS
110 DIM C(100), D(100), P(100)
120 C(1) = 0 : D(1) = 0 : P(1) = 0

130 INPUT A$
140 IF A$ = "C" THEN C(1) = C(1) + 1 : GOTO 130
150 IF A$ = "D" THEN D(1) = D(1) + 1 : GOTO 130
160 IF A$ = "P" THEN P(1) = P(1) + 1 : GOTO 130
170 IF A$ = "DONE" GOTO 200
180 PRINT "WHAT IS THAT INPUT AGAIN?"; GOTO 130

200 CLS
210 C(0) = C(1)*.40

220 D(0) = _____
230 P(0) = _____

240 PRINT "CUPS OF COFFEE SOLD ="; _____
250 PRINT "COFFEE RECEIPTS ="; _____
260 PRINT "DONUTS SOLD ="; _____

270 _____
280 _____
290 _____

-----

220 D(0) = D(1)*.25
230 P(0) = P(1)*.35
240 PRINT "CUPS OF COFFEE SOLD ="; C(1)
250 PRINT "COFFEE RECEIPTS ="; C(0)
260 PRINT "DONUTS SOLD ="; D(1)
270 PRINT "DONUT RECEIPTS ="; D(0)
280 PRINT "POP SOLD ="; P(1)
290 PRINT "POP RECEIPTS ="; P(0)

```

← A\$ = $\begin{cases} \text{C for coffee} \\ \text{D for donuts} \\ \text{P for pop} \end{cases}$

26. Suppose you want to check the receipts without disturbing the current counts of coffee, pop, and donuts in the program in frame 25, and you don't want to stop the program. Can you add one line to the program so you could do this? (*Hint:* Where should the computer go from line 290?)

```

300 _____

```

```
300 GOTO 130
```

(This sends the program back for a new value of A\$. C(1), D(1), and P(1) have not been disturbed. The total receipts will be recounted the next time DONE is input.)

You have now completed this chapter and are ready to go to the Self-Test, which follows.

SELF-TEST

Try these Self-Test exercises to show how much you've learned.

1. Put a check before each *subscripted* variable.

____ (a) X3 ____ (e) H(51)
 ____ (b) XY(2) ____ (f) K5
 ____ (c) J(3) ____ (g) 6(K)
 ____ (d) H+2 ____ (h) K(7)

2. Name the elements of the array dimensioned by:

```
DIM E(4)
```

3. If $A = 2$, $B = 5$, and $C = 7$, fill in the values for the array below from these facts.

```

K(0) = B - A
K(B - A) = C
K(2*A) = B
K(A) = B + C
K(1) = K(2) + K(3)
  
```

| | |
|------|--|
| K(0) | |
| K(1) | |
| K(2) | |
| K(3) | |
| K(4) | |

4. Study this program.

```

100 K(0) = 0
110 INPUT N
120 FOR X = 1 TO N
130   READ K(N)
140   K(0) = K(0) + K(N)
150 NEXT X
160 CLS : PRINT "SUM = "; K(0)
200 DATA 1,2,1,2,1,2,1,2,1,2,1,2
  
```

If 6 is entered for N when the program is run, what will be printed?

5. Suppose 12 is entered for N in a run of the program in question 4.
- (a) Will the program print the sum? _____
- (b) If so, what will the sum be? _____
- If not, what message will be printed? _____
6. Modify the banking program in frame 16 so that a message is printed if the account shows a negative balance. Here is a typical response to a negative balance.

```
PHIL HAS A NEGATIVE BALANCE OF $ 15.52
PLEASE MAKE A DEPOSIT AS SOON AS POSSIBLE!
```

Put a check for a negative balance between lines 170 and 180. If a negative balance is found, send the program to a subroutine at line 500 to print the message.

```
175 _____
500 _____
_____
_____
_____
_____
```

7. Write a program that simulates the roll of a single six-sided die 500 times. Use an array to keep the total number of times each side comes up. Remember (or look back to review) how a random number is generated. Have the computer print the results, which would be something like this:

```
1    80
2    91
3    79
4    82
5    72
6    96
```


Answers to Self-Test

1. (b) XY(2) (c) J(3) (e) H(51) (h) K(7) (frame 1)
 2. E(0) E(1) E(2) E(3) E(4) (frame 11)

3.

| | |
|------|----|
| K(0) | 3 |
| K(1) | 19 |
| K(2) | 12 |
| K(3) | 7 |
| K(4) | 5 |

(frames 3, 4)

4. SUM = 9 (frames 3-6)
 5. (a) No (no DIM statement)
 (b) ?BS ERROR IN 130 (10 is the highest subscript available)
 (frame 10)
 6. 175 IF N(X) < 0 GOSUB 500
 500 PRINT A\$(X) "HAS A NEGATIVE BALANCE OF \$" N(X)
 510 PRINT "PLEASE MAKE A DEPOSIT AS SOON AS POSSIBLE!"
 520 RETURN
 (frame 16)
 7. Here is our program. Once again, yours may be different.

```
100 CLS
110 FOR X = 1 TO 6
120   G(X) = 0
130 NEXT X
```

Set totals to zero.

```
140 PRINT 0542, "ROLLING"
150 FOR T = 1 TO 500
160   N = RND(6)
170   G(N) = G(N) + 1
180 NEXT T
```

500 random throws.

```
190 CLS
200 FOR X = 1 TO 6
210   PRINT X; G(X)
220 NEXT X
(frames 21, 22, 25)
```

PRINT results.

CHAPTER TEN

Multidimensional Arrays

In Chapter 9, you used one-dimensional arrays and were introduced to the DIMension statement. In this chapter, you will expand your capability with arrays by learning how to use multidimensional arrays.

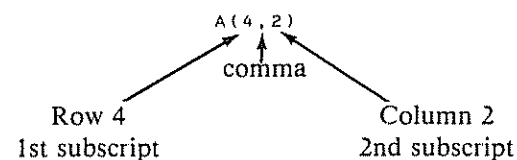
We will extend single-subscripted variables to double-subscripted variables in order to understand two-dimensional arrays. After you have become familiar with two-dimensional arrays, we will move on to those with three dimensions. Whereas one-dimensional arrays are referred to as *lists*, multidimensional arrays are called *tables* or *matrices*.

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

- recognize variables with two, or more, dimensions;
- assign values to multisubscripted variables;
- use the arrays that you have created in programs;
- use the DIMension statement to tell the computer how large your tables will be.

1. Two subscripts are used to assign values to the elements of a two-dimensional array. You can think of a two-dimensional array as a table made up of rows and columns. One subscript assigns the element's row and the other assigns its column.

An element of a two-dimensional array



Here is a table that contains A(4,2) among its elements. What value has been assigned to A(4,2)? _____

Table for A(K,L)

| | | Columns | | | |
|------|---|---------|----|----|----|
| | | 1 | 2 | 3 | 4 |
| Rows | 1 | 5 | 6 | 7 | 8 |
| | 2 | 9 | 10 | 20 | 19 |
| | 3 | 18 | 17 | 16 | 15 |
| | 4 | 14 | 13 | 12 | 11 |
| | 5 | 21 | 22 | 23 | 24 |

$$A(4,2) = 13$$

2. Array A(K,L), used in frame 1, is filled with twenty elements in this order:

| | | Columns 1-4 | | | |
|-------------|---|-------------|--------|--------|--------|
| Rows 1-5 | 1 | A(1,1) | A(1,2) | A(1,3) | A(1,4) |
| | 2 | A(2,1) | A(2,2) | A(2,3) | A(2,4) |
| | 3 | A(3,1) | A(3,2) | A(3,3) | A(3,4) |
| | 4 | A(4,1) | A(4,2) | A(4,3) | A(4,4) |
| | 5 | A(5,1) | A(5,2) | A(5,3) | A(5,4) |

Using the table in frame 1, name the values held by each of these elements.

$$A(3,3) = \underline{\hspace{2cm}} \quad A(2,1) = \underline{\hspace{2cm}}$$

$$A(1,2) = \underline{\hspace{2cm}} \quad A(5,3) = \underline{\hspace{2cm}}$$

$$A(2,4) = \underline{\hspace{2cm}} \quad A(4,2) = \underline{\hspace{2cm}}$$

$$\begin{array}{ll}
 A(3,3) = 16 & A(2,1) = 9 \\
 A(1,2) = 6 & A(5,3) = 23 \\
 A(2,4) = 19 & A(4,2) = 13
 \end{array}$$

3. Like a single-subscripted variable, a double-subscripted variable is a name for a memory location in the computer. It is convenient to think of these locations in terms of rows and columns. Think of the memory "boxes" arranged as in this example.

Array H(I,J)

| | | | | | |
|--------|--|--------|--|--------|--|
| H(1,1) | | H(1,2) | | H(1,3) | |
| H(2,1) | | H(2,2) | | H(2,3) | |

Fill in the boxes above with the values assigned by the following program:

```
100 LET H(1,1) = 12
110 LET H(1,2) = 14
120 LET H(1,3) = 16
130 LET H(2,1) = H(1,1) + H(1,3)
140 LET H(2,2) = H(2,1) - H(1,2)
150 LET H(2,3) = 0
```

| | | | | | |
|--------|----|--------|----|--------|----|
| H(1,1) | 12 | H(1,2) | 14 | H(1,3) | 16 |
| H(2,1) | 28 | H(2,2) | 14 | H(2,3) | 0 |

4. You could also READ in the values from a DATA statement using a nested FOR-NEXT loop.

```
100 REM**READ ROW ONE THEN ROW TWO**
110 FOR I = 1 TO 2
120   FOR J = 1 TO 3
130     READ A(I,J)
140   NEXT J
150 NEXT I
200 DATA 11,12,13,14,15,16
```

Fill in the array values as they will be assigned when the program is run.

| | | | | | |
|--------|--|--------|--|--------|--|
| A(1,1) | | A(1,2) | | A(1,3) | |
| A(2,1) | | A(2,2) | | A(2,3) | |

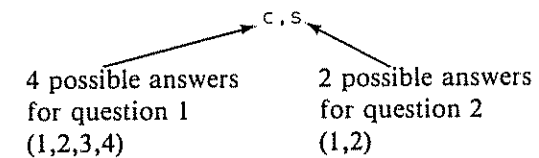
| | | | | | |
|--------|----|--------|----|--------|----|
| A(1,1) | 11 | A(1,2) | 12 | A(1,3) | 13 |
| A(2,1) | 14 | A(2,2) | 15 | A(2,3) | 16 |

5. The secretary of your computer club is conducting a survey of members to see the kinds of computer equipment owned. He knows that the most popular computers are the BLAST I and the GOFUR II. He also wants to determine the ratio of female/male members in the club. He decides to tabulate the data on his TRS-80.

The questionnaire looks like this:

1. Check the brand of computer that you own.
 - 1 _____ BLAST I
 - 2 _____ GOFUR II
 - 3 _____ OTHER
 - 4 _____ NONE
2. Check your sex.
 - 1 _____ FEMALE
 - 2 _____ MALE

Since there are two questions, the secretary would like to set up a two-dimensional array to handle the results of the questionnaire. He decides to use C for computer and S for sex.



Thus a reply of 1,2 would mean a BLAST I computer is owned by a male.

What would a reply of 2,1 mean? _____

A GOFUR II is owned by a female.

6. The secretary then sits down to write the program. But first, he tries to formulate the problem. Since there are four possible answers to question 1, he visualizes four rows in his data table.

| | |
|-------|----------|
| Row 1 | BLAST I |
| Row 2 | GOFUR II |
| Row 3 | OTHER |
| Row 4 | NONE |

Since there are two possible answers to question 2, he visualizes two columns in his data table. Using numbers 1, 2, 3, 4:

| | <i>FEMALE</i> | <i>MALE</i> |
|----------|-----------------------------|----------------------|
| BLAST I | A(1,1) <input type="text"/> | <input type="text"/> |
| GOFUR II | <input type="text"/> | <input type="text"/> |
| OTHER | <input type="text"/> | <input type="text"/> |
| NONE | <input type="text"/> | <input type="text"/> |

Using the array A(C,S) and the numbers 1, 2, 3, 4, he labels each possible reply. Fill in the blanks in front of the boxes above with the correct element labels (the first one is given).

| | <i>FEMALE</i> | <i>MALE</i> |
|----------|-----------------------------|-----------------------------|
| BLAST I | A(1,1) <input type="text"/> | A(1,2) <input type="text"/> |
| GOFUR II | A(2,1) <input type="text"/> | A(2,2) <input type="text"/> |
| OTHER | A(3,1) <input type="text"/> | A(3,2) <input type="text"/> |
| NONE | A(4,1) <input type="text"/> | A(4,2) <input type="text"/> |

7. To make sure that all elements of the array are set to zero at the beginning of the program, the secretary writes these lines:

```
100 REM**FILL ELEMENTS WITH ZERO**
110 FOR S = 1 TO 2
120   FOR C = 1 TO 4
130     A(C,S) = 0
140   NEXT C
150 NEXT S
```

In what order would the computer fill the array with zeros?

- | | |
|---------|---------|
| 1 _____ | 5 _____ |
| 2 _____ | 6 _____ |
| 3 _____ | 7 _____ |
| 4 _____ | 8 _____ |

-
- | | | |
|----------|----------|---|
| 1 A(1,1) | 5 A(1,2) | (Row is changed in the inside loop this time.) |
| 2 A(2,1) | 6 A(2,2) | |
| 3 A(3,1) | 7 A(3,2) | |
| 4 A(4,1) | 8 A(4,2) | |

8. Now that the initialization is done, the secretary tallies the votes this way:

```
200 REM**COUNT EACH VOTE**
210 INPUT N ←————— Number of replies
220 FOR X = 1 TO N
230   READ C,S
240   A(C,S) = A(C,S) + 1
250 NEXT X
```

(a) If a response is checked "NONE" for computer and "MALE" for sex on the questionnaire in frame 5, what would be the value of:

C = _____

S = _____

(b) Which element of the array would be increased by one? _____

(a) C = 4 S = 2

(b) A(4,2)

9. After the responses have been read in and tallied, the secretary prints out the results this way:

```
300 REM**PRINT THE RESULTS**
310 FOR C = 1 TO 4
320   PRINT "A(" C ",1) =" A(C,1),"A(" C ",2) =" A(C,2)
330 NEXT C
```

(a) How many elements (columns) of the array will be printed on each line (row)? _____

(b) How many lines (rows) will be printed? _____

(a) 2 columns per line

(b) 4 lines

10. Then the replies are supplied in the DATA statements at lines 400-420.

```
400 REM**DATA FROM REPLIES**
410 DATA 1,1,2,2,1,2,2,1,3,2,4,2,1,3,1,1,4,1
420 DATA 2,1,3,1,2,2,2,1,4,1,1,1,2,2,1,1,2,1
430 DATA 1,2,4,2,3,2,4,1,2,2,1,1,1,2
```

(a) How many replies were received according to the data list? _____

(b) How many replied that they owned no computer? _____

(a) 25; (b) 5

11. Here is the complete program. Enter it in your TRS-80 and run it. Then fill in the answers to the questions below.

```

100 REM**FILL ELEMENTS WITH ZERO**
110 FOR S = 1 TO 2
120   FOR C = 1 TO 4
130     A(C,S) = 0
140   NEXT C
150 NEXT S

200 REM**COUNT EACH VOTE**
210 INPUT N ← Use 25 here when you run.
220 FOR X = 1 TO N
230   READ C,S
240   A(C,S) = A(C,S) + 1
250 NEXT X

300 REM**PRINT THE RESULTS**
310 FOR C = 1 TO 4
320   PRINT "A(" C ",1) =" A(C,1),"A(" C ",2) =" A(C,2)
330 NEXT C

400 REM**DATA FROM REPLIES**
410 DATA 1,1,2,2,1,2,2,1,3,2,4,2,1,2,1,1,4,1
420 DATA 2,1,3,1,2,2,2,1,4,1,1,1,2,2,1,1,2,1
430 DATA 1,2,4,2,3,2,4,1,2,2,1,1,1,2

```

Fill in your results:

A(1,1) = _____ A(1,2) = _____

A(2,1) = _____ A(2,2) = _____

A(3,1) = _____ A(3,2) = _____

A(4,1) = _____ A(4,2) = _____

A(1,1) = 5 A(1,2) = 4

A(2,1) = 4 A(2,2) = 4

A(3,1) = 1 A(3,2) = 2

A(4,1) = 3 A(4,2) = 2

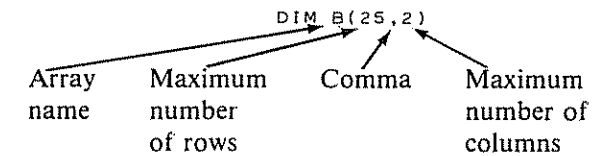
12. The secretary had trouble interpreting the array printed in frame 11. Perhaps you can help. Study the printed results and answer the following questions:

- How many female members replied? _____
 - How many male members replied? _____
 - How many members own BLAST I computers? _____
 - How many female members own computers? _____
-

- (e) How many male members own computers? _____
- (f) Any other interesting results? _____

-
- (a) 13 $A(1,1) + A(2,1) + A(3,1) + A(4,1)$
 - (b) 12 $A(1,2) + A(2,2) + A(3,2) + A(4,2)$
 - (c) 9 $A(1,1) + A(1,2)$
 - (d) 10 $A(1,1) + A(2,1) + A(3,1)$
 - (e) 10 $A(1,2) + A(2,2) + A(3,2)$
 - (f) Answers may vary widely!

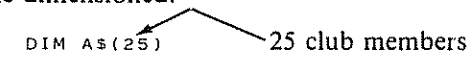
13. The DIM statement for a two-dimensional array is similar to that for one dimension. However, two subscripts must be given. The first subscript designates the maximum number of rows, and the second subscript designates the maximum number of columns. For example:



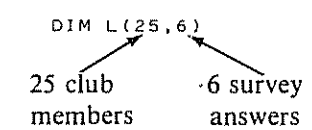
- (a) How many subscripts does a subscripted variable have for a two-dimensional array? _____
- (b) What do the subscripts designate?

-
- (a) two
 - (b) the maximum number of rows (the left value)
the maximum number of columns (the right value)

14. Our computer club secretary has created a new program to display more detailed results of the club survey. He has made a one-dimensional array for the club names, which he dimensioned:



Then he set up this two-dimensional array for the data:



He wanted to create a table something like this:

| | BLAST I | GOFUR II | OTHER | NONE | FEMALE | MALE |
|--------|---------|----------|-------|------|--------|------|
| Name 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| Name 2 | 0 | 1 | 0 | 0 | 0 | 1 |
| Name 3 | 1 | 0 | 0 | 0 | 0 | 1 |
| (etc.) | | | | | | |

In the first section of the program, he would like to put zeros in all the elements of array L. Complete this section.

```

100 REM**SET STRING SPACE AND ZERO ARRAY**
110 CLS : CLEAR 2000 ← String space reserved for names
120 DIM L(25,6), A$(25)

130 FOR X = _____ ← Rows
140   FOR Y = _____ ← Columns
150     L(X,Y) = 0
160   NEXT _____
170 NEXT _____

```

```

-----
130 FOR X = 1 TO 25
140   FOR Y = 1 TO 6
160   NEXT Y
170 NEXT X

```

15. In the second section of the program, the secretary wants to input the names. He also wants to place a one in the correct elements for computer owned and sex. Complete this section.

```

200 REM**GET NAME AND RESPONSE**
210 FOR K = _____
220   INPUT "NAME"; _____
230   READ _____
240   L(K,C) = 1
250   READ _____
260   L(K,S) = 1
270 NEXT

```

```

-----
210 FOR K = 1 TO 25
220   INPUT "NAME"; A$(K)
230   READ C
250   READ S
270 NEXT K

```

16. Now he is ready to write the last section to print out the table. This is how he did it.

```

300 REM**PRINT THE TABLE**
310 CLS
320 PRINT "NAME" TAB(8) "BLAST I" TAB(16) "GOFUR II" TAB(25)
      "OTHER" TAB(31) "NONE" TAB(36) "FEMALE" TAB(43) "MALE"
330 FOR K = 1 TO 25
340   PRINT A$(K) TAB(11) L(K,1) TAB(18) L(K,2) TAB(26) L(K,3)
      TAB(31) L(K,4) TAB(37) L(K,5) TAB(43) L(K,6)
350 NEXT K
360 GOTO 350

```

(a) How many rows of information will be printed by this section of the program? _____

(b) Will it all fit on the screen at one time? _____

(a) 26; (b) no

17. Since the table would not fit on the screen, the secretary split the printout into two parts using subroutines. Study this modification below. Look at the hints, and then complete lines 310, 320, 340, and 350 so that the correct subroutines will be called.

```

300 REM**PRINT THE TABLE**

310 GOSUB _____           To print the column headings
320 GOSUB _____           To print first half of table
330 INPUT "PRESS 'ENTER' TO CONTINUE"; R
340 GOSUB _____           To print the headings again
350 GOSUB _____           To print last half of table
360 INPUT "DO YOU WANT TO SEE THE FIRST PART AGAIN (Y OR N)"; B$
370 IF LEFT$(B$,1) = "Y" THEN 300
380 END

1000 REM**PRINT HEADINGS**
1010 CLS
1020 PRINT "NAME" TAB(8) "BLAST I" TAB(16) "GOFUR II" TAB(25)
      "OTHER" TAB(31) "NONE" TAB(36) "FEMALE" TAB(43) "MALE"
1030 RETURN

2000 REM**PRINT FIRST HALF OF TABLE**
2010 FOR K = 1 TO 13
2020   PRINT A$(K) TAB(11) L(K,1) TAB(18) L(K,2) TAB(26) L(K,3)
      TAB(31) L(K,4) TAB(37) L(K,5) TAB(43) L(K,6)
2030 NEXT K
2040 RETURN

3000 REM**PRINT LAST HALF OF TABLE**
3010 FOR K = 14 TO 25
3020   PRINT A$(K) TAB(11) L(K,1) TAB(18) L(K,2) TAB(26) L(K,3)
      TAB(31) L(K,4) TAB(37) L(K,5) TAB(43) L(K,6)
3030 NEXT K
3040 RETURN

```

```
310 GOSUB 1000
320 GOSUB 2000
340 GOSUB 1000
350 GOSUB 3000
```

18. The secretary's final program, with your modifications, is now ready to run except for the data that will be used. It can be inserted in lines 500 through 530. It is the same except that:

```
500 REMARK**DATA FROM REPLIES**
510 DATA 1,5,2,6,1,6,2,5,3,6,4,6,1,6,1,5,4,5
520 DATA 2,5,3,5,2,6,2,5,4,5,1,5,2,6,1,5,2,5
530 DATA 1,6,4,6,3,6,4,5,2,6,1,5,1,6
```

Compare this data with that used in the program in frame 11.

- (a) All Female 1's have been changed to _____ .
- (b) All Male 2's have been changed to _____ .

 (a) 5's; (b) 6's

19. Here is the list of club members. Run the program and input the names in the order shown.

- | | | |
|------------|------------|------------|
| 1. ABLE | 10. IRVING | 19. SAMPLE |
| 2. BAKER | 11. JONES | 20. TREND |
| 3. CANDY | 12. KANE | 21. URSA |
| 4. C.DUNKS | 13. LEMON | 22. VRUHM |
| 5. D.DUNKS | 14. MARKS | 23. WEST |
| 6. EVERS | 15. NOBLE | 24. YATES |
| 7. FINK | 16. OTTO | 25. ZEBB |
| 8. GOOD | 17. PINKS | |
| 9. HENRY | 18. RIOT | |

Fill in the results of the run.

| | NAME | BLAST I | GOFUR II | OTHER | NONE | FEMALE | MALE |
|---------------------|---------|---------|----------|-------|------|--------|------|
| From 1st half | ABLE | | | | | | |
| | BAKER | | | | | | |
| | CANDY | | | | | | |
| | C.DUNKS | | | | | | |
| | D.DUNKS | | | | | | |
| | EVERS | | | | | | |
| | FINK | | | | | | |
| | GOOD | | | | | | |
| | HENRY | | | | | | |
| | IRVING | | | | | | |
| | JONES | | | | | | |
| | KANE | | | | | | |
| | LEMON | | | | | | |

| | |
|---------------------|--------|
| From 2nd half | MARKS |
| | NOBLE |
| | OTTO |
| | PINKS |
| | RIOT |
| | SAMPLE |
| | TREND |
| | URSA |
| | VRUHM |
| | WEST |
| YATES | |
| ZEBB | |

BLAST I GOFUR II OTHER NONE FEMALE MALE

| | | | | | | |
|---------|---|---|---|---|---|---|
| ABLE | 1 | 0 | 0 | 0 | 1 | 0 |
| BAKER | 0 | 1 | 0 | 0 | 0 | 1 |
| CANDY | 1 | 0 | 0 | 0 | 0 | 1 |
| C.DUNKS | 0 | 1 | 0 | 0 | 1 | 0 |
| D.DUNKS | 0 | 0 | 1 | 0 | 0 | 1 |
| EVERS | 0 | 0 | 0 | 1 | 0 | 1 |
| FINK | 1 | 0 | 0 | 0 | 0 | 1 |
| GOOD | 1 | 0 | 0 | 0 | 1 | 0 |
| HENRY | 0 | 0 | 0 | 1 | 1 | 0 |
| IRVING | 0 | 1 | 0 | 0 | 1 | 0 |
| JONES | 0 | 0 | 1 | 0 | 1 | 0 |
| KANE | 0 | 1 | 0 | 0 | 0 | 1 |
| LEMON | 0 | 1 | 0 | 0 | 1 | 0 |
| MARKS | 0 | 0 | 0 | 1 | 1 | 0 |
| NOBLE | 1 | 0 | 0 | 0 | 1 | 0 |
| OTTO | 0 | 1 | 0 | 0 | 0 | 1 |
| PINKS | 1 | 0 | 0 | 0 | 1 | 0 |
| RIOT | 0 | 1 | 0 | 0 | 1 | 0 |
| SAMPLE | 1 | 0 | 0 | 0 | 0 | 1 |
| TREND | 0 | 0 | 0 | 1 | 0 | 1 |
| URSA | 0 | 0 | 1 | 0 | 0 | 1 |
| VRUHM | 0 | 0 | 0 | 1 | 1 | 0 |
| WEST | 0 | 1 | 0 | 0 | 0 | 1 |
| YATES | 1 | 0 | 0 | 0 | 1 | 0 |
| ZEBB | 1 | 0 | 0 | 0 | 0 | 1 |

20. You could add another section to the program to print the totals of each column. It would have a FOR-NEXT loop something like this:

```
400 REMARK**PRINT TOTALS**
410 CLS
420 FOR C = 1 TO 6
430   T = 0
440   FOR K = 1 TO 25
450     T = T + L(K,C)
460   NEXT K
470   PRINT "ITEM" C "TOTALS" T
480 NEXT C
490 END
```

Delete line 380 from the old program. The END is now at line 490.

After the second half of the table, shown in frame 19, is printed, the computer asks if you want to see the first part again. If your answer is NO, lines 400-490 will be executed. Show what it will print.

```
ITEM 1 TOTALS 9
ITEM 2 TOTALS 8
ITEM 3 TOTALS 3
ITEM 4 TOTALS 5
ITEM 5 TOTALS 13
ITEM 6 TOTALS 12
```

21. Our secretary was still not satisfied with his program. What he really wants to see is a list of each member, the member's computer, and the member's sex. He decides to add another section to his program to do this.

```

600 REM**PRINT INDIVIDUAL DATA**
610 CLS
620 FOR K = 1 TO 25
630   PRINT AS " OWNS ";
640   FOR C = 1 TO 4
650     IF L(K,C) = 1 THEN GOSUB 4000
660   NEXT C
670   PRINT "COMPUTER AND IS A ";
680   FOR C = 5 TO 6
690     IF L(K,C) = 1 THEN GOSUB 5000
700   NEXT C
710   INPUT "PRESS 'ENTER' FOR ANOTHER NAME"; R
720 NEXT K
730 END

4000 REM**PRINT COMPUTER DATA**
4010 IF C = 1 THEN PRINT "A BLAST I ";
4020 IF C = 2 THEN PRINT "A GOFUR II ";
4030 IF C = 3 THEN PRINT "SOME OTHER ";
4040 IF C = 4 THEN PRINT "NO ";
4050 RETURN

5000 REM**PRINT SEX DATA**
5010 IF C = 5 THEN PRINT "FEMALE."
5020 IF C = 6 THEN PRINT "MALE."
5030 RETURN

```

Remove the END statement from line 480. Change line 480 to:

```
480 INPUT "PRESS 'ENTER' TO CONTINUE"; R
```

Run the program again. Then fill in the information that follows from your screen display.

```

ABLE OWNS _____ COMPUTER AND IS A _____
BAKER OWNS _____ COMPUTER AND IS A _____
LEMON OWNS _____ COMPUTER AND IS A _____
TREND OWNS _____ COMPUTER AND IS A _____
URSA OWNS _____ COMPUTER AND IS A _____

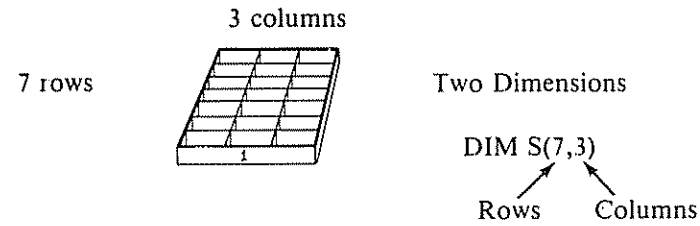
```

```

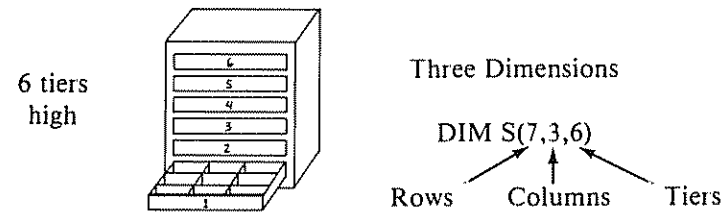
-----
ABLE OWNS A BLAST I COMPUTER AND IS A FEMALE.
BAKER OWNS A GOFUR II COMPUTER AND IS A MALE.
LEMON OWNS A GOFUR II COMPUTER AND IS A FEMALE.
TREND OWNS NO COMPUTER AND IS A MALE.
URSA OWNS SOME OTHER COMPUTER AND IS A MALE.

```

22. We have many versatile people in our computer club. Cliff Sample is in charge of keeping track of all the spare parts for our computers. Each drawer of a cabinet is separated into many compartments. Cliff decided that he could write a program for keeping inventory of all the club's resistors. He decided to think of each drawer and its compartments as being composed of rows and columns like a two-dimensional array.



But the cabinet has 6 drawers. He needs another dimension.



Why not use a three-dimensional array to keep track of the spare resistors?

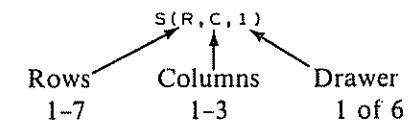
- (a) Each drawer has how many compartments? _____
- (b) How many drawers are in the cabinet? _____
- (c) How many elements will there be in Cliff's array? _____

 (a) 21; (b) 6; (c) 126

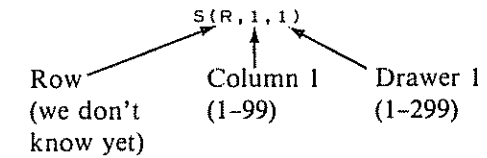
23. Cliff has the *first* drawer filled with resistors varying from 1 ohm to 300 ohms in size. These are arranged in compartments as follows:

- Resistors 1 to 99 ohms in column 1
- Resistors 100 to 199 ohms in column 2
- Resistors 200 to 299 ohms in column 3

Therefore, all the resistors under 300 ohms in size are placed in drawer 1.



We now know how Cliff classifies his resistors by column in drawer 1. A 33-ohm resistor would be stored in:



Give the column and drawer number for these resistors:

280 ohms in S(R,____,____)

120 ohms in S(R,____,____)

20 ohms in S(R,____,____)

280 ohms in S(R,3,1)

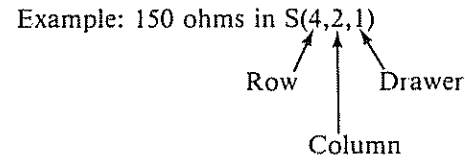
120 ohms in S(R,2,1)

20 ohms in S(R,1,1)

24. The resistor values in drawer 1 are further broken down by rows as follows:

| | <i>Column</i> | | |
|---|---------------|---------|---------|
| | 1 | 2 | 3 |
| 1 | 1-14 | 100-114 | 200-214 |
| 2 | 15-29 | 115-129 | 215-229 |
| 3 | 30-44 | 130-144 | 230-244 |
| 4 | 45-59 | 145-159 | 245-259 |
| 5 | 60-74 | 160-174 | 260-274 |
| 6 | 75-89 | 175-189 | 275-289 |
| 7 | 90-99 | 190-199 | 290-299 |

Using the above table, give the element that would designate the location of each of the following resistors.



80 ohms in S(____,____,____)

220 ohms in S(____,____,____)

170 ohms in S(____,____,____)

40 ohms in S(____,____,____)

120 ohms in S(____,____,____)

80 ohms in S(6,1,1)

220 ohms in S(2,3,1)

170 ohms in S(5,2,1)

40 ohms in S(3,1,1)

120 ohms in S(2,2,1)

25. Cliff follows the same plan for all six drawers of his cabinet. His cabinet will then hold resistors from 1 to (but not including) 1800 ohms. He now acquires a new batch of resistors and writes a program on his TRS-80 to tell him where to put each resistor. Here is his program:

```

200 REM**INPUT NEW DATA**
210 CLS
220 INPUT "RESISTOR SIZE"; S
230 D = INT(S/100) + 1
240 C = INT((S-(3*D - 3)*100)/100) + 1
250 R = INT((S-100*INT(S/100))/15) + 1
260 PRINT S; "OHMS GOES IN S(" R ", " C ", " D ")";
270 GOTO 220
    
```

Calculates the drawer
The column
The row

Use this program to classify these resistors (all measured in ohms):

500, 170, 1200, 1000, 330, 270, 70

Show the display after the run is completed.

```
RESISTOR SIZE? 500
500 OHMS GOES IN S( , , )
```

```
RESISTOR SIZE? 170
170 OHMS GOES IN _____
```

```
RESISTOR SIZE? _____
1200 OHMS GOES IN _____
```

```
RESISTOR SIZE? 500
500 OHMS GOES IN S(1,3,2)
RESISTOR SIZE? 170
170 OHMS GOES IN S(5,2,1)
RESISTOR SIZE? 1200
1200 OHMS GOES IN S(1,1,5)
RESISTOR SIZE? 1000
1000 OHMS GOES IN S(1,2,4)
RESISTOR SIZE? 330
330 OHMS GOES IN S(3,1,2)
RESISTOR SIZE? 270
270 OHMS GOES IN S(5,3,1)
RESISTOR SIZE? 70
70 OHMS GOES IN S(5,1,1)
RESISTOR SIZE? ←
```

Waiting for the next input

26. The program in frame 25 could be used as part of a larger program that updates Cliff's inventory records. The data would get rather large if we demonstrated all six drawers. We'll use only drawer 1. You may expand the program for all six drawers if you wish.

```

100 REM**INITIALIZE AND READ OLD DATA**
110 CLS
120 DIM S(7,3,6) ← Dimensioned for all six drawers
130 D = 1 ← But we set D (drawer) to 1
140 FOR C = 1 TO 3
150   FOR R = 1 TO 7
160     READ N           Read in old inventory
170     S(R,C,1) = N
180   NEXT R
190 NEXT C

200 REM**INPUT NEW DATA**
220 INPUT "RESISTOR SIZE"; S ← Input new resistors
240 C = INT((S-(3*D - 3)*100)/100) + 1
250 R = INT((S-100*INT(S/100))/15) + 1
260 PRINT S; "OHMS GOES IN  S(" R ", " C ",1)"
270 S(R,C,1) = S(R,C,1) + 1 ← Increase inventory
280 INPUT "IF MORE DATA, PRESS 'ENTER'. IF DONE, TYPE D."; AS
290 IF AS <> "D" THEN GOTO 220

300 REM**PRINT NEW INVENTORY**
310 CLS
320 FOR R = 1 TO 7
330   FOR C = 1 TO 3
340     PRINT S(R,C,1)
350   NEXT C
360 PRINT
370 NEXT R
380 GOTO 380

400 REM**OLD INVENTORY DATA**
410 DATA 10,20,30,40,50,60,70
420 DATA 10,30,50,70,60,40,20
430 DATA 70,60,50,40,30,20,10

```

Enter the program and run it. Press the **BREAK** key to stop it after you've filled in the values below.

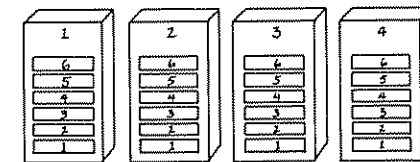
Input values: 10, 150, 220, 90, 50, 200, 33, 270, 15, 18

Show the final array:

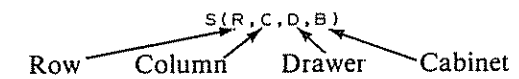
| | | <i>Columns</i> | | |
|-------------|---|----------------|-------|-------|
| | | 1 | 2 | 3 |
| | 1 | _____ | _____ | _____ |
| | 2 | _____ | _____ | _____ |
| <i>Rows</i> | 3 | _____ | _____ | _____ |
| | 4 | _____ | _____ | _____ |
| | 5 | _____ | _____ | _____ |
| | 6 | _____ | _____ | _____ |
| | 7 | _____ | _____ | _____ |

| | | <i>Columns</i> | | |
|-------------|---|----------------|----|----|
| | | 1 | 2 | 3 |
| <i>Rows</i> | 1 | 11 | 10 | 71 |
| | 2 | 22 | 30 | 61 |
| | 3 | 31 | 50 | 50 |
| | 4 | 41 | 71 | 40 |
| | 5 | 50 | 60 | 31 |
| | 6 | 60 | 40 | 20 |
| | 7 | 71 | 20 | 10 |

27. You are now entering the fourth dimension! Cliff has other cabinets filled with resistors. The first cabinet only took sizes up to 1800 ohms. He could number it cabinet 1 and create his own four-dimensional array.



The subscripted variable naming the array could be:



Write a dimension statement for four cabinets with six drawers in each.

Each drawer contains seven rows and three columns. _____

`DIM S(7,3,6,4)`

28. Write a nested FOR-NEXT loop to read the values in the four-dimensional array in frame 27. Make the loop for the cabinet outside, then drawer, then column, then row. Don't worry about the DATA list.

```

400 REM**READ FOUR-DIMENSIONAL DATA**
410 FOR B = 1 TO 4
420   _____
430   _____
440   _____
450   READ N
460   _____ = N
470   NEXT _____
480   _____
490   _____
500   _____

```

```

420 FOR D = 1 TO 6
430   FOR C = 1 TO 3
440     FOR R = 1 TO 7
450       S(R,C,D,B) = N
460     NEXT R
470   NEXT C
480 NEXT D
490 NEXT D
500 NEXT B

```

SELF-TEST

Here's another chance to show how much you have learned.

Remember that a two-dimensional array may be thought of as a table made up of rows and columns. Here is such an array, which we will call A. Questions 1 through 5 refer to array A below.

Array A

| | | | |
|---|---|----|-----------------|
| 5 | 7 | 3 | |
| 4 | 2 | 6 | Rows 1, 2, 3, 4 |
| 1 | 8 | 9 | Columns 1, 2, 3 |
| 7 | 3 | 10 | |

- The value 5 is located in the element named A(1,1). Give the element names (a double-subscripted variable) for:
 - 8 _____
 - 6 _____
 - 10 _____
 - 3 _____ and _____

2. Write a DIMension statement for array A.

120 _____

3. Give the value held in each of the following:

(a) $A(2,1) =$ _____

(b) $A(3,3) =$ _____

(c) $A(4,1) =$ _____

(d) $A(1,2) =$ _____

4. If $X = 2$ and $Y = 3$, what value is held in each of the following elements of array A?

(a) $A(X,Y) =$ _____

(b) $A(X+2,Y-1) =$ _____

(c) $A(X-1,Y-2) =$ _____

5. Write a FOR-NEXT loop to fill the array A in the computer.

200 REM**READ IN DATA**

210 FOR R = _____

220 FOR C = _____

230 _____

240 _____

250 _____

260 _____

500 DATA 5,7,3,4,2,6,1,8,9,7,3,10

6. Fill in the elements in the array below as they would be read in by this program.

100 REM**READ IN DATA**

110 FOR R = 1 TO 5

120 FOR C = 1 TO 4

130 READ N

140 B(R,C) = N

150 NEXT C

160 NEXT R

200 DATA 5,6,7,8,9,10,11,12,20,21

210 DATA 22,23,24,25,1,2,3,4,0,26

| | | Column | | | |
|-----|---|--------|---|---|---|
| | | 1 | 2 | 3 | 4 |
| Row | 1 | | | | |
| | 2 | | | | |
| | 3 | | | | |
| | 4 | | | | |
| | 5 | | | | |

7. Write a program to put zeros in all the elements of the following array.

```

100 REM**ZERO THE ARRAY**
110 DIM A(10,10,3)           Elements A(R,C,D)

120 FOR R = _____
130   FOR C = _____
140     FOR D = _____
150       _____
160     _____
170   _____
180 _____
    
```

8. Look back to the results of the program in frame 11.

- (a) How many club members own GOFUR II computers? _____
- (b) How many members do not own computers? _____
- (c) How many females own BLAST I computers? _____
- (d) How many males own GOFUR II computers? _____

9. From the program run in frame 21, give the names of the club members who own no computer.

| | |
|-------|-------|
| _____ | _____ |
| _____ | _____ |
| _____ | |

10. Use the program in frame 25 to tell in which drawer, column, and row each of these resistors should be placed.

| | | Drawer | Column | Row |
|-----|-----------|--------|--------|-----|
| (a) | 400 ohms | | | |
| (b) | 770 ohms | | | |
| (c) | 1600 ohms | | | |
| (d) | 22 ohms | | | |

Answers to Self-Test

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

1. (a) 8 A(3,2)
 (b) 6 A(2,3)
 (c) 10 A(4,3)
 (d) 3 A(1,3) and A(4,2) (frames 1-3)

2. 120 DIM A(4,3) (frames 13, 14)

3. (a) A(2,1) = 4
 (b) A(3,3) = 9
 (c) A(4,1) = 7
 (d) A(1,2) = 7 (frames 1-3)

4. (a) A(X,Y) = 6 A(2,3)
 (b) A(X+2,Y-1) = 3 A(4,2)
 (c) A(X-1,Y-2) = 5 A(1,1) (Frame 3)

5. 210 FOR R = 1 TO 4 (frames 4, 7-11)
 220 FOR C = 1 TO 3
 230 READ N
 240 A(R,C) = N
 250 NEXT C
 260 NEXT R

- 6.

| | Column | | | |
|---|--------|----|----|----|
| | 1 | 2 | 3 | 4 |
| 1 | 5 | 6 | 7 | 8 |
| 2 | 9 | 10 | 11 | 12 |
| 3 | 20 | 21 | 22 | 23 |
| 4 | 24 | 25 | 1 | 2 |
| 5 | 3 | 4 | 0 | 26 |

(frames 4, 11)

```

7. 120 FOR R = 1 TO 10
    130   FOR C = 1 TO 10
    140     FOR D = 1 TO 3
    150       A(R,C,D) = 0
    160     NEXT D
    170   NEXT C
    180 NEXT R
    
```

(frame 7)

- 8. (a) 8 (4 males, 4 females)
- (b) 5 (3 females, 2 males)
- (c) 5
- (d) 4

(frames 11, 19)

- 9. EVERS TREND (in any order) (frames 19, 21)
- HENRY VRUHM
- MARKS

10.


| | Drawer | Column | Row |
|---------------|--------|--------|-----|
| (a) 400 ohms | 2 | 2 | 1 |
| (b) 770 ohms | 3 | 2 | 5 |
| (c) 1600 ohms | 6 | 2 | 1 |
| (d) 22 ohms | 1 | 1 | 2 |

(frames 25, 26)

CHAPTER ELEVEN

Editing and Debugging Programs

Up to this point in the book you have been shown just two ways to correct mistakes in programs:

- using the backspace key, , to move the cursor to the left, causing characters to be erased (Chapter 2, frame 9).
- retyping the entire program line (Chapter 3, frame 9).

This chapter now introduces you to the other *editing* features that are built into the TRS-80—features that allow you to correct program mistakes quickly and easily. Editing a program is like carefully retyping an important letter you have written to get a job. What you want to do is to correct the typing mistakes, insert some extra phrases and sentences, and *not* disturb the rest of the stuff you have written. The TRS-80 *editor* performs these tasks well.

After completing this chapter and working with the TRS-80's editor, you will find that you can:

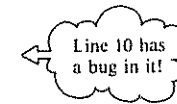
- add information to the end of a program line;
- change the information on a program line;
- delete information from a program line;
- insert information into the middle of a line;
- search any line for repetitions of a character;
- skip to the end of a program line;
- drop the remaining portion of a line;
- use combinations of editing features to quickly make changes and corrections anywhere within a program.

To use the TRS-80 editor, you type the word `EDIT` followed by the line number of the program line to be edited. The computer enters what is called the *edit mode* and waits for whatever commands you might give it. For example, you may wish to change one character in the middle of a program line. With the editor, you will soon discover that it is unnecessary to retype the entire line. You can go into the edit mode and change that one character.

When you enter the proper editing commands, the editor will take care of copying that part of a line you don't wish to disturb. You just enter the changes, corrections, or insertions. The TRS-80 edit mode can save you typing effort. If you are like us, the more typing you do the more chances there are for mistakes. So type less by editing more! The edit mode is easy to learn to use, and will save you time in the long run.

The second thing to be covered in this chapter are some hints on how to go about *debugging* a program. What is this stuff about "debugging"? Do we mean that there are actually bugs in the program?

10



No! "Bugs" is a name for the logic errors and typing mistakes that cause your programs not to work the way you want them to work. Discovering where these "bugs" are and correcting them is called "debugging." (One way to get real bugs in your programs is to store your TRS-80 in a closet or basement for several months. You may also get mice and a few other creatures.)

You will be introduced to two BASIC commands that are used to debug programs: TRON and TROFF. These two commands turn on and turn off, respectively, the TRS-80 *trace* function. The trace function allows you to follow the flow of a program as it executes. But before we get into debugging, let's try our hand at editing.

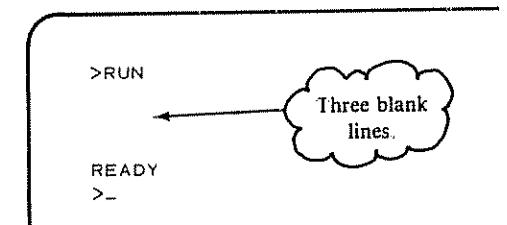
1. As beginners on a computer, most of the editing we do is to correct mistakes. As we do more programming, we find that a lot of the editing we do is to improve our programs—to add features and enhancements that make the programs work better. The TRS-80 editing commands work the same way in both cases. With this frame, we begin to show you each of the features of the TRS-80 editor. The object here is to familiarize you with each editing command—how it works and how it is used, whether on mistakes or program enhancements.

To learn to use the TRS-80 editor, we first need a program to edit. Let's enter one. To do so, we clear the machine of any old programs by typing NEW. Then we enter the following one-line program.

We type: 10 PRINT:PRINT:PRINT

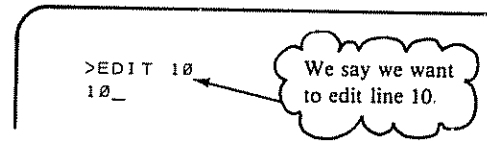
What does this program do? Well, it should print three blank lines on the screen. To see if this is true, we clear the screen and RUN the program.

We type: CLEAR key
RUN



Let's now look at how we can get the TRS-80 into and out of the edit mode. If we type the word EDIT followed by the line number of the program line we wish to edit, the machine is placed in the edit mode. For our current program, here is what happens.

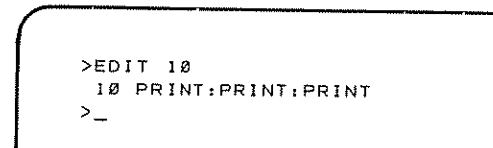
We type: EDIT 10
It prints: 10_



We are now *in* the editor. The TRS-80 is ready to make changes or corrections to line 10. The cursor appears after the number 10 on the screen. As you remember, the cursor indicates that it is our turn to do something. For now, we will *not* make any changes or corrections but will show you how to get out of the edit mode. We press the **ENTER** key.

Pressing the **ENTER** key while in the editor is a signal to the TRS-80 to *exit* the edit mode. The editor prints the rest of the line that is being edited as the exit from the edit mode is taking place.

We type: **ENTER**



So, to enter the edit mode on your TRS-80, you type the word _____ followed by the _____ of the line you wish to edit. Once you are in the edit mode, to leave the editor you simply press the _____ key.

EDIT; line number; **ENTER**

2. The format of the EDIT command that is used to enter the TRS-80 edit mode is:

The word EDIT followed by
the line number of the line to be edited.

If we misspell the word EDIT, the TRS-80 will respond with the ?SN ERROR message. This message is what we get any time we misspell a BASIC language command.

The line number that is specified in the EDIT command must be a valid line number—one that is used in the program. What do you think happens if we ask for a line number that is *not* in the program? You probably guessed it! We get an error message, but one that is slightly different from any seen so far.

To check this out, let's ask for a line number that is not in our current program. Since our program is only one line long, just about any line number will do.

We type: EDIT 15
It types: ?UL ERROR
READY
>_

```
>EDIT 15
?UL ERROR
READY
>_
```

Have you guessed what UL stands for? It is short for Undefined Line. In the EDIT command, we referenced a nonexistent line number. The TRS-80 prints the error message and exits from the edit mode since we can't edit a line that isn't there.

You might try other line numbers that don't exist. Each attempt will result in another ?UL ERROR message. The only line number that will work is line number 10—the valid line number in our current program. Of course, programs can have many valid line numbers based on the number of lines in the program. Our example just happens to have only one line and one line number.

3. In the last two frames, we covered the steps of getting into and out of the editor, and the error messages that can occur in typing in the EDIT command. At this point, we know that you can successfully get into the edit mode and that you are ready to do some actual editing (you might review the past two frames if you want to). We begin by looking at an editing feature that allows you to position the cursor on the line to be edited.

If we press the **Space bar** while in the edit mode, the cursor moves one space to the right on the screen. As the cursor moves, the characters of the line being edited appear on the screen. It's almost as if the computer has the program line "hidden" and the movement of the cursor "uncovered" each character. Each time we press the **Space bar**, one character appears. If we press the bar five times, the first five characters of the program line show right after the line number and the leading space. For our current program, pressing the bar five times causes the word PRINT to appear.

We type: EDIT 10
It types: 10_

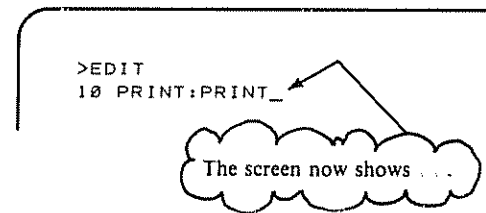
```
>EDIT 10
10_
```

We type: **Space bar**
five times.
It types: 10 PRINT_

```
>EDIT 10
10 PRINT_
```

If we continue to press the bar, soon the entire line would be displayed. But hold it! We said the editor is designed to save you typing effort. There is an easier way. If we enter a number before we press the bar, the editor will move to the right that number of spaces. Look at how this feature works.

We type:
 It types: 10 PRINT:PRINT_



Did you notice that when the was pressed *nothing* happened on the screen? That is because we are in the edit mode. Most of the time, when we are editing, pressing a number is interpreted by the editor to be a *repeat count*—a command to the editor to repeat the next editing command that many times. Ah! The editor is so clever! That is why it can save us typing time.

Now it is your turn. What must you enter to expose the next three characters on the line? The line, after your entry, will look like this:

10 PRINT : PRINT : PR_

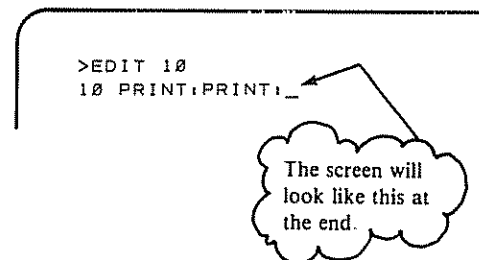
or


Remember, if you wish to exit the edit mode, just press the **ENTER** key. If you run into any difficulty in this frame, press the **ENTER** key until the prompt (>) appears. Then start back at the beginning of the frame with the EDIT 10 command.



4. In the last frame, we moved the cursor to the right on the screen. Here we will learn how to move it to the left. Once we can do this, we can place the cursor anywhere on the line to be edited.

The key that moves the cursor to the left is the key—the left-pointing arrow. Each time the key is depressed, the cursor moves one space to the left, and the character at that position disappears.

We type:
 It types: 10 PRINT:PRINT:P_
 We type:
 It types: 10 PRINT:PRINT: _






How would you move the cursor to the left seven more spaces? One way is to press the  key seven times. Another, and more convenient, way is to use the repeat count feature. It works as it did in the last frame.

We type:  
It types: 10 PRINT_

```
>EDIT 10
10 PRINT_
```

The screen now looks like this.

Once again, when we hit the  key, nothing happened on the screen. The editor simply waited for us to enter the EDIT command that we wished to repeat. You can now move to the right and the left while in the edit mode. Try what you have learned and move the cursor about on the line. Does anything peculiar happen when you move all the way to either the right or the left?

Yes! No matter how many times you hit the  , you never move beyond the last character of the line. Similarly, repeated pressings on the  key will not move you left, past the space after the line number. Try them both and see.

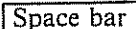

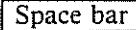


```
10 PRINT:PRINT:PRINT
```

You can move this far left.

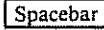
You can move this far right.

5. Move the cursor so that the screen looks like this:

```
>EDIT 10
10 PRINT_
```

If you place a finger from your left hand on the  , a finger from your right hand on the  key, and alternately press each key, what seems to occur? Well, if you start with the  followed by the  key, the colon (:) appears to "flash." If you begin with the  key, the letter T "flashes."

```
>EDIT 10
10 PRINT:|
```

Start with the  and the colon "flashes"

```
>EDIT 10
10 PRINT_
```

What might this be used for? Nothing much, really. But we thought you might be ready for a rest frame, so go ahead and try to make the letters flash. Watch them carefully. Maybe the flashing will begin to hypnotize you. Firedrake the Dragon is mumbling something about making the whole thing into a game. Perhaps you can invent a game based on these two edit features. After all, as the cursor moves to the left, it does seem to “eat” the characters; to the right, it “spits” them back on the screen. Ugh! There must be a better game than that. Why don’t you try to think of one? Maybe the game should have two persons involved: one moving left, one right?

6. Ready to learn more about the edit mode? Good! Now we are going to examine an editing feature called the *search* command.

While editing, it is often difficult to visually count the exact number of characters needed to move the cursor to a particular line position. When the line is long or there are repeated sets of characters, we lose the count easily. The search command can be used to move the cursor quickly and exactly. To activate the search operation in the edit mode, press the S key. We start with our screen showing the following:

(Do whatever you need to do to make your screen look like this. If you are not in the editor, type EDIT 10. If you are still in the edit mode from the last frame, move the cursor to the left as far as it will go.)

```
>EDIT 10
10 _
```

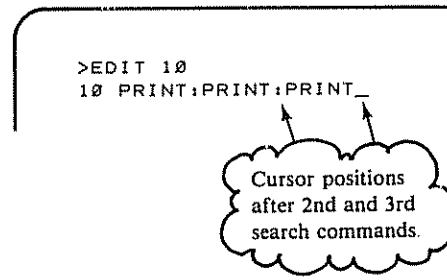
OK! (Did you know that OK comes from the first letters of the birthplace of Martin Van Buren? Or is it from the 1790s phrase *orl korrek*?) Anyway, with the screen as shown above, let’s say we want to move the cursor to the end of the first PRINT statement. We could press the Space bar five times as we did in frame 3. Or we can tell the editor to “search” down the line for the first occurrence of the colon (:) character at the end of the first PRINT statement. We press the S key and then the colon (:). Here is what happens:

We type: S :
 It types: 10 PRINT_

```
>EDIT 10
10 PRINT_
```

If we continue to issue the same search operation, this is what happens:

We type:
 It types: 10 PRINT:PRINT_
 We type:
 It types: 10 PRINT:PRINT:PRINT_



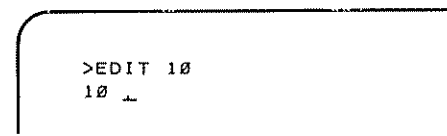
For the last search, there was no colon to the right of the cursor. The editor recognized this condition and placed the cursor at the end of the line. When the search character is not found, the cursor is always moved to the end of the line being edited.

What edit command would you use to move the cursor from the beginning of the following line to the first occurrence of the letter T?

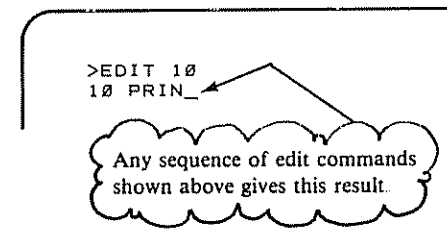
10 PRINT : PRINT : PRINT

Assume you are in edit mode and the screen shows the following:

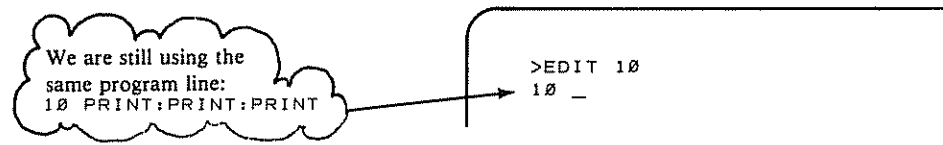
You type: _____



; or ; or followed by the key. You will find that there are often several ways to edit a line. All of them work. As you learn more editing commands, you will discover sequences of editing that are best for you. The TRS-80 editor is flexible and ready to edit a line in whatever way you tell it to do so.

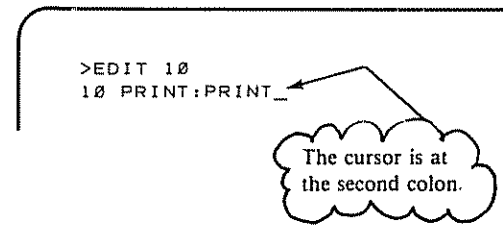


7. We wish to start the next editing sequence at the beginning of line 10. We press **ENTER** (to exit the edit mode from the last frame) and type **EDIT 10** (to restart the editing operation). Our screen shows the following information:



Do you recall the repeat count feature? If we combine the repeat count and the search command, look at what we can do:

We type: **2 S :**
 It types: `10 PRINT:PRINT_`



The repeat count works with the search command. For the example shown, we told the editor to search out the second occurrence of the colon (:). It did so, and it did it quickly! Using this command certainly beats counting the number of characters needed to position the cursor.

By now, you must be getting a good feel for how the search command and the editor are working. Test your skills on the following questions. Assume we are using the same program example:

`10 PRINT:PRINT:PRINT`

If you are at the beginning of the line, in the edit mode, what editing instructions would you use to:

- (a) move the cursor to the third occurrence of the letter I? _____
- (b) move the cursor to the end of the line (the position beyond the last letter in the line, T)? _____

-
- (a) **3 S I** or **1 4 Space bar**. There are many ways to perform this editing task; your way may be different. The edited line will look like this:

`10 PRINT:PRINT:PR_`

- (b) followed by the or entering a repeat count of 17 or greater followed by the . Again, with the TRS-80 editor, there are numerous editing sequences, all of which get the job done. You may have used a different set of commands. The edited line appears on the screen as follows:

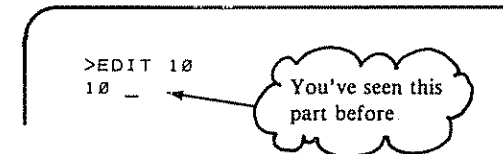
```
10 PRINT:PRINT:PRINT_
```

8. The program that we are editing contains only a single program line. Most programs that we have been writing have several lines. Thus, when we first enter the edit mode, we may not know or remember exactly what each line has in it.

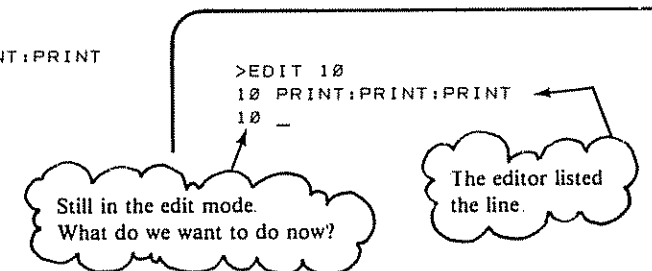
Once more, the TRS-80 editor comes to the rescue. The editor provides us with a *list* command that displays everything on the line currently being edited, starting at the position of the cursor and going to the end of the line. The editor then reprints the line number, dropping down to the next line on the screen, and places the cursor at the beginning of the line.

Let's try this command and see what occurs. We will begin by entering the edit mode.

We type: `EDIT 10`
 It types: `10 _`



We type:
 It types: `10 PRINT:PRINT:PRINT`
`10 _`



As before, the edit command does not appear on the screen. The editor accepts the command, performs the editing operation, and waits for your next instruction. The list command will work anytime, as long as we are not in the middle of another edit operation. If you keep hitting the L key, what happens? Try it. The screen will fill with copies of line 10, but you will still remain in the edit mode.

Let's review some of the editing commands you have tried so far. What keys do you press to:

- (a) Exit the edit mode?
- (b) Move the cursor to the right?
- (c) Move the cursor to the left?

- (d) Search for a character?
- (e) List the contents of a line in the edit mode?

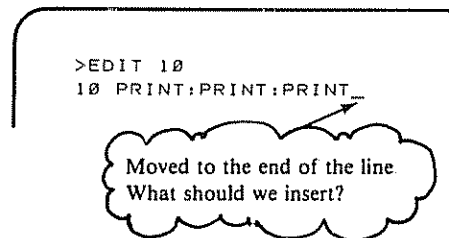
(a) ; (b) ; (c) ; (d) ; (e)

9. What other wonders does the TRS-80 editor possess? More, much more. Let's see. What have we done when we edited a line that the editor might be able to do more easily?

How about moving the cursor to the end of the line? The way we did this operation before was to enter a large repeat count (more than the number of characters in the line) and press the . Or we used the search command to move us out near the end of the line and then hit the several times, as was needed.

But the TRS-80 editor has a better way! It is called the *end-of-the-line-and-insert* command. Wow! That's a long name. You might think that the command must be pretty long, as well. Nope! Remember, the editor is designed to save typing effort. A single key, the **X** key, activates this editing feature. We will use it to move the cursor to the end of the line. As usual, we assume we have just entered the edit mode and that the cursor is at the beginning of the line. (We will not show the screen when the edit starts this time. Why? Because you are probably tired of looking at the same picture . . . just as we are.)

We type:
 It types: 10 PRINT:PRINT:PRINT_



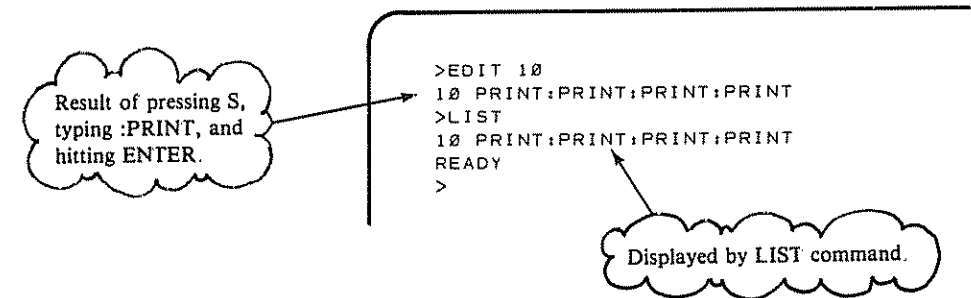
Before, when we moved to the end of the line, the cursor would not go any farther. Pressing the in this case causes the cursor to move. In fact, both the and the key work beyond the end of the line. Try it. Hit the twice, then the key twice. The cursor moves right two spaces, left two spaces, and ends up where it started—at the end of the line. The reason is that the editor is waiting for us to *insert* something.

This command does two things: (1) moves the cursor to the end of the line, and (2) allows us to insert information at that point.

What shall we insert? How about another PRINT statement.

We type: PRINT ENTER
 It types: 10 PRINT:PRINT:PRINT:PRINT

By pressing **ENTER**, we left the edit mode; the new line, with four PRINT statements, is now our new program. If we type **LIST**, the screen should look like this:



10. Based on the editing in the last frame, our program now looks like this:

```
10 PRINT ; PRINT ; PRINT ; PRINT
```

We noted that the **X** command allows us to move the cursor beyond the end of the line being edited. We can also move the cursor to the left, into the line. But watch out!

When we did this operation at the start of this chapter, the cursor made the characters disappear as we moved left. Pressing the **Space bar** caused the characters to reappear. With the **X** command, moving the cursor left causes the characters to disappear, but they are also *deleted* from the line. Let's edit line 10, move to the end of the line, hit the **←** key several times, and look at the results.

We type: **EDIT 10**

It types: 10 _

We type: **X**

It types: 10 PRINT:PRINT:PRINT:PRINT_

We type: **← ← ← ← ← ←**

It types: 10 PRINT:PRINT:PRINT_


Now press the **Space bar** a couple of times. The cursor moves to the right, but the colon and the letter P are gone. In fact, the colon and the fourth PRINT statement have been deleted. In most of the edit commands we have left to examine, the **←** key works as it does with the **X** command. The **←** key deletes characters from the line as the cursor moves to the left.

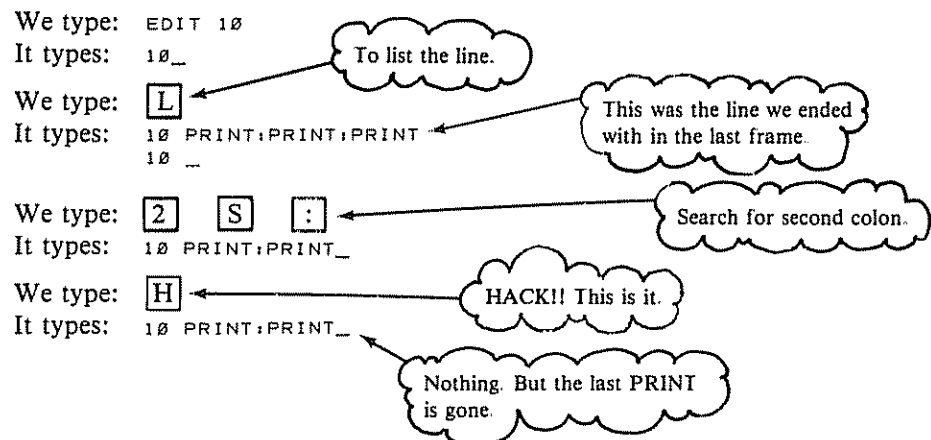
Complete the current editing. Press **ENTER**. Our original program, with three PRINT statements, should now be in the computer's memory. The **X** command, in the edit mode, moves the cursor to the _____ of the program line and waits for you to _____ anything you want. Pressing the **←**


key after entering the **X** command causes the cursor to move to the _____.

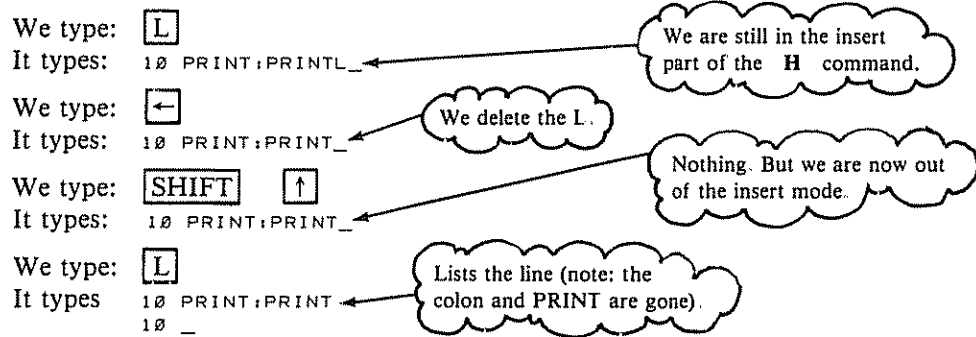
Any characters that the cursor moves over are _____ from the line.

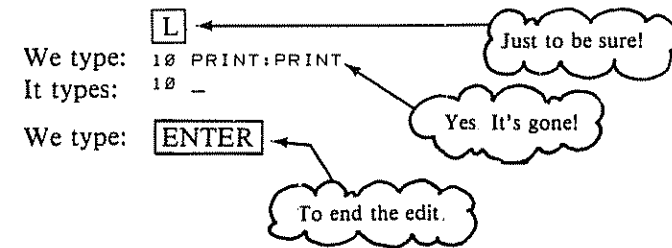
end; insert; left; deleted

11. What is the next thing that the editor has to offer? Why, a command that makes it easy to delete from the cursor position to the end of the line. Instead of moving the cursor to the end of the program line with the **X** command, and then using the  key to delete characters, we can delete characters by using the *hack-and-insert* command. Hack?? That's right, hack! The command is activated by the letter **H**. Pressing the **H** key while in the edit mode drops everything from the cursor position on. Let's try it.

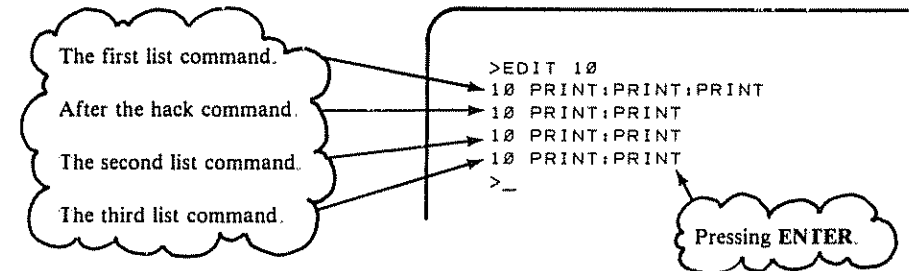


How can we check that the colon and the last PRINT statement have been deleted? The cursor is at the end of the line waiting for us to *insert* anything we care to insert. If we try to type anything, like the **L** command, we will just put an **L** on the end of the line. We have to *escape* from the insert that is being expected. To do so, we hold down the **SHIFT** and press the  key. Nothing happens on the monitor, but if we now press the **L** key, the editor accepts the key as the list command. Let us demonstrate all of this.





The screen shows the following results:



So, to delete all the characters from the position of the cursor to the end of the line, press the _____ key. When you want to escape from the insert part of an edit command, hold down the _____ key and press the _____ key.

H ; **SHIFT** ; **↑**

12. Enter the following line into the computer:

```
10 PRINT:PRINT:PRINT
```

Suppose we wish to drop the last colon and PRINT statement, and replace them by: "HELLO":PRINT"FRIEND". We want line 10 to look like this: 10 PRINT:PRINT"HELLO":PRINT"FRIEND". Can you supply the appropriate editing commands below?

You type: (a) _____
 It types: 10 _

You type: (b) _____
 It types: 10 PRINT:PRINT:PRINT
 10 _

You type: (c) _____
 It types: 10 PRINT:PRINT_

```

You type: (d) _____
It types: 10 PRINT:PRINT_

You type: (e) _____
It types: 10 PRINT:PRINT"HELLO":PRINT"FRIEND"

You type: (f) _____
It types: 10 PRINT:PRINT"HELLO":PRINT"FRIEND"
>
    
```

- (a) `EDIT 10` to activate the edit mode.
- (b) `L` to list the line.
- (c) `2 S :` to move the cursor to the second colon.
- (d) `H` to drop the colon and the last `PRINT` statement. Nothing happens on the monitor. You are left in the insert mode.
- (e) `"HELLO":PRINT"FRIEND"` to insert the new parts of the line.
- (f) `ENTER` to end the edit.

13. Both the **X** command and the **H** command leave us in the insert mode. However, because of what they do first—either move to the end of the line or delete everything to the end of the line—we can only insert new stuff at the end. What happens if we want to insert into the middle of the line? Why, we can use the *insert* command all by itself. And (you probably guessed it) the letter **I** controls the insert mode while we are in the editor.

If we use the program from the last frame:

```
10 PRINT:PRINT"HELLO":PRINT"FRIEND"
```

and change it to the following program:

```
10 PRINT:PRINT"HELLO TRACY":PRINT"MY FRIEND"
```

there are several ways to edit the original line to get the second one. We could use the **S** command to move us to the second quote (") mark; enter the **H** command to drop the rest of the line; type the new information, beginning with the space after the word `HELLO`. That works fine and gets the job done. But let's look at using the insert command to do the same thing.

```

We type: EDIT 10
It types: 10 _

We type: 2 S "
It types: 10 PRINT:PRINT"HELLO_

We type: I TRACY SHIFT ↑
It types: 10 PRINT:PRINT"HELLO TRACY_
    
```

The diagram illustrates the editing process with callouts:

- Skip to the second quote mark.** Points to the `2 S "` command.
- Remember! We are in the insert mode. To leave that mode we must do this.** Points to the `SHIFT ↑` command.
- We type a leading space here.** Points to the `I` command.
- Now out of insert mode; ready for next command.** Points to the `SHIFT ↑` command.

We type: **S** **F** ← Skip to F.
 It types: 10 PRINT:PRINT"HELLO TRACY":PRINT"
 We type: **I** MY **ENTER** ← Edit is complete.
 We type a trailing blank.

It types: 10 PRINT:PRINT"HELLO TRACY":PRINT"MY FRIEND"

RUN the program and check how it looks on the monitor:

We type: RUN

```
>RUN
HELLO TRACY
MY FRIEND
READY
>
```

If you wish to insert something into a line being edited, you press the _____ key, and then type in the characters you want _____ into the line at that point. To leave the insert mode, but still remain in the edit mode, press the **SHIFT** _____.

I ; inserted; **SHIFT** **↑**

14. Let's try the insert command once more. The current program line looks like this:

10 PRINT:PRINT"HELLO TRACY":PRINT"MY FRIEND"

Suppose you wanted to insert the word GOOD into the last PRINT statement. Fill in the sequence of edit commands that does the insertion.

You type: (a) _____
 It types: 10 _ ← Skip to the F.

You type: (b) _____
 It types: 10 PRINT:PRINT"HELLO TRACY":PRINT"MY _ ← Start with insert command.

You type: (c) _____
 It types: 10 PRINT:PRINT"HELLO TRACY":PRINT"MY GOOD _

You type: (d) _____
 It types: 10 PRINT:PRINT"HELLO TRACY":PRINT"MY GOOD FRIEND"
 > ← Exit the edit mode.


- (a) EDIT 10 to enter the edit mode.
- (b) **S** **F** to skip to the letter F. Note: there are other commands that would do the same thing.
- (c) **I**, then type in the word GOOD, and a *trailing space*.
- (d) **ENTER** to terminate the editing.

You can check the line by RUNning the program:

You type: RUN

```

>RUN
HELLO TRACY
MY GOOD FRIEND
READY
>
    
```



15. How about a quick review of all the editing commands so far? What happens when you press the following keys, in the edit mode?

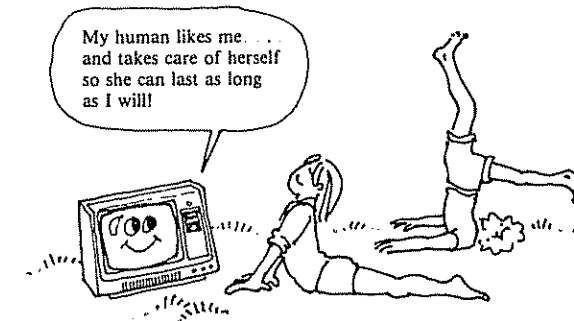
- (a) **ENTER** key: _____
- (b) **Space bar** : _____
- (c) **←** key: _____
- (d) **S** key: _____
- (e) **L** key: _____
- (f) **X** key: _____
- (g) **H** key: _____
- (h) **I** key: _____
- (i) **SHIFT** **↑** : _____

-
- (a) Exit the edit mode.
 - (b) Moves the cursor one space to the right.
 - (c) Moves the cursor one space to the left.
 - (d) Activates the search command. The next character entered is *searched* for in the current line. If found, the cursor moves to the position of the character; if not found, the cursor is placed at the end of the line.
 - (e) Lists the contents of the current line.
 - (f) Positions the cursor at the end of the line, and waits for you to insert new information.
 - (g) Deletes all the characters from the position of the cursor to the end of the line. After the delete, the editor is left in the insert mode. You may insert new information.
-

- (h) Places you in the insert mode, at the current position of the cursor. Whatever you type is inserted into the line.
- (i) Escapes from the insert mode. Can be used with the **X**, **H**, and **I** commands to break the insert operation.

Wow! You are really covering a lot of material on the TRS-80 editor. Why don't you do a few push-ups to refresh yourself, and then charge into the next set of frames . . .

16. Feel better? Did you do something physical: stretch, jog, a few push-ups? Do it. Sitting too long in front of your computer is not always good for your body. A little physical activity also helps sharpen your mind. So take a break!



Great! What else does the editor do that will help you make changes to your program? Perhaps, there is a command that allows you to stop the editor if you have made a mistake while editing. What's that again? That's right, a sort of *editing-the-editing* command.

Actually, this command is called the *cancel-and-restart* command. The key to press for this command is the letter A. You can use this key when you have made a mistake while editing, and you just want to start things over. Let's begin with our original program line, and see how this command works.

```
10 PRINT:PRINT:PRINT
```

Remember this one.

Again, we want to insert the messages "HELLO" and "FRIEND" after the second and third PRINT statements.

We type: EDIT 10

It types: 10 _

We type: **S** **:**

It types: 10 PRINT_

Oops! We meant to go to the second colon.

We type: **I** "HELLO" **SHIFT** **↑**
 It types: 10 PRINT"HELLO" _

Thinking we are ready for the last insert, we skip to the end.
 We don't notice that we are at the wrong place and insert the first message.

We type: **X**
 It types: 10 PRINT"HELLO";PRINT:PRINT_

At this point, we notice that the first insert was incorrect. Good! A chance to use the **A** command.

We type: **SHIFT** **↑**
 Why? Because we must escape from the insert mode. The **X** command left us in the insert mode.
 The restart command.

We type: **A**
 It types: 10 PRINT"HELLO";PRINT:PRINT
 10 _
 Stops the edit and resets to the beginning of the line.
 The list command. We want to check the line.

We type: **L**
 It types: 10 PRINT:PRINT:PRINT
 10 _
 Our original line is back. The incorrect insert is gone.
 Now you can make the changes again.

When a mistake occurs during editing, pressing the **SHIFT** _____ key and then the _____ key will cancel all editing changes and allow you to restart the edit from the beginning of the line.

SHIFT **↑** ; **A**

REMEMBER! The **A** command will not work if you are in the insert mode. You must press the

SHIFT **↑**

key before you enter the **A** command.

! ! !

17. With the editing commands that you know to this point, it still is awkward to delete characters that are in the middle of the line. What you need is the *delete* command. Voila! The TRS-80 editor has just such a command.

The *delete* command is activated by the letter D, and it may be used with a repeat count. The command allows you to delete characters anywhere in the line. For our example line:

```
10 PRINT:PRINT:PRINT
```

if we enter the edit mode and press the 6 and D keys, the first PRINT statement and the colon are deleted. On the screen, a pair of exclamation marks (!) bracket the six characters that are deleted.

We type: EDIT 10
 It types: 10 _

We type: 6 D
 It types: 10 !PRINT;!_

We type: L L
 It types: 10 !PRINT;!PRINT:PRINT
 10 PRINT:PRINT
 10 _

It does this every time.

Exclamation marks appear before and after the deleted characters.

Once to list rest of line; again to list line with characters gone.

The first six characters are deleted.

What happens if you press the A key at this point, followed by the L key?

Show what is on the screen: _____ What if you press

3 D 3 D? What is on the monitor now? _____

```
10 PRINT:PRINT:PRINT
10 _
```

The edit changes are ignored and the original line is listed.

```
10 !PRI!INT;!_
```

The next six characters appear between exclamation marks, indicating that they were deleted.

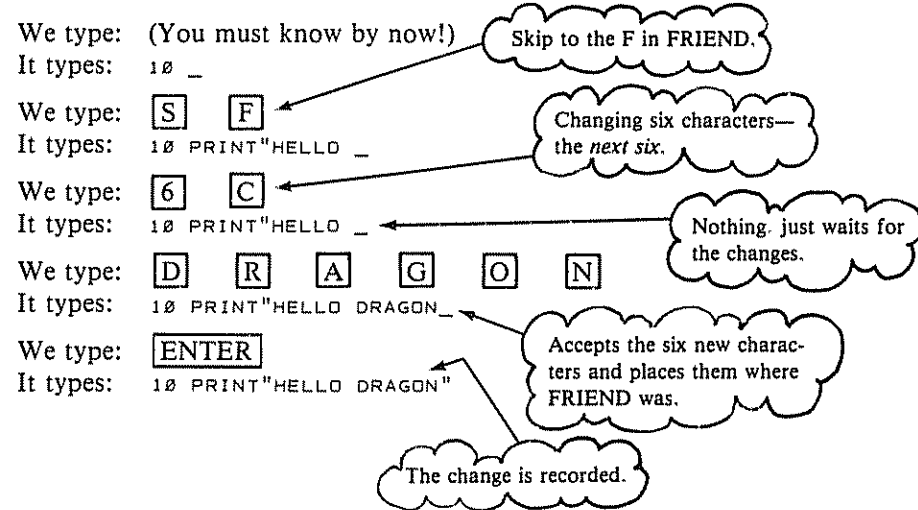
18. In the last frame, we introduced the *delete* command for the TRS-80 editor. The command could be used to delete characters in the middle of the line. There is also an edit command that allows us to *change* characters in the middle of the program line.

You are probably getting ahead of us now. You have most likely guessed that the letter C activates the *change* command while in the edit mode.

Let's take the following program line, and see how the change command works:

```
10 PRINT"HELLO FRIEND"
```

Let's change the word FRIEND to the word DRAGON. They have the same number of letters. This will be easy!



Nothing to it! With a little bit of knowledge, you can now *change* the world. Try other changes on your own.

19. Take the program from the last frame:

```
10 PRINT"HELLO DRAGON"
```

and change it to read:

```
10 PRINT:PRINT:HELLO DRAGON":PRINT
```

Explain how you would go about making these changes to line 10.

One way to edit the line would be to do the following:

- (a) enter the edit mode (EDIT 10)
- (b) insert (I) the six characters PRINT:
- (c) escape from the insert with **SHIFT** **↑**
- (d) skip to the end of the line (X)

- (e) insert the last six characters :PRINT
 (f) press **ENTER** to end the edit.

BUT . . . there are many other ways to make these changes, based on the particular edit commands you wish to use. As long as the line looks, at the end of the edit, like the one above, any set of TRS-80 edit commands can be used.

20. We have been delaying discussion of the next edit command for two reasons:

- We needed to cover the preceding editing commands before we could logically talk about this one, and
- The name of this command is a bit frightening.

Are you ready? This command is called the *search-and-kill* command. Isn't that a horrible command for a friendly computer to have? This edit feature combines the operations of two previous commands: the *search* and the *delete* functions. The letter that is used for this command is the letter K.

Actually, the K command doesn't really "kill" the characters, but only deletes them from the program line. Maybe the computer designers should have named this edit function the "search-and-kindly-remove" command. The letter K would have still been appropriate and the command would not sound so warlike. Oh, well! We can always refer to it as the K command and not think of its name.

Let's try this command on our last program line:

```
10 PRINT:PRINT"HELLO DRAGON":PRINT
```

We enter the editing mode by typing the normal EDIT 10.

We type: **K** **:**
 It types: 10 !PRINT!_

Searches for the first colon and deletes everything up to this character.

Note: It does not delete the colon.

In order to delete the colon, we must issue one more instruction.

We type: **D**
 It types: 10 !PRINT!!!_

But that was more complex than just typing **6 D**, which would have done the same thing. Right! The observation is true for deleting what we did. But what if we had wanted to delete the first two PRINT statements? Well, that's where the K command comes into its own. Proceed to the next frame for more revelations.

21. Cancel the editing changes that have been made by pressing the A key.

We type: **A**
 It types: 10 !PRINT!!!
 10 _

Search for second colon and delete everything up to that point.

We type: **2** **K** **:**
 It types: 10 !PRINT:PRINT"HELLO DRAGON"!_

A simple way to delete the whole two statements, except for the colon. Now, if there was only a way to get that colon at the same time. Maybe there is a way.

Cancel the edit changes once again.

We type: **A**

It types: 10 !PRINT:PRINT"HELLO DRAGON":
10 _

What's this?

We type: **2** **K** **P**

It types: 10 !PRINT:PRINT"HELLO DRAGON":!_

It got both statements, including the colon. Why?

Well, when the search began the cursor was already positioned at the first occurrence of the letter P. Looking to the right, from the cursor, for the next two P's just happens to cause the two beginning statements to be deleted completely. By being clever, we can get the **K** command to do a lot of work for us.

How about a quick review of the last few editing commands we have discussed.

- (a) To cancel editing changes and restart, you press the _____ key.
- (b) To delete characters to the right of the cursor, you use the _____ key.
- (c) If you want to change some characters in the middle of the line, you can use the _____ key.
- (d) Pressing the _____ key combines the two operations of searching and deletion.

(a) **A**; (b) **D**; (c) **C**; (d) **K**

22. Two last editing commands are discussed in this frame. They are presented more for completeness, in case you accidentally hit them in the editing process. They are:

- Save-changes-and-exit command: **E** key
- Cancel-and-exit command: **Q** key

Both cause you to exit the editing mode. The **E** command records all the editing changes you have made before exiting. The **Q** command cancels all changes and then exits. Hitting the **E** key is like pressing **ENTER**, in terms of the results it produces. Pressing the **Q** key is like pressing both the **A** key and the **ENTER** key. Neither command works while you are in the insert mode.

That completes your tour of the edit features of your TRS-80. Why don't you take another stretch break. When you return we will discuss some of the debugging features that are available on your computer. Then we will wrap up

this chapter with a review of all the material that has been discussed, and have some Self-Test questions ready for you to try.

So move away from your chair and get that blood circulating once again.



23. Now we'll go on to debugging programs. There are no exact "rules" for debugging programs. Usually, if a program has an error—it does not do what you want it to do—you must uncover the problem in any way you can.

You can begin by carefully looking over the program. You try to see if there are any obvious typing errors, misplaced lines, or logic problems. But mistakes in programs are often difficult to discover visually. So you move on to more formal procedures, like running the program, and

- tracing the steps the program makes during execution, and
- using PRINT statements to display the contents of variable locations.

Finally, if all else fails, you call on one of your computing friends for help. At this point, several things can happen. The most aggravating event is that your friend looks at the program for thirty seconds, and points out the "obvious" error—the one you've been looking for for over two days!

The second thing to happen is a little less aggravating. Your friend looks at the program for several minutes and, because he or she is seeing the logic "fresh," discovers a subtle flaw or typing mistake that you couldn't readily see.

The third possibility is that the error is not in the program but in the hardware of the computer. This last problem occurs infrequently. If your program doesn't work, *check the program!* If you can't find the error, *check the program again!* If you still can't find anything wrong, call in a friend to *check the program!*

If your program doesn't work the way you want it to work:

THE PROGRAM IS GENERALLY THE PROBLEM!

Debugging a program may involve (true or false?):

- _____ (a) Spraying the inside of your computer with bug spray.
- _____ (b) Visually checking your program for errors.
- _____ (c) Removing tiny microphones the CIA has placed in your program.
- _____ (d) Tracing the flow of a program and possibly displaying the contents of variable locations.
- _____ (e) Dropping your computer from the tenth floor of an office building.
- _____ (f) Having several friends over for beer.

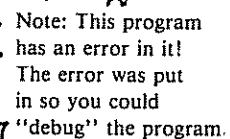
-
- (a) False. In fact, you probably *never* want to spray anything into your computer.
 - (b) True. This process is a good first step to uncovering program errors.
 - (c) False. Really, now!
 - (d) True. If a visual check of the program doesn't uncover the problem, these are good next steps in the debugging process.
 - (e) False. This procedure will only compound your problem by giving you hardware errors as well.
 - (f) True. In fact, in our experience, when we ask a friend for debugging assistance, we end up consuming an average of three beers per person per "bug." At current prices (based on the beer we drink), this aid costs us \$1.89 to \$3.25 per "bug." Cheap labor, but it's heck on productivity.

24. We have cleared our machine by typing NEW and entered the following program:

```

100 REM**POSITIVE, NEGATIVE, ZERO PROGRAM**
110 REM***MESSAGES TO USER
120 CLS
130 PRINT "ENTER A NUMBER AND I WILL TELL YOU IF"
140 PRINT "YOUR NUMBER IS POSITIVE, NEGATIVE, OR ZERO."
150 REM**REQUEST THE NUMBER**
160 PRINT : INPUT "WHAT IS YOUR NUMBER?" ; TEST
170 REM**DETERMINE IF NUMBER IN VARIABLE TEST IS**
180 REM**POSITIVE, NEGATIVE, OR ZERO.**
190 IF TEST > 0 THEN PRINT "YOUR NUMBER IS POSITIVE"
200 IF TEST > 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
210 IF TEST = 0 THEN PRINT "YOUR NUMBER IS ZERO"
220 REM**GO ASK FOR NEXT NUMBER**
230 GOTO 160

```



Note: This program has an error in it! The error was put in so you could "debug" the program.

This program is taken from Chapter 5, frame 12. What does it do (or is supposed to do)? First, it clears the screen and displays a message to the user about the program's function (lines 120 to 140). At line 160, the user's number is accepted and placed in the variable TEST. Lines 190 to 210 check the user's number to see if it is positive, negative, or zero. An appropriate message is displayed based on the results of the checks. The program then loops back (line

230) to get another number for testing. But does the program do all this? Let's RUN the program and look at the results.

We type: RUN

It types: ENTER A NUMBER AND I WILL TELL YOU IF
YOUR NUMBER IS POSITIVE, NEGATIVE, OR ZERO.

WHAT IS YOUR NUMBER? _

We type: 5

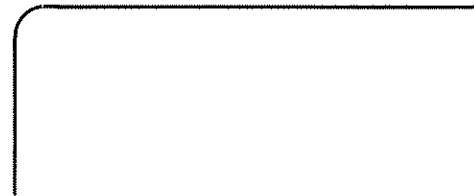
It types: YOUR NUMBER IS POSITIVE
YOUR NUMBER IS NEGATIVE

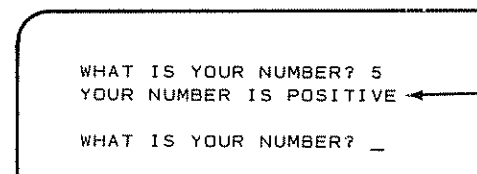
WHAT IS YOUR NUMBER? _

Yes, the number is positive.

Huh!? What's this?

How can this be true? The program says our number, 5, is both positive and negative. On the following empty screen, show what the program should have done (assuming it was working correctly).





The correct display.

What happens if we enter a negative number? Try one and see for yourself. *The program doesn't print anything!* Exceedingly strange! Let's go to the next frame, where we will begin a visual check of the program. We will find that "bug."

25. For the program in the last frame, press the **BREAK** key and stop the program execution. Let's LIST the program, and see if we can visually find the problem. If you already know where the error is, stick with us for a while anyway, as we show others how to go about locating the error.

```

>LIST
100 REM**POSITIVE, NEGATIVE, ZERO PROGRAM**
110 REM**MESSAGES TO THE USER**
120 CLS
130 PRINT "ENTER A NUMBER AND I WILL TELL YOU IF"
140 PRINT "YOUR NUMBER IS POSITIVE, NEGATIVE, OR ZERO."
150 REM**REQUEST THE NUMBER**
160 PRINT : INPUT "WHAT IS YOUR NUMBER?" : TEST
170 REM**DETERMINE IF NUMBER IN VARIABLE TEST IS**
180 REM**POSITIVE, NEGATIVE, OR ZERO**
190 IF TEST > 0 THEN PRINT "YOUR NUMBER IS POSITIVE"
200 IF TEST > 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
210 IF TEST = 0 THEN PRINT "YOUR NUMBER IS ZERO"
220 REM**GO ASK FOR THE NEXT NUMBER**
230 GOTO 160
>_

```

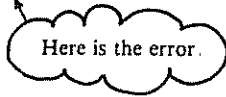
Look at the program. Start at the first line. Pretend you are the computer and mentally "execute" each statement. Lines 120 to 140 seemed to work. The screen cleared and the messages to the user appeared on the video display. The INPUT statement displayed its message. The input value goes into the variable TEST. OK, so far. Let's see if TEST is spelled correctly in lines 190 to 210. Yep, it is! *Misspelling a variable name is a common source of program errors.*

Well, let's look closely at line 190. If TEST is greater than zero, line 190 displays the message "YOUR NUMBER IS POSITIVE." Our number *was* greater than zero and the message *was* printed. Line 190 looks all right.

What about line 200? If TEST is greater than zero . . . what?? Greater than? We just did that check in line 190! Maybe this is the source of the error.

Let's see . . . we've checked for positive values in line 190. Here we want to check for negative numbers. Oh! The relational symbol is reversed—it should be less than (<), not greater than (>). No wonder we got the strange double message when we tried the program. *We* told the computer to print the "negative" message if our number was greater than zero. So much for our typing skills.

```
200 IF TEST > 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"
```



Here is the error.

The error just portrayed is typical of the mistakes that can occur. A simple typing error caused the program logic flow to be altered, producing seemingly unreasonable results. However, the results are perfectly consistent with what we told the computer to do.

If TEST is greater than zero, display:

```
—YOUR NUMBER IS POSITIVE
—YOUR NUMBER IS NEGATIVE
```

Consistent but wrong!!

Don't fix the error just yet. Let's pretend our eagle eyes and quick minds did not find the problem during the visual check. If we still have the "bug," we can explore some of the TRS-80 debugging features.

26. The TRS-80 has two functions, built into its memory, that allow us to *trace* the flow of a program during execution. They are called (guess what?) the *trace* commands. They are:

TRON—to turn the trace feature *on*.

TROFF—to turn the trace feature *off*.

The commands work both as immediate statements (you can enter them directly like NEW, LIST, and RUN commands) and from within a program.

Let's assume that we have been programming for several days, that our eyes are tired, and that we could find our mistake in our program from the last frame. We turn to the trace feature to try and get some insight into the problem.

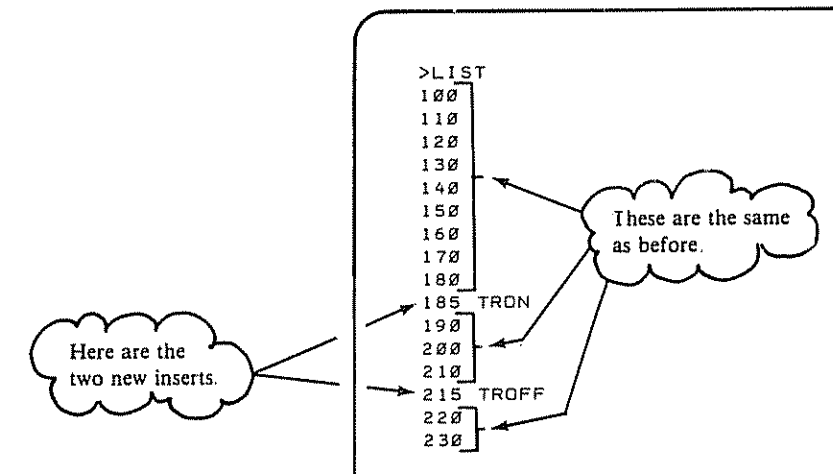
We first alter the program, entering trace commands in the neighborhood of the IF-THEN statements (lines 190-210). Let's say we are convinced that the problem is occurring in that area of the program.

We type: 185 TRON
215 TROFF

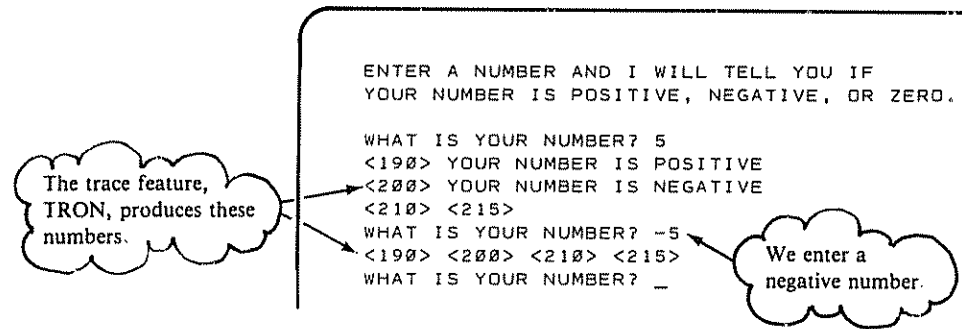
We start the trace here.

We shut it off here.

The trace features now bracket the three statements that contain the IF-THEN checks. LIST the program and see where the new statements fit into the flow.



We RUN the program and enter the value 5 for TEST. Assume, once again, that the program line with the error has not been fixed. Look at what appears on the screen.



The screen is going crazy!! The trace feature causes the line numbers of the program statements between the TRON and TROFF commands to be printed. The line numbers are enclosed in the symbols < and >.

- Line 190 is the IF check for positive numbers.
- Line 200 is the IF check for negative numbers.
- Line 210 is the check for zero.
- Line 215 is the TROFF command.

What can be learned from the trace? For one thing, the program seems to be executing each of the statements. When a negative number is entered, no messages are displayed from the IF statements. But we still have no hint of the error.

You can see from the example, however, just how the trace feature lets you observe the program flow. While TRON is active, the line number of each executing statement is displayed.

To review quickly, the trace feature is turned on by the command _____; turned off by _____; when active, the trace feature prints the _____ of the program statements being executed; the _____ are displayed on the screen enclosed in these two symbols, _____ and _____.

 TRON; TROFF; line number; line numbers; < and >

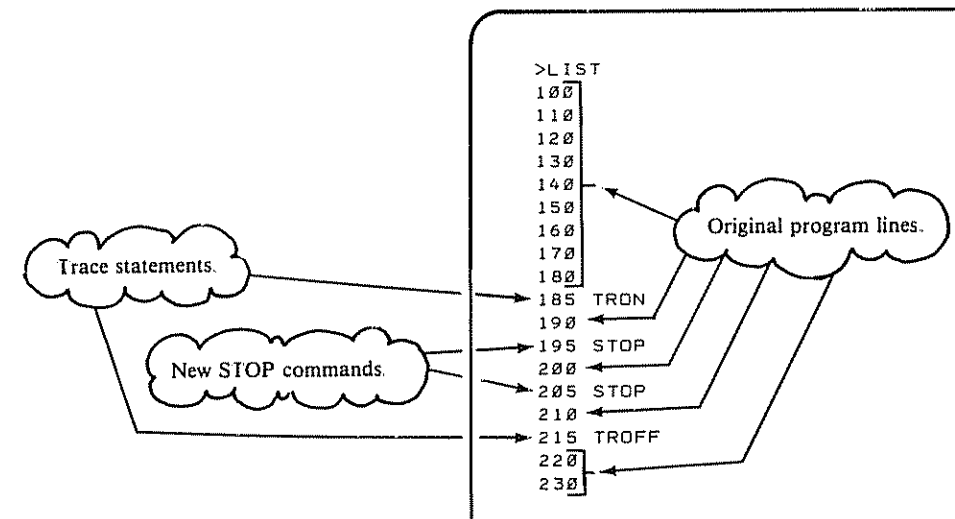
27. Once more, stop the program by pressing the **BREAK** key. We now want to take the debugging and trace commands a bit more slowly. In the last frame, the line numbers from the trace appeared and the program went on to the next line. We want to slow things down and look at each step the program makes.

To do so, we will place STOP statements at lines 195 and 205. The new statements will cause the program to pause at those points. We will then use

some immediate mode statements to do some debugging. Here is how it will work. We enter the two lines:

```
195 STOP
205 STOP
```

If we LIST this new program, we get:



We now RUN the program, enter the value 5 for TEST, and observe the trace and pause features that we have added.

We type: RUN

It types: ENTER A NUMBER AND I WILL TELL YOU IF
YOUR NUMBER IS POSITIVE, NEGATIVE, OR ZERO.

WHAT IS YOUR NUMBER? _

We type: 5

It types: <190> YOUR NUMBER IS POSITIVE
<195>
BREAK IN 195
READY

Trace is on.
Program STOPS.

We type: PRINT TEST

It types: 5

TEST

TEST has the correct value, and line 190 displayed a message indicating that TEST is positive. If there were other variables in the program, they could also be displayed in this manner. To continue the execution of the program, we type CONT, the BASIC command to continue program operations.

We type: CONT
 It types: <200> YOUR NUMBER IS NEGATIVE
 <205>
 BREAK IN 205
 READY

To continue program execution.

Trace still on. Second STOP is encountered.

We type: PRINT TEST
 It types: 5

TEST

TEST still has a value of five (5) and the message at line 200 is being displayed—definitely an error in this part of the program. Let's try one more value for TEST.

We type: CONT
 It types: <210> <215>
 WHAT IS YOUR NUMBER? _

We type: -5
 It types: <190> <195>
 BREAK IN 195
 READY

Program STOPS. No message is printed.

We type: PRINT TEST
 It types: -5

TEST

TEST has the negative number and the message at line 190 is not displayed. This action is correct. We continue.

We type: CONT
 It types: <200> <205>
 BREAK IN 205
 READY

Program STOPS. No message!!

We type: PRINT TEST
 It types: -5

TEST

So TEST still contains the negative number, but the message did not appear. Let's see . . . at line 200 we get the message when we shouldn't (TEST = 5) and we don't get the message when we should (TEST = -5). Line 200 must be the problem.

We type: LIST 200
 It types: 200 IF TEST > 0 THEN PRINT "YOUR NUMBER IS NEGATIVE"

Examination of the line indicates that TEST is spelled correctly. That fact eliminates one possible source of error. Aha! The line says that if TEST is greater than (>) zero, then display the message. There is the culprit! The "greater than" symbol (>) should be a "less than" symbol (<).

If we used the edit mode to fix the error in line 200, and to remove lines 185, 195, 205, and 215, the program would work as intended. Make the changes and try it for yourself.

This frame concludes the discussion on debugging. You were given a brief introduction to the use of the TRS-80 trace features. Using the trace features, you can begin operating as a computer "sleuth," tracking those nasty "bugs" that always seem to creep into programs. We used the trace commands on a simple program with an easy-to-find error to show you how they work. When you encounter errors of a more complex nature, the techniques that were suggested in this frame will provide you with a powerful way to unravel the problems.

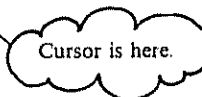
Happy "bug" hunting!!

28. Time for a break. This is the last rest frame before you jump into the Self-Test at the end of the chapter. While you rest, we will review all the material you have touched on in this editing and debugging section of the book. Relax and review. Then move on to the Self-Test section that follows.

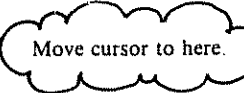
*Review of
Editing Commands/Features*

| | |
|---|---|
| EDIT<line number> | Puts you in the edit mode; <line number> must be a valid line number in the program. |
| ENTER key | Exits from the edit mode. |
| Space bar | Moves cursor one space to right. |
| ← key | Moves cursor one space to left; will also delete characters when used with other edit commands. |
| S key | Search command—searches for the next key that is pressed. |
| L key | List command. |
| X key | End-of-the-line-and-insert command—moves the cursor to the end of the line and waits in the insert mode. |
| H key | Hack-and-insert command—deletes all characters from cursor to the end of the line; waits in insert mode. |
| I key | Insert command—allows insertions in the middle of the line. |
| SHIFT ↑ | Escapes from the insert mode. |
| A key | Cancel-and-restart command—cancels all editing changes and repositions cursor at the beginning of the line. |
| D key | Delete command—deletes characters in the middle of the line. |

In the following questions, assume you are already in the edit mode and positioned at the start of the line. The screen shows:

10

 Cursor is here.

3. What edit keys would you press to move the cursor to the position shown in this line?

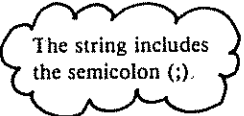
10 A = B : PRINT "YES"; : Z\$ = "FINE"

 Move cursor to here.

Edit keys: _____

4. Using the program line of question 3, how would you insert the word JUST before the word FINE, including a space after JUST?

Edit keys: _____

5. How would you delete the string "YES";?


 The string includes the semicolon (;).

Edit keys: _____

6. How would you go about changing the word YES to SAM?

Edit keys: _____

7. If you press S ; , what else would you do to delete the string:


 :PRINT "YES";

from the line?

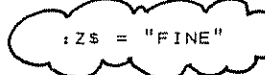
Edit keys: _____

8. What keys do you use to:

(a) List the line in the edit mode? _____

(b) Move the cursor to the end of the line? _____

9. How would you delete the last statement:


 : Z\$ = "FINE"

from the program line?

Edit key: _____

10. How do you, within a program:
- (a) Turn the trace feature *on* at line 75?

 - (b) Turn the trace feature *off* at line 125?

 - (c) Cause the program to stop executing by pausing at line 27?

 - (d) Resume execution after a STOP statement has been executed?

11. What happens when TRON is put into a program?

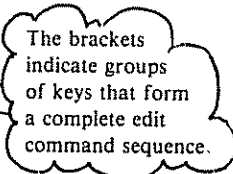
Answers to Self-Test

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

1. (a) The edit mode terminates. All editing changes are recorded. (frame 1)
- (b) The **SHIFT** **↑** stops the insert mode, if you are in an insert subcommand. The **E** key exits the edit mode, saving all edit changes. (frames 11, 22)
- (c) The **SHIFT** **↑** stops the insert subcommand, if any. The **Q** key cancels all edit changes and you leave the edit mode. The program line is unchanged. (frames 11, 22)
2. The **A** key unless you are in the insert mode. Then you must press **SHIFT** **↑** before pressing **A**. (frames 11, 16)

There are usually several ways to make editing changes. The way you do it may be different from the ones shown below. The important thing is: *If your way gets the job done, fine!!* Do it that way.

3. **2** **S** **.** or **1** **7** **Space bar** (frames 3, 6)



The brackets indicate groups of keys that form a complete edit command sequence.

4. S F I J U S T Space bar (frames 6, 13)

or

3 S " Space bar I J U S T Space bar
(frames 6, 13)

5. S " 6 D (frames 6, 17)

S6. Y 3 C S A M (frames 6, 18)

7. K : or 1 3 D (frames 17, 20)

8. (a) L ; (b) X (frames 8, 9)

9. 2 S : H or 2 S : 1 0 D
(frames 6, 11, 17)

10. Put the following lines in the program:

(a) 75 TRON

(b) 125 TROFF

(c) 27 STOP

(d) To continue program execution type: CONT
(frames 26, 27)

11. The line numbers of all the program statements being executed are displayed on the screen. The line numbers are enclosed in the symbols < and >. The display of line numbers continues until a TROFF command is executed. (frame 26)

CHAPTER TWELVE

Graphics, Games, and Programs for the Home

Sit back and *relax*. It's almost graduation time. You have covered a lot of material in the last eleven chapters on how to program your TRS-80 computer. In this chapter, you will learn to do even more. What you are about to learn is not like the byte-(oops—bite-)sized stuff that you got used to in Chapters 1 through 11. The other chapters have slowly and carefully presented you with information about the BASIC language. Now we cut loose and use what you have learned to show you *how to* take some first steps in the direction of *building* complete programs.

Since you will need to put your attention on studying and understanding the programs that we present, this chapter contains *no* Self-Test. We encourage you to spend your time following the step-by-step methods we discuss, and changing and improving the programs. Sound like a lot of work? Yes, it will be. But what we show you about the graphical abilities of your TRS-80, the programming of games, and programs for the home will also be enjoyable and rewarding. Let's take a peek at what we are going to do.

First, we will tell you more about the TRS-80 *graphics characters* introduced in Chapter 8. These are the "characters" inside your computer that you use to make "pictures" on the screen. Each character has a unique "shape" and is about the size of a letter. You will learn to make the TRS-80 display these symbols from within a program, placing them anywhere on the screen. You will also be shown how to simulate "animation" using graphics characters. Remember . . . you have already done "animation" with letters of the alphabet and strings, in several places, in the preceding chapters.

Next, you will use what you learned about graphics characters to begin the construction of several games. The games will be primarily "fun" games, designed for the young children in your home. If you don't have any young children, it is all right to go borrow some to try the programs on.

Third, you will be shown how to construct a more elaborate game or simulation. In this game, the emphasis will be on using graphics on the screen and the *design* of a logic game that makes you think. By the time you reach this section of the chapter, you will have been introduced several times to a *modular*

approach to building programs. *Modular* refers to programming each part of the game or simulation section by section, testing each section as you go and connecting new sections only after the old sections are already working properly.

The final section, home management programs, shows how you might construct programs for use around the home. Two programs are developed as a starting point from which you are encouraged to create and build your home management *library*. The program library that each of you will produce with your TRS-80 is certain to be unique. We only wish we could hear about them all when the libraries are well developed. Hmmmm . . . write to us at *Recreational Computing*, P.O. Box E, Menlo Park, CA 94025.

We encourage you to type in the programs that are about to be discussed. Some of the programs are lengthy, but worth the typing effort. Run the programs; make modifications; change the programs to do what you want them to do. When you get a program to run the way you want it to run, save it on cassette tape or disk (if you have one) as part of your new "library."

Are you ready? Good! Let's get on with the final chapter!

1. Your TRS-80 has 63 special *graphics characters* built into the BASIC language. These characters are groups of lighted "dots" that can be placed on the screen. In fact, all the characters on the screen (letters, numbers, and special symbols) are constructed from these little dots of light.

The BASIC language function that allows you to display one character from the set of 63 graphics symbols is CHR\$. You have used this function back in Chapter 8 to PRINT letters and numbers on the video display. Do you recall that PRINTing CHR\$(65) produces the same result as PRINT "A"? CHR\$ will work in the direct input mode, so CLEAR the screen and enter the following line of BASIC:

We type: PRINT CHR\$(65)

It types: A
READY
>_

CHR\$ generates a string that is one character long. The character produced is determined by the number placed within the parentheses following CHR\$. If the number is 65, the letter A is generated. Using 66, we get the letter B; 67, the letter C; and so on. You probably notice that typing CHR\$(65) is *not* the best way to get the letter A printed. It is simpler to use the string "A."

But using CHR\$ to produce graphics characters is another story. Look at what happens when we enter the following codes along with CHR\$.

We type: PRINT CHR\$(129)

It types: █
READY
>_

We type: PRINT CHR\$(191)

It types: █
READY
>_

```
>PRINT CHR$(129)
█
READY
>PRINT CHR$(191)
█
READY
>_
```

These are the first and last graphics characters.

The character produced by CHR\$(129) is the first symbol in the graphics character set. CHR\$(191) generates the last character. When PRINT CHR\$(129) is executed, a small rectangle of light appears on the screen. PRINTing CHR\$(191) results in a large rectangle of light being displayed.

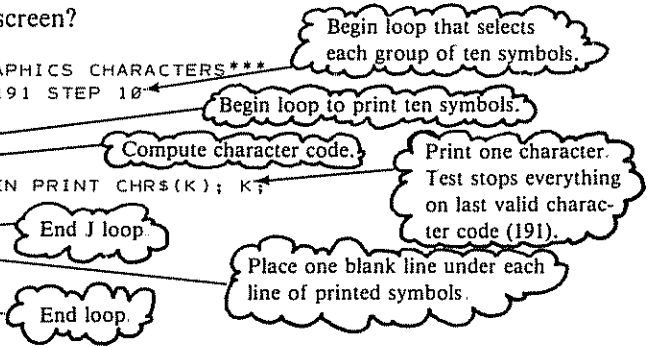
Go ahead and try several numbers between 129 and 191 in a PRINT statement containing CHR\$. There are a lot of "strange" characters inside your TRS-80.

2. If you tried CHR\$ in the immediate mode, you quickly noticed that it is difficult to remember just which graphics code produces each graphics symbol. There are so many of them!

Well, let's write a program that displays all the graphics symbols on the screen, at the same time. Enter the following lines for this program, and RUN it. What appears on the screen?

```

5 REM***DISPLAY GRAPHICS CHARACTERS***
10 FOR I = 129 TO 191 STEP 10
20 FOR J = 0 TO 9
30 K = I + J
40 IF K <= 191 THEN PRINT CHR$(K); K;
50 NEXT J
60 PRINT : PRINT
70 NEXT I
    
```



The screen, when you RUN the program, should fill with lines of graphic symbols and graphic code numbers that produce those symbols. The first symbol is a small rectangle of light (129); the last symbol is a large rectangle (191). In between, the screen is filled with all the other possible graphics symbols that your TRS-80 can produce. Perhaps, at first, it is difficult to see how these symbols might be used. But look closely; some of the symbols naturally suggest how they might be used.

For example, the symbols for 151 (█) and 171 (█) look like the top of a periscope. Need a telephone? How about using 183 (█) and 187 (█)? The symbol 153 (█) looks like a simple airplane. (Well, use your imagination a bit!)

There are many other possibilities. But first, let's discuss just how each character is constructed from "dots" of light.

3. If you look at symbol 129 (█) closely, you can see that it is made up of four small dots of light. You may have to adjust the contrast (the C knob on the front of the video monitor) to see the dots clearly. In fact, upon inspection, all the graphics symbols are made up of these dots of light.

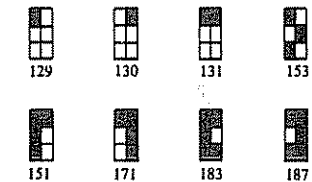
But what is the difference between characters 129 and 130? At first, both look like they are the same character (█). Examine them carefully. They are

almost identical, except for their placement on the screen. The character 131 (█) tells the story. This symbol is a combination of 129 and 130.

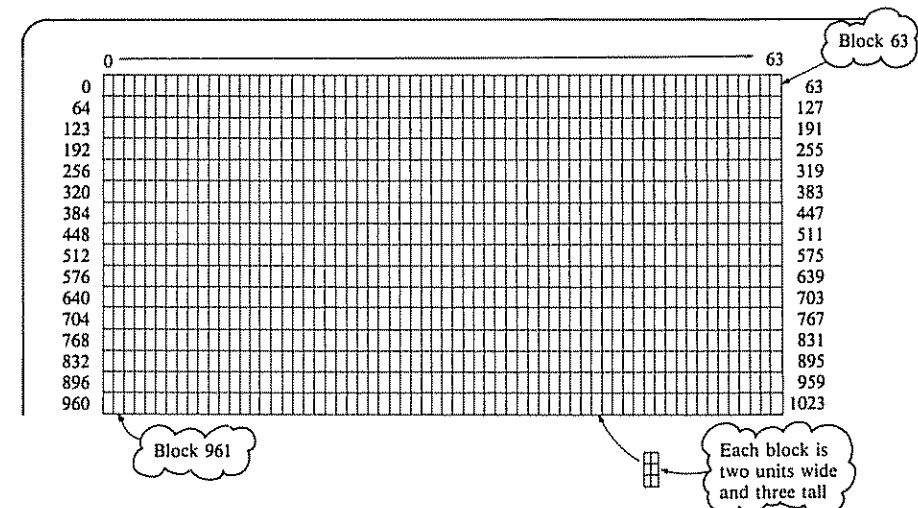


In a way, we could say that character 130 is really the character 129 moved over one column. Also, 131 is really 129 printed where it usually is as well as being moved over one column. You are probably already guessing where this is leading. Yes, all the graphics characters are composed of the basic symbol 129. The presence or absence of the symbol in the various positions of the overall character creates the unique character designs. We know that each character has two column positions. How many rows are there in each character? Look again at character 153 (█).

Yes! There are three rows. So each character is made up of two columns and three rows. The basic building-block character, 129, is then used to construct each individual symbol. Here are some diagrams of the characters we have used up to this point in the discussion:



4. The video monitor on the TRS-80 is divided into "blocks" that are two positions wide and three positions tall. Each line on the monitor contains 64 of these character "blocks," and there are 16 lines. This configuration gives a total of 1,024 "blocks" on the screen. Any graphics character (or standard letter, number, etc.) fits within one of the graphics blocks. The blocks are numbered in the following manner:



The block numbering begins in the upper left corner. That block is numbered zero (0). The blocks on that line go from 0 to 63. Block 64 begins on the next line. The last block on the second line is 127. The last block on the entire screen is 1023. So there are blocks numbered from 0 to 1023, or a total of 1,024 blocks on the monitor.

To address one of these blocks, we use the PRINT @ statement. If we clear the screen and type PRINT @ 480, CHR\$(191), a large rectangle appears near the center of the screen.

Try several block positions and character codes for yourself. What happens if you print two graphics symbols side by side (in two adjacent blocks)? Try it! You might be surprised.

One note of caution: In using the PRINT @ statement, if you hold down the **SHIFT** key while pressing the @ key, the line looks all right but will give you a syntax error. So, if you get a syntax error and your typed line looks correct, retype the line but do not press the **SHIFT** key when pressing the @ symbol.

5. Clear the TRS-80's memory and enter the following program:

```

10 CLS
20 BL = RND(1024) - 1
30 PRINT @BL, CHR$(191);
40 GOTO 20

```

When you run this program, the screen clears and then begins to fill with the large rectangle produced by graphics code 191. Notice the interesting patterns that start to appear. If the program is allowed to continue running, the screen turns into a solid block of white light. (Except for the times that BL = 1023. When this occurs, the whole screen moves up one line when the graphics character is printed at this block location.)

You can stop the program by pressing the **BREAK** key. Changing line 30 to print another graphics symbol produces other patterns on the screen. You may want to rewrite the program so that you can input the graphics code you wish to see printed. You may also want to add something to the program that will allow you to "freeze" the screen so that you can look at the pattern. Hmm! How would you be able to do that? Maybe the INKEY\$ function would help. Go ahead and experiment with the program for a while.

Here is one other experiment you may want to perform. Change the 191 in line 30 to RND(63) + 128. The line will now look like this:



```

30 PRINT @BL, CHR$(RND(63) + 128);


```

What will this new line display? It picks a random graphics character to be printed at the random block location. Try it also!

6. In the last frame, as the characters were printed on the screen, you may have noticed that when two or more symbols are near each other interesting designs start developing.

Earlier, we indicated that two graphics symbols, 183 and 187, look like two telephones, () and (). What will happen if we print these two characters side by side? Let's try it first in the immediate mode.

We type: `PRINT CHR$(183) CHR$(187)`

It types:  ←
READY
>_


This is the combination of 183 and 187 printed together!

When the two symbols are printed next to each other, they appear as one larger symbol. It looks as though they are actually joined together. In one sense, we have "added" them, or joined them to form the new character, the rectangle with a hole in it. In TRS-80 BASIC, there is a formal way to join character combinations. It is called *catenation*. (You did this with strings of letters back in Chapter 8.)

- Take two or more string variables.
- Put a plus sign (+) between variables.
- The result is a string variable made up of all the elements of the individual strings that you started with.

For example, if we put a plus sign (+) between the two CHR\$ functions in the PRINT statement, we get the same result on the screen.

We type: `PRINT CHR$(183) + CHR$(187)`

It types:  ←
READY
>_


The same result!

Why don't you try some combinations of characters and string variables. *Notice:* The result of CHR\$ is a string variable of one character in length!!

7. If we change the program from frame 5, adding another CHR\$ in line 30, we get the following:

```
10 CLS
20 BL = RND(1024) - 1
30 PRINT @BL, CHR$(183) + CHR$(187);
40 GOTO 20
```

Here is new line. Two graphics symbols have been *catenated*!

The joining of the symbols CHR\$(183) and CHR\$(187) give the small rectangle,  , from the last frame. The program, when it is RUN, places the combined shape all over the screen. Change the program and RUN it.

Hit the **BREAK** key when you wish to stop the program. So it is possible to place several graphics symbols on the screen, at the same time, creating a new symbol or form. What happens if we add another character to the two being printed in line 30?

Let's try it. Rewrite line 30, and add the graphics character 153 to what is already being displayed.

```
30 PRINT @BL, CHR$(183) + CHR$(187) + CHR$(153);
```

When RUN, this new program now displays all three characters at once. The combined characters look like a new graphics symbol—one that is put directly on the screen.

Constantly changing and retyping line 30 is a pain in the keyboard! How about a program that lets *you* select the number of graphics codes you wish to see? The program could then ask you for the codes and display the graphic symbol that you design. Look at the following program:

```
5 REM***CREATE-A-CHARACTER PROGRAM***
10 CLS
20 INPUT "NO. OF CHARACTERS"; NC
30 CLS : PRINT @B96, "CHAR CODES:";
40 FOR I = 1 TO NC
50 PRINT @0, STRING$(20," ") : PRINT @0, "CHARACTER CODE";
60 INPUT CC
70 PRINT @470 + I, CHR$(CC);
80 PRINT @903 + 5*I, CC;
90 NEXT I
```

Enter the program into your TRS-80. We will run this program and discuss what it does in the next frame.

8. If we RUN the program in the last frame, this is what occurs:

- The screen is cleared (line 10) and a request appears on the display for the "NO. OF CHARACTERS" (line 20). The input is placed in the variable NC. You may enter a number from 1 to 10.
- The screen clears again, and the message "CHAR CODES:" appears at the bottom of the screen (line 30). The bottom area of the display will be used to store the character codes that you use to construct your graphics symbol.
- A loop is started for the number of codes to be entered (line 40). NC is the value input in line 20.
- The first PRINT statement in line 50 uses the STRING\$ function to clear the first line of the screen. Twenty spaces are printed beginning at block zero (0) of the first line of the display. This action clears the line but does not affect anything else on the screen. The second PRINT statement displays the message that precedes the request for the input of a character code.
- The character code is accepted (line 60) and placed in the variable CC.
- The character code is used to display the graphics symbol (line 70). The symbols are placed on the screen with a PRINT @ statement beginning at block 470+1 or 471. Each successive character is placed in the next available block position. Thus, the characters are joined on the display as if they were one solid symbol.

—The record of the codes you used is updated (line 80). The codes are placed at block 903 plus multiples of 5 times the loop counter, I. Line 90 ends the loop.

Try the program using several different combinations of character codes. What kind of interesting forms can you construct on the screen? What appears if you enter the following five codes: 178, 179, 180, 181, 182? An alien spaceship? It does look like one!

Also, this program constructs combined symbols on only one line of the screen. How would you go about modifying the program so that you could construct larger symbols that use up more of the screen? Let's see . . . if we could input the block number as well as the code we wish to use, then we could place the symbols anywhere. Why don't you give this program modification a try? The current program and the one you might develop based on the above suggestions may be helpful to you when you later wish to design special "characters" for any games or simulations that you put together.



9. Once you have found a set of character codes that you wish to use as a new graphics symbol on the screen, you need a simple way to store and recall the set. One way is to store the entire sequence of CHR\$ produced characters in a string variable. If we store our alien "spaceship" (codes 178, 179, 180, 181, and 182) in the string variable A\$, we can then print the combined symbol at will.

We type: `A$ = CHR$(178) + CHR$(179) + CHR$(180) + CHR$(181) + CHR$(182)`
`PRINT A$`

It types:
 READY
 >_

To use the string variable in a program, we have several methods of creating the combined symbol. Here is one way to do it. Clear the memory of any old programs by typing NEW. Then enter the following program:

```
5 REM***SPACESHIP DISPLAY***
10 CLS
20 FOR I = 178 TO 182
30   A$ = A$ + CHR(I)
40 NEXT I
50 PRINT @ 470, A$
```

Start loop on codes.

Build up string.

Display combined string
near center of monitor.

When a character is "created" in this way in a program, it can then be used anywhere within the program. Using the string variable to hold the symbol saves typing and memory space in the program.

If you have several symbols that are to be used in many places in a program, you can store them in string variables and then recall them when needed. Displaying the symbols with the PRINT @ statement gives you the capability to do a lot of "fancy" graphics on your TRS-80. Putting graphics into your programs will make them more interesting, especially to children.

10. By adding a couple of lines to the program in the last frame, we can turn our "spaceship" into a UFO (Unidentified *Flashing* Object). Add these lines to the program:

```

60 GOSUB 100
70 CLS : GOSUB 100
80 GOTO 50
100 FOR I = 1 TO 200 : NEXT : RETURN
    
```

When you RUN this modified program, the "spaceship" appears near the center of the screen, and winks in and out of existence. If you change the delay counter in line 100, you can make the ship wink either faster or slower.

Press the **BREAK** key when you wish to stop the program. Can you begin to think of uses for the capabilities we have discussed so far? We have covered:

- The TRS-80 graphics characters
- Use of CHR\$ to put characters on the screen
- Combining several graphics characters
- Using string variables to store combined characters
- Making a group of characters "flash" on the screen

As you begin to develop your own programs and applications, you will notice more and more opportunities to add graphical displays to your routines. Graphics sometimes enhance and simplify your output displays in ways that pure typed text cannot do. In developing new programs, experiment with your TRS-80 graphics characters. They are one of the more powerful features of your computer.

What we will cover now is how to make your "spaceship" *warp* off the edge of the monitor. Are you ready? Yes! Blast off!

11. To make the "spaceship" move, we need to alter the program so that the illusion of motion is produced by repeated printing and erasing of the screen. Two minor changes to the existing program produce this result. Here is the entire program with the two altered lines:

```

5 REM***SPACESHIP DISPLAY***
10 CLS
20 FOR I = 178 TO 182
30 AS = AS + CHR$(I)
40 NEXT I
50 FOR J = 0 TO 35 STEP 5 : PRINT @ 470 + J, AS
60 GOSUB 100
70 CLS : GOSUB 100
80 NEXT J : GOTO 50
100 FOR I = 1 TO 200 : NEXT : RETURN
    
```


Line 50 was changed to include the beginning of a loop whose loop counter is used in the PRINT @ statement to "move" the A\$ string across the screen. The first time through the loop, the string is printed at block location 470; the second time, at 475; and so on until the "ship" reaches the edge of the monitor. Line 80 was changed to contain the NEXT statement for this loop. Program control is then returned back to line 50.

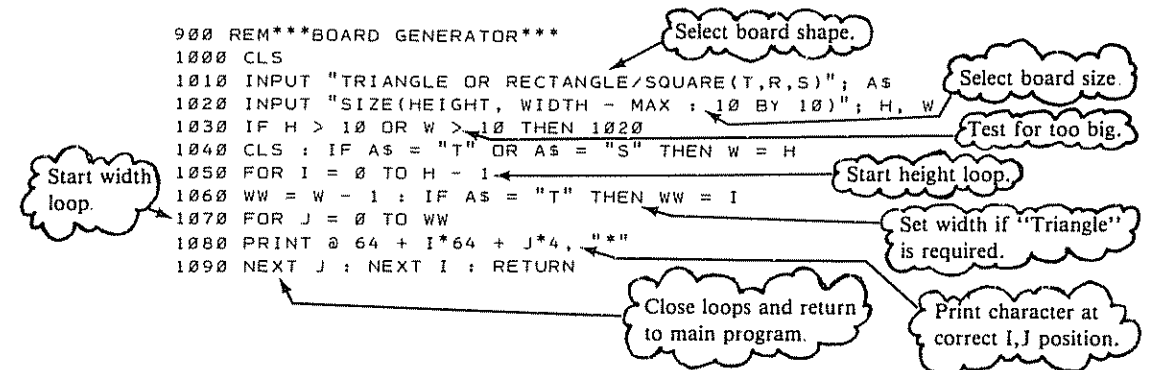
When this program is RUN, the "ship" moves to the edge of the screen, winks out of existence, and then appears back at the center to begin the journey again. Could Mr. Spock do any better?

It's time for you to create other graphics characters and experiment with making them flash and move about on the screen. How would you get the "ship" to warp off the screen at an angle—not just straight up or side to side? How would you make the "ship" vibrate and "glimmer" before it winks off the screen or appears back at the center? The effect is something like the image on the TV screen when the "transporter" beam on *Star Trek* is active. This effect is possible to reproduce on your TRS-80. Give it a try!

12. With the information on graphics behind you, it is now time to get "serious" about games. In the next few frames, we will develop a motor skill game, one that is good for smaller children. We will unfold the game in modules, doing a piece of the overall game at a time. We want to give you a feel for how you might go about producing your own programs.

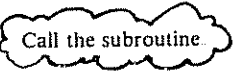
Since many games use a board, let's start by looking at a routine that prints a board (or many boards) on the screen. The concept we use in this module can be modified to produce nearly any kind of board. Our boards will have a simple structure, and will use only the graphics character asterisk (*) to represent board positions.

Enter and examine the following routine:



What? One routine to set up any sized triangular, rectangular, or square board? Yes, that's what this routine is designed to do. Notice that we have constructed the program as a subroutine. In this way, we can use the routine in the development of our game, but we can first test it without the entire game structure being around. Let's do just that! Enter the following small main program and type RUN:

```
5 REM***MAIN PROGRAM***
10 GOSUB 1000 : END
```



13. When the program to display boards is RUN, the routine clears the screen and requests an input for the type of board to be displayed— triangular (T), rectangular (R), or square (S). This activity occurs in lines 1000 and 1010.

Next, a board size is requested (line 1020). The dimensions of the board are stored in variables H and W. Line 1030 then checks these two variables against the maximum sizes allowed. For this routine, the maximum size for any board is 10 by 10.

Line 1040 clears the screen again, and sets W to H when the board is a triangle or a square. Line 1050 starts the height loop—the loop that controls how high the board is to be.

Line 1060 sets the loop counter for the width loop upper limit. WW, the width loop upper limit, is set to $W - 1$ for rectangles and squares. For triangles (when $A\$ = "T"$), the upper limit for each line of the board varies. Since the upper limit varies directly with the height counter, WW is set to I each time. For triangles, if W and H are not the same value (2×2 , or 3×3 , etc.), the size of the triangle will always be $H \times H$.

The width loop starts in line 1070.

The board *positions* are displayed by line 1080. Each *position* is marked by an asterisk (*). The block number where the asterisk is displayed is computed. The first asterisk is placed in block 64. The rest of the asterisks on the first line of the board are placed at 68, 72, etc. This positioning is controlled by the addition of $J*4$ to the initial starting block position (64). The addition of $I*64$ to the starting block position controls which line is being printed. When $I = 0$, the first line is being printed; when $I = 1$, the second line; and so on for each line needed.

Line 1090 contains the end points for both height and width loops, and the RETURN statement to get you back to the main program.

If we RUN the program, ask for a triangular board (T) of size 5×5 , the screen shows:

```
*
* *
* * *
* * * *
* * * * *
READY
>_
```

Try several board shapes and sizes on your own. Keep experimenting with the program until you get bored with boards.

14. Ready for the next routine? How about a subroutine that lets you “drive” around the boards produced by the last routine? That would be a nice tool to have in your “library.” But how can that be done?

Well, it will involve using several TRS-80 features such as the INKEY\$ function and the ASC function. We will also use the graphics technique of “flashing” that we talked about earlier in this chapter. Also, we will make the routine smart enough so that it works on rectangles, squares, or triangles. Quite a deal! But, it does require some typing. Once this routine is added to what you already have in memory (the board display subroutine), you may want to save everything on cassette tape or disk. Here is the new routine:

```

1900 REM***FLASH AND MOVE ROUTINE***
2000 I = H - 1 : J = W - 1 : B$ = " "
2010 KK = 64 + 64*I + 4*J : PRINT @KK, " " ;
2020 GOSUB 3000 : PRINT @KK, "v";
2030 GOSUB 3000
2040 C$ = INKEY$ : IF C$ <> " " THEN B$ = C$
2050 II = I : JJ = J
2060 IF ASC(B$) = 91 THEN II = II - 1
2070 IF ASC(B$) = 10 THEN II = II + 1
2080 IF ASC(B$) <> 91 AND ASC(B$) <> 10 THEN 2120
2090 IF II < 0 OR II > H - 1 THEN 2010
2100 IF AS = "T" AND II < JJ THEN 2010
2110 PRINT @KK, "*"; : I = II : GOTO 2010
2120 IF ASC(B$) = 8 THEN JJ = JJ - 1
2130 IF ASC(B$) = 9 THEN JJ = JJ + 1
2140 IF JJ < 0 OR JJ > W - 1 THEN 2010
2150 IF AS = "T" AND JJ > II THEN 2010
2160 PRINT @KK, "*"; : J = JJ : GOTO 2010
3000 FOR TT = 1 TO 100 : NEXT : RETURN

```

When you have entered this routine, change line 10 of your program to:

```
10 GOSUB 1000 : GOSUB 2000
```

Go ahead and RUN this program. You will be asked for a board shape and size. Then you will see the board appear, and the letter Y will start flashing in the lower corner of the board. At this point, if you press the following keys, the letter Y will move about the board:

- ↑ Moves the letter *up* the board.
- ↓ Moves the letter *down* the board.
- ← Moves the letter to the *left*.
- Moves the letter to the *right*.

Note: The program will not allow you to move “off” the board. Also, once you have started the letter moving in a particular direction, it continues to move that way until it either hits the edge of the board or you change its direction.

Experiment with different board shapes and sizes. The next frame discusses what each line in this subroutine does.

15. Let’s examine what each line in the last subroutine (frame 14) is doing.

—Line 2000: Saves the dimensions of the board in I and J. Sets B\$ to a blank (“ ”) to start.

—Line 2010: Computes the block location of the letter Y. The letter starts in the

lower corner of the board. To begin the flashing of the character, a blank is printed at that location.

- Line 2020: Call on the delay routine (line 3000). Then the letter Y is printed.
- Line 2030: Call on the delay routine again. This operation completes the flashing operation.
- Line 2040: Use the INKEY\$ function to strobe (read) the keyboard. If a key has been pressed, put the keystroke into B\$. If no key was depressed, B\$ remains whatever it was from the last time through the loop.
- Line 2050: Set temporary location variables II and JJ to the current position of the letter.
- Line 2060: Test for B\$ = ↑ . If B\$ is an ↑ , ASC(B\$) = 91. A move up the board is indicated, and the temporary position counter is decreased by one.
- Line 2070: Test for B\$ = ↓ . If B\$ is an ↓ , ASC(B\$) = 10. A move down the board is indicated. The position counter is increased by one.
- Line 2080: If B\$ is not an ↑ or ↓ , then go check for moves to the right and left.
- Line 2090: Check to see if move is off the board (top, bottom).
- Line 2100: For triangles, check to see if off board (top, bottom).
- Line 2110: Move is good. Put an asterisk in the current block location (where the letter is). Set the height indicator, I, to the new position, II. Go back to the flashing part of the routine.
- Lines 2120–2160: Repeat the same kind of tests for left and right movements on the board.
- Line 3000: The delay routine.

Short, but very powerful! Now, how can we make a game out of this? Let's see, we can move about the board at will. What if we were searching as we moved, or trying to catch something? That's it! Let's pretend there is a dragon on the board, and that we are to catch the dragon. Look to the next frame for how we might do this. What does happen when you catch a dragon?

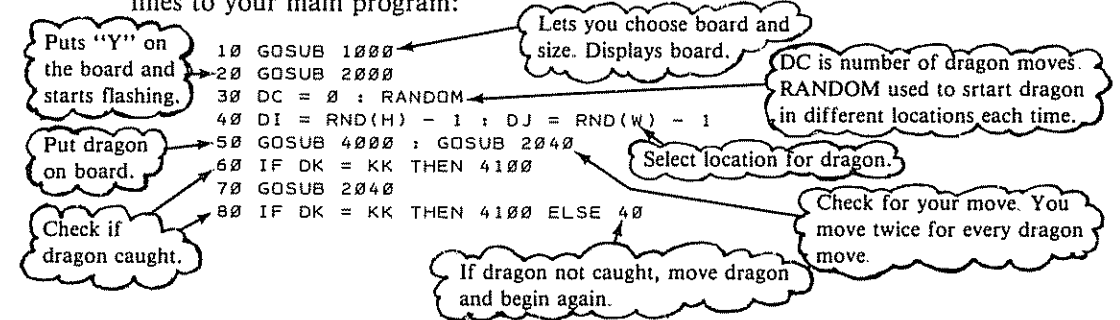
16. To add a dragon to our game, we enter the following lines of BASIC. Our dragon has some unusual properties. First, the dragon hops around the board randomly. He never seems to stay in one place very long. Also, every once in a while, he jumps right into your arms. So, if you are patient, you will always catch the dragon, even sitting still. When you catch the dragon, you get a magic spell of your choice.

```

3900 REM***DRAGON SECTION***
4000 IF A$ = "T" AND DI < DJ THEN RETURN
4010 IF DC <> 0 THEN PRINT @DK, "**";
4020 DK = 64 + 64*DI + 4*Dj
4030 DC = DC + 1
4040 PRINT @ DK, "D";
4050 RETURN
4100 PRINT @ 896, "YOU CAPTURED THE DRAGON IN"; DC;
4110 PRINT "DRAGON MOVES"
4120 PRINT "CONGRATULATIONS - YOU GET A MAGIC SPELL OF YOUR
CHOICE!"
4130 FOR TT = 1 TO 2000 : NEXT
4140 CLS : GOSUB 1040 : GOTO 20

```

Enter the lines shown above into your computer, and add the following lines to your main program:



Also, change line 2000 to read:

```
2000 GOSUB 3000 : RETURN
```

Enter the new lines, and RUN the program. Try catching the dragon on different boards of various sizes. The next frame will discuss what each of these new lines of program does. Have fun catching the dragon!

17. The subroutine that puts the dragon on the board does the following:

- Line 4000: Checks for a valid location on a triangular board.
- Line 4010: If the dragon has moved once, an asterisk (*) is placed where the dragon currently sits. This line is not executed the first time through the program ($DC = 0$).
- Line 4020: Computes the block location for the dragon.
- Line 4030: The dragon-move-counter is increased by one.
- Line 4040: Puts the dragon on the board, in the new location.
- Line 4050: Returns to calling program.

The end-of-the-game messages are in lines 4100–4130. Line 4140 is a delay so that the messages can be read. Line 4150 clears the screen, sets up the same board again (GOSUB 1040), and then returns to the main program at the point where the game is begun.

The main program module does the following:

- Line 10: Calls on routine that lets you choose the board and the size of the board, and then displays the board on the screen.
- Line 20: Calls on that part of subroutine 2000 that sets up the “flashing” of the letter Y.
- Line 30: Initializes the dragon-move-counter, DC, and calls on the RANDOM function so that the placing of the dragon is different each time you start the game.
- Line 40: Selects a random location for the dragon.
- Line 50: Calls routine that puts the dragon on the board (subroutine 4000), and then calls on the routine that allows you to make your moves (subroutine 2040). *Note:* With this last GOSUB to 2040, we enter the routine that was called in line 20 at a different point.
- Line 60: Checks to see if the position of the dragon, DK, and your position, KK, are the same. If they are, the game is over. The program branches to line 4100 for the end-game messages.
- Line 70: Calls on routine that allows you to make your moves. *Note:* This is a second time we go to this routine. You get two moves for every one move made by the dragon. It’s only fair! The dragon can fly all over the place.
- Line 80: Again, a check for an end-game condition is made. If the dragon is not caught, a branch back to line 40 is made. At line 40, a new dragon location is computed, and the game continues.

SOME IMPORTANT MESSAGES!!

- If you have not saved this program on tape, do so now. We will be moving on to other programs. You don’t want to lose this one, if you have spent time typing it into the TRS-80. (Refer to your *User’s Reference Manual* or Appendix B for instructions on saving programs on tape.)
 - This game can be changed and enhanced in many ways. You could make the dragon move about differently. You could slow him down; make him move faster. Try to make one of these changes yourself. Can you think of other changes or enhancements that could be made?
 - We have built up an entire game from separate modules. Building programs in this manner is usually a good way to test out the individual sections of the game or application. It also tends to reduce errors. We *were* careful, though, to avoid variable name conflicts among all the various routines.
 - The next frame gives the entire listing of the “Dragon Capture” program. This is for your easy reference should you want to reenter or change this program at a later date.
-

18. Here is a listing of the "Dragon Capture" program:

```

5 REM***MAIN PROGRAM - DRAGON CAPTURE***
10 GOSUB 1000
20 GOSUB 2000
30 DC = 0: RANDOM
40 DI = RND(H)-1: DJ = RND(W)-1
50 GOSUB 4000: GOSUB 2040
60 IF DK = KK THEN 4100
70 GOSUB 2040
80 IF DK = KK THEN 4100 ELSE 40

900 REM***BOARD GENERATOR***
1000 CLS
1010 INPUT "TRIANGLE OR RECTANGLE/SQUARE(T, R, S)"; AS
1020 INPUT "SIZE(HEIGHT, WIDTH - MAX: 10 BY 10)"; H,W
1030 IF H > 10 OR W > 10 THEN 1020
1040 CLS : IF AS = "T" OR AS = "S" THEN W = H
1050 FOR I = 0 TO H-1
1060 WW = W - 1: IF AS = "T" THEN WW = I
1070 FOR J = 0 TO WW
1080 PRINT @ 64 + I*64 + J*4, "*"
1090 NEXT I : NEXT J : RETURN

1900 REM***FLASH AND MOVE ROUTINE***
2000 I = H - 1: J = W - 1: BS = " "
2010 KK = 64 + 64*I + 4*J : PRINT @ KK, " ";
2020 GOSUB 3000 : PRINT @ KK, "Y";
2030 GOSUB 3000 : RETURN
2040 CS = INKEY$ : IF CS <> " " THEN BS = CS
2050 II = I : JJ = J
2060 IF ASC(BS) = 91 THEN II = II - 1
2070 IF ASC(BS) = 10 THEN II = II + 1
2080 IF ASC(BS) <> 91 AND ASC(BS) <> 10 THEN 2120
2090 IF II < 0 OR II > H - 1 THEN 2010
2100 IF AS = "T" AND II < JJ THEN 2010
2110 PRINT @ KK, "*";: I = II: GOTO 2010
2120 IF ASC(BS) = 8 THEN JJ = JJ - 1
2130 IF ASC(BS) = 9 THEN JJ = JJ + 1
2140 IF JJ < 0 OR JJ > W - 1 THEN 2010
2150 IF AS = "T" AND JJ > II THEN 2010
2160 PRINT @ KK, "*";: J = JJ: GOTO 2010
3000 FOR TT = 1 TO 100 : NEXT : RETURN

3900 REM***DRAGON SECTION***
4000 IF AS = "T" AND DI < DJ THEN RETURN
4010 IF DC <> 0 THEN PRINT @ DK, "*";
4020 DK = 64 + 64*DI + 4*DJ
4030 DC = DC + 1
4040 PRINT @ DK, "D";
4050 RETURN

4100 PRINT@896,"YOU CAPTURED THE DRAGON IN";DC;
4110 PRINT " DRAGON MOVES"
4120 PRINT "CONGRATULATIONS - YOU GET A MAGIC SPELL OF YOUR
CHOICE"
4130 FOR TT = 1 TO 2000 : NEXT
4140 CLS : GOSUB 1040 : GOTO 20

```

MAIN PROGRAM

BOARD SELECTION/
DISPLAY SUBROUTINE

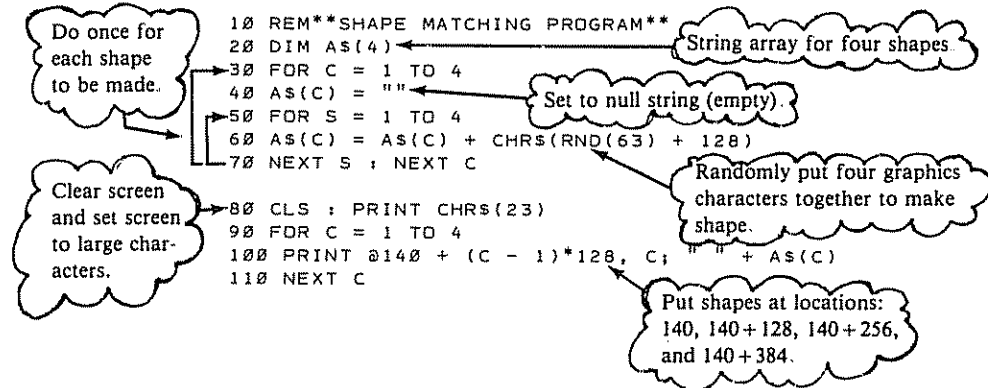
LETTER Y MOVE
SUBROUTINE

DRAGON ROUTINE

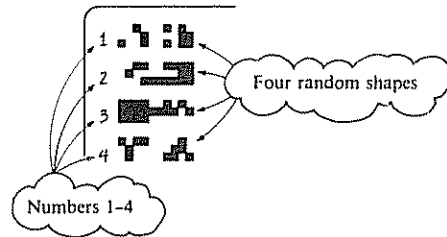
END-GAME MESSAGE AND
BOARD REDISPLAY ROUTINE

19. Let's construct another game in a modular fashion. This program, designed for small children, is a shape-matching exercise. The child is presented with four arbitrary "shapes" and a single shape selected at random from the four. The program asks the child to match the single form to one of the four original shapes. When the child responds, the single shape moves up the screen and is placed next to the matching form. The single shape then "flashes" several times, and the program asks if the user wants a new set of shapes. The program combines several techniques that you have used earlier in this chapter.

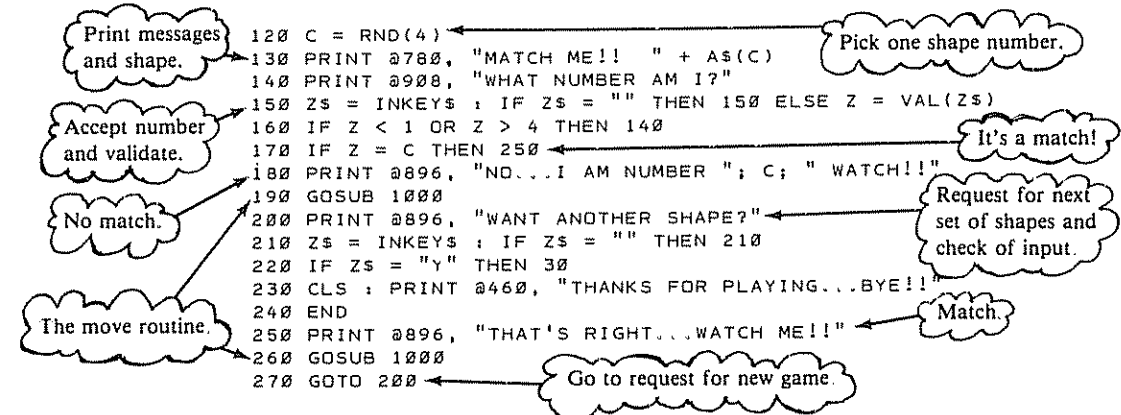
First, we construct the portion of the program that generates and displays the four shapes. The shapes are generated randomly!



The graphics codes run from 129 through 191. The argument to CHR\$ in line 60, RND(63) + 128, randomly selects one of these codes. The graphics character for the selected code is then catenated (attached) to three other characters and placed in one of the A\$-array positions. The PRINT CHR\$(23) in line 80 sets the screen so that all display is in the larger character format (32 characters wide). Enter and RUN this part of the program. The screen will show something like the following (Remember! The shapes change each time.):



20. Now we can add the part of the program that selects the single shape to be matched, displays that shape, asks for the number of the matching shape, and checks for a match. Here goes!

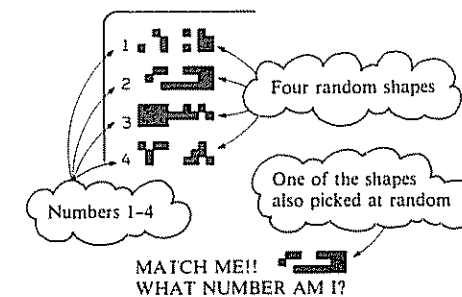


In line 150, the function VAL is used to convert the key that is pressed into a numeric value. If 1 to 9 is pressed, Z is assigned the appropriate value from VAL. Any other key produces a zero. A check is made in line 160 for the keys 1 to 4. If some key other than 1 to 4 is pressed, the request is repeated.

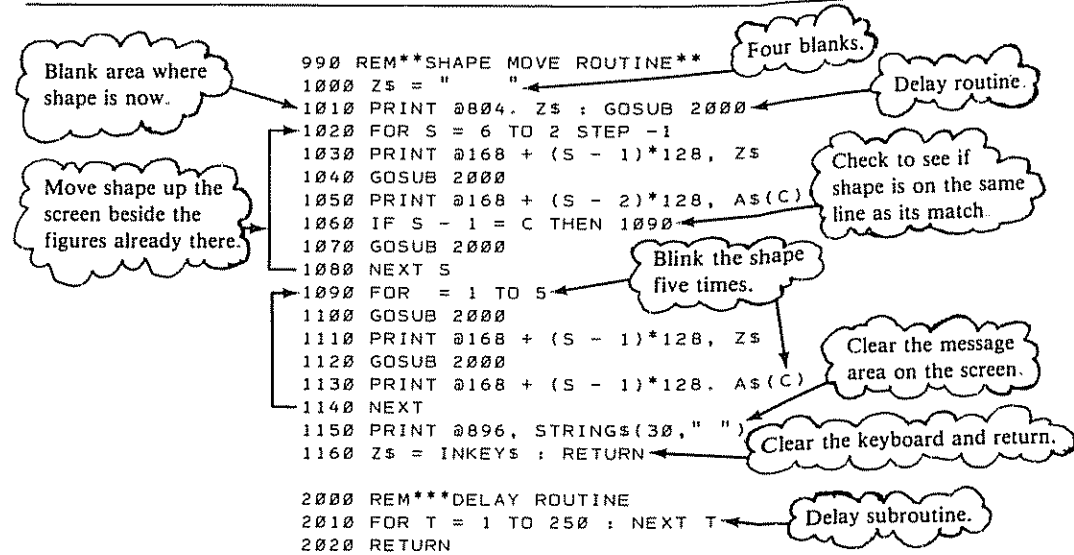
The program needs the subroutine beginning at line 1000 before the last part can be checked out. However, if you enter the following line, you can run the program:

```
1000 FOR T = 1 TO 300 : NEXT : RETURN
```

This just puts a short delay in the program so the screen can be observed. RUN the program. The screen should look something like this:



21. In the last set of statements, an appropriate message is displayed based on whether there is or is not a match. Then a call is made on the routine that moves the shape up the screen. The following is that move routine:



Enter this part of the program and RUN the game. The screen will show the four random forms and messages as before. Now, when you enter a number, the program prints the message of whether you guessed right or not, and the shape then "moves" up the screen. It will move until it is beside its matching shape. Once there, it blinks five times. The program then requests a response from you as to whether you wish another game. Try it!!

22. A complete listing of the last program follows. First, though, let's look at some of the things in the routine that moved the shape.

Each shape is made up of four graphics characters. To "animate" a character and make it "move," we have to blank the area where it currently sits, and then display the character farther up the screen. The variable Z\$ is filled with four blanks. It is printed when we wish to blank the shape. Lines 1010, 1030, and 1110 clear the area where the shape resides. Lines 1050 and 1130 redisplay the character in the new positions.

The first loop, from 1020 to 1080, moves the symbol up the screen. There is a check at line 1060 to stop the loop when the shape is beside its matching symbol. The second loop, lines 1090 to 1140, flashes the character once it is in place beside its matching symbol.

Line 1150 clears the message area below the MATCH ME!! line. Line 1160 clears the keyboard, in case several keys were pressed by the user, and returns to the main program.

Can you think of any interesting variations of this game? What if you placed the symbols not in a column but anywhere on the screen? Then the child would have to search for the character a bit more. Also, you could replace the numbers with random letters. That would teach recognition of the keys on the keyboard. How about changing the program to give the child another chance if a match is not made the first time? There are many possibilities for this program. See if you can come up with more.

The principal use of this program might be for you to borrow ideas on the graphics techniques for your own programs. In any case, have fun with the program.

The next frame contains a complete listing of the program.

23. Listing of the "Shape Matching" program:

```

10 REM**SHAPE MATCHING PROGRAM**
20 DIM A$(4)
30 FOR C = 1 TO 4
40 A$(C) = ""
50 FOR S = 1 TO 4
60 A$(C) = A$(C) + CHR$(RND(63) + 128)
70 NEXT S : NEXT C

80 CLS : PRINT CHR$(23)
90 FOR C = 1 TO 4
100 PRINT @140 + (C - 1)*128, C; " " + A$(C)
110 NEXT C

120 C = RND(4)
130 PRINT @780, "MATCH ME!! " + A$(C)
140 PRINT @908, "WHAT NUMBER AM I?"
150 Z$ = INKEY$ : IF Z$ = "" THEN 150 ELSE Z = VAL(Z$)
160 IF Z < 1 OR Z > 4 THEN 140
170 IF Z = C THEN 250
180 PRINT @896, "NO...I AM NUMBER "; C; " WATCH!!"
190 GOSUB 1000
200 PRINT @896, "WANT ANOTHER SHAPE?"
210 Z$ = INKEY$ : IF Z$ = "" THEN 210
220 IF Z$ = "Y" THEN 30
230 CLS : PRINT @460, "THANKS FOR PLAYING...BYE!!"
240 END
250 PRINT @896, "THAT'S RIGHT...WATCH ME!!"
260 GOSUB 1000
270 GOTO 200

990 REM***SHAPE MOVE ROUTINE
1000 Z$ = " "
1010 PRINT @804, Z$ : GOSUB 2000
1020 FOR S = 6 TO 2 STEP -1
1030 PRINT @168 + (S - 1)*128, Z$
1040 GOSUB 2000
1050 PRINT @168 + (S - 2)*128, A$(C)
1060 IF S - 1 = C THEN 1090
1070 GOSUB 2000

1090 FOR S = 1 TO 5
1100 GOSUB 2000
1110 PRINT @168 + (S - 1)*128, Z$
1120 GOSUB 2000
1130 PRINT @168 + (S - 1)*128, A$(C)
1140 NEXT S
1150 PRINT @896, STRING$(30, " ")
1160 Z$ = INKEY$ : RETURN

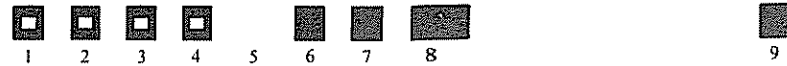
2000 REM***DELAY ROUTINE
2010 FOR T = 1 TO 250 : NEXT T
2020 RETURN

```

24. The next program is an ancient and classic game of logic. The game is challenging and not that easy to complete in the minimum number of moves. The game is called "Frogs."*

"Frogs" is a game in which two sets of four objects are placed on a nine-position board. One set is placed to the left, the other to the right. This positioning leaves a vacant spot in the center. The game consists of completely reversing the two sets of objects. The only legal moves are either a slide of one object to an empty position or a jump of one object over another object to a vacant place.

The game looks like this at the start:



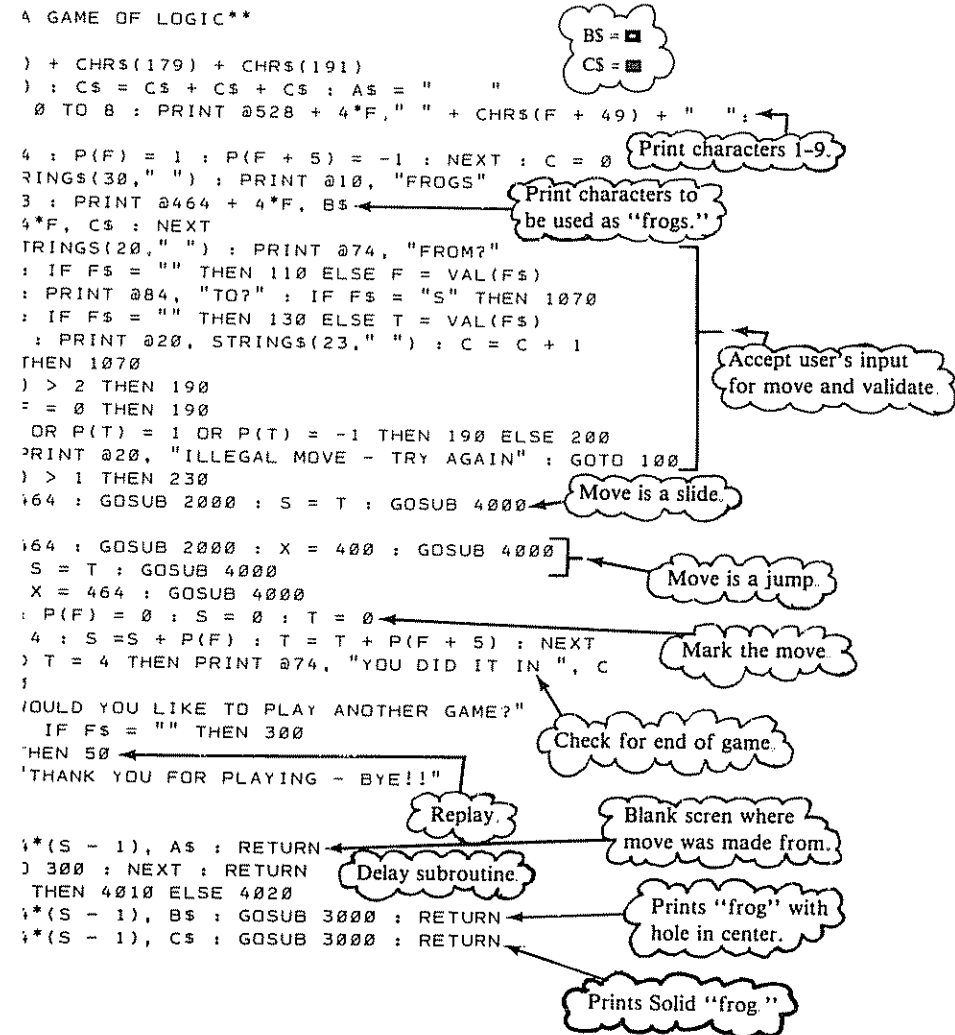
The only legal moves, at the beginning, are: a slide from 4 to 5; a slide from 6 to 5; a jump from 3 to 5; a jump from 7 to 5. The object of the game is to have the pieces end up this way:



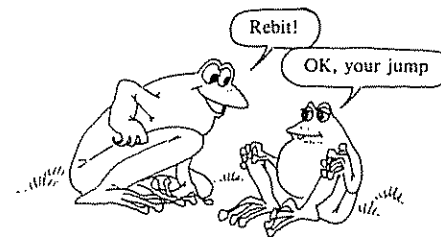
You might want to lay some objects on this page, take the book and try the game by hand. You can use pennies (heads and tails), pennies and dimes, buttons in two colors, or whatever you have handy. By the way, it is possible to completely reverse the objects in 24 moves!!

*The game was presented as a computer program by Mac Ogelsby in the People's Computer Company newspaper, Nov.-Dec. 1976 issue. The program in its present form appeared in *Recreational Computing* magazine, Vol. 7, No. 3, Nov.-Dec. 1978.

is a complete listing of "Frogs." Enter and RUN the program.



When you run the program, the screen shows the "frog" symbols like those in the program. The program requests two numbers: where you are moving from, and where you are moving to. There is no need to press the ENTER key. The RETURN key is used to scan the keyboard and pick up your entries.



Go ahead and try the game! The next frame provides some comments on what is happening within the program. You can look at that information when you wish. Can you do the "Frog" game in 24 moves?

26. The following comments apply to the program in frame 25, the "Frogs" game. This program has used nothing new to you. It just provides another example of how you can incorporate the graphics capabilities of your TRS-80 in your own applications.

- Line 20: The array P is used to mark the positions of the "frog" symbols on the screen.
- Lines 30-40: The "frog" symbols are constructed.
- Line 50: This line clears the screen, and places the numbers 1-9 on the screen. CHR\$(49) is a zero, CHR\$(50) is a one, and so on.
- Line 60: The starting positions of the game are marked. Symbols with the holes are given the value 1 in the P array; solid symbols the value -1. A zero in the P array indicates an empty board position. The variable C is the move counter.
- Lines 70-90: Places the "frog" symbols on the screen.
- Lines 100-190: This part of the program handles the input of the player's moves. If the letter S is pressed, the game ends. The check at line 160 is for an attempt to jump more than two spaces on the board. Line 180 checks for a jump to the piece's present place and for jumps to occupied positions.
- Line 200: Check for a jump or slide.
- Line 210: Move is a slide. The parameters S and X provide information on a piece's position. S is set to either F (*from* position) or T (*to* position). X is set to either 464 (the line on the screen of the board) or 400 (the line above the board). The last setting is for a jump. The routine at 2000 erases a symbol from the screen. The routine at 4000 displays the symbols.
- Lines 230-250: Move is a jump.
- Line 260: The P array is marked with the value of the moved piece. S and T are set to zero for use in next line.
- Line 270: Sums of the left and right place markers are computed.
- Line 280: If S is -4 and T is 4, the game is complete.
- Line 2000: Erases the symbol from the current position.
- Line 3000: Delay routine for slowing down jumps.
- Line 4000: Check for solid symbol or symbol with hole being moved.
- Line 4010: Prints the symbol with the hole in the center.
- Line 4020: Prints the solid symbol.

What would be some interesting changes to this program? Why not redesign the "frog" symbols so that they look like frogs? You could even add legs that move during jumps.

Well, enough games!! Time to get to the "serious" stuff. Read on!!

27. There are many possible "serious" applications that you can program on your TRS-80. We will demonstrate a couple of routines that you might find useful around the home. The emphasis in presenting the programs is still on the techniques of the BASIC programming language. So, even if you don't find the applications of interest, you may still be able to use some of the procedures in your own programming tasks.

We start with a program that allows you to search a file of data on names, addresses, and telephone numbers, and extract the records that match a search key that you specify.

The file, in this case, is contained in DATA statements within the program. You could generalize this routine to a larger data base that is perhaps stored on tape or disk. However, we will demonstrate the program using only DATA statements.

In fact, to keep things simple, we will use only two data records. The records will be the names, addresses, and telephone numbers for two people:

Jack Jones 1511 Westport Dr. Menlo Park, CA 94025 415/555-0707
Jane Jonarts 1521 Westbury St. Menlo Pk., Calif. 94025 415/555-1975

Adding more data records will only complicate the running and testing of the program. We have deliberately chosen two data records that are similar in many respects so that we can demonstrate the search capabilities of the program.

The program that we will design will search each data record by: first name, last name, street address, street name, city, state, zip code, or area code. The search key can be from one to several letters long. If an exact match is needed, you can use more letters. If fewer letters are used, all records that match the search key are displayed. Sounds like a lot, doesn't it? The program is actually quite simple. Move on to the next frame to see what it looks like.

28. Here is the program for the computerized directory search:

```

10 REM**COMPUTERIZED DIRECTORY SEARCH**
20 DIM N1$(10),N2$(10),A1$(10),A2$(10),C$(10),S$(10),Z$(10),T$(10)
30 CLS : INPUT "NUMBER OF DATA ITEMS: "; D : GOSUB 1000
40 CLS : PRINT "COMPUTERIZED DIRECTORY"
50 PRINT STRING$(22,"-")
60 PRINT @256, "WHAT DO YOU WANT TO DO? SEARCH BY:"
70 PRINT @320, " 1- FIRST NAME"
80 PRINT @384, " 2- LAST NAME"
90 PRINT @448, " 3- STREET ADDRESS"
100 PRINT @512, " 4- STREET NAME"
110 PRINT @576, " 5- CITY"
120 PRINT @640, " 6- STATE"
130 PRINT @704, " 7- ZIP CODE"
140 PRINT @768, " 8- AREA CODE"
150 PRINT @832, " 9- STOP THE PROGRAM"
160 K$ = INKEY$ : IF K$ = "" THEN 160 ELSE K = VAL(K$)
170 IF K = 9 THEN END
180 IF K = 0 THEN 160
190 CLS : INPUT "ENTER THE SEARCH KEY: "; K$ : P = 0

```

```

200 FOR N = 1 TO D : ON K GOTO 210,220,230,240,250,260,270,280
210 IF LEFT$(N1$(N),LEN(K$)) = K$ THEN 290 ELSE 320
220 IF LEFT$(N2$(N),LEN(K$)) = K$ THEN 290 ELSE 320
230 IF LEFT$(A1$(N),LEN(K$)) = K$ THEN 290 ELSE 320
240 IF LEFT$(A2$(N),LEN(K$)) = K$ THEN 290 ELSE 320
250 IF LEFT$(C$(N),LEN(K$)) = K$ THEN 290 ELSE 320
260 IF LEFT$(S$(N),LEN(K$)) = K$ THEN 290 ELSE 320
270 IF LEFT$(Z$(N),LEN(K$)) = K$ THEN 290 ELSE 320
280 IF LEFT$(T$(N),3) = K$ THEN 290 ELSE 320
290 PRINT @256 + P*64,N1$(N) + " " + N2$(N); TAB(25); A1$(N) + " " + A2$(N)
300 PRINT @256 + (P + 1)*64,C$(N) + " " + S$(N) + " " + Z$(N); TAB(40); T$(N)
310 P = P + 2
320 NEXT N
330 PRINT @780, "HIT ANY KEY TO CONTINUE:"
340 K$ = INKEY$ : IF K$ = "" THEN 340 ELSE 40

1000 REM**DATA INPUT ROUTINE**
1010 FOR N = 1 TO D
1020 READ N1$(N),N2$(N),A1$(N),A2$(N),C$(N),S$(N),Z$(N),T$(N)
1030 NEXT N : RETURN
1040 DATA "JACK","JONES","1511","WESTPORT DR.,""MENLO PARK",
"CA","94025","415/555-0707"
1050 DATA "JANE","JONARTS","1521","WESTBURY ST.,""MENLO PK.,"
"CALIF","415/555-1975"

```

Loop through data file and branch to specific search item.

Test for match with search key.

Display data record.

Increment the screen position index.

Pause to view output.

Loop to read data records.

Data records.

Enter and RUN the program. What appears on the screen? Look at the next frame for the answer!

29. When the program in the last frame is RUN, the screen clears and the request for the number of data items appears. If you enter a 2, the two data records in lines 1040 and 1050 are read into the appropriate arrays. The screen then clears again and the "menu" of selections appears. The screen shows:

"menu"

```

WHAT DO YOU WANT TO DO? SEARCH BY:
1- FIRST NAME
2- LAST NAME
3- STREET ADDRESS
4- STREET NAME
5- CITY
6- STATE
7- ZIP CODE
8- AREA CODE
9- STOP THE PROGRAM

```

The program then waits for you to enter a number from 1 to 9. If you enter 9, line 170 causes the program to end. If you enter a number in the range 1 to 8, the ON K GOTO statement in line 200 will cause one, and only one, of lines 210 to 280 to be executed.

```

200 FOR N = 1 TO D : ON K GOTO 210,220,230,240,250,260,270,280

```


The ON K GOTO statement in line 210 tells the computer:

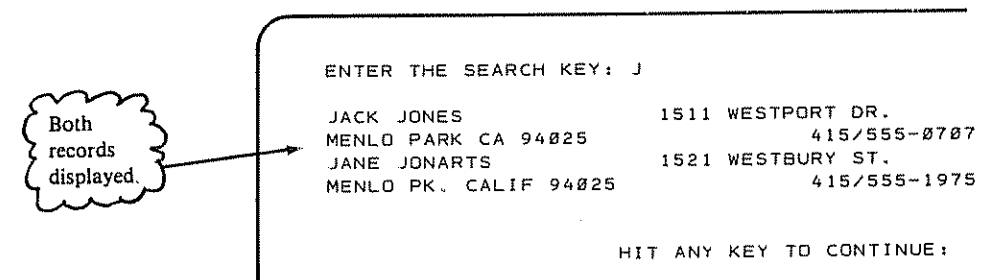
If K is 1, then GOTO 210
 If K is 2, then GOTO 220
 If K is 3, then GOTO 230
 and so on

K = 1 K = 2 K = 3 K = 4 K = 5 K = 6 K = 7 K = 8

ON K GOTO 210, 220, 230, 240, 250, 260, 270, 280

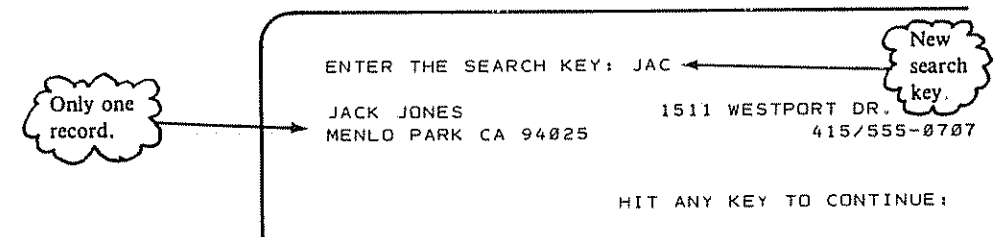
When you do enter a number, the program then requests a search key. For menu items 1-7, the key can be from 1 to as many characters as you wish to enter. For item 8, you must enter three area code digits.

Let's say we elect menu item 1, a search by first name, and enter the letter J for the search key. The screen will show:

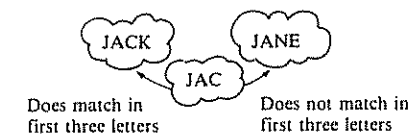


Pressing any key will cause the "menu" to reappear. What would appear on the screen if a search by first name were made with the key JAC?

30. If menu selection 1, a search by first name, and the key JAC is entered, the screen shows:



Why does this occur? Well, the search key is three letters long, and only matches exactly with the first three letters of Mr. Jones' first name. The key does not match Ms. Jonarts' first name in the first three positions. So only the record for Mr. Jones is displayed.



If the search key had been JA, both records would have been put on the screen. Let's examine line 210 of the program where this matching operation is taking place.

```
210 IF LEFT$(N1$(N),LEN(K$)) = K$ THEN 290 ELSE 320
```

The search key is in K\$. N1\$(N) contains the first name of the Nth record being examined. The function LEFT\$ extracts the leftmost characters from N1\$(N) based on the length, LEN, of K\$. That is, the number of characters looked at in N1\$(N) is determined by the length of the search key.

All the other searches are carried out in the same way in lines 220–270. The search for the area code in line 280 is the only different kind of match. This last match is made on exactly the first three characters of the telephone number field, T\$(N)

Try the various “menu” selections, and use different search keys. Notice how quickly the TRS-80 is able to scan the records and display the results of the search.

Can you think of a way to *really* simplify the program? *Hint:* Change the arrays that are being used to one array!! If one data array, say D\$, is used with dimensions D\$(8,10), lines 210–270 reduce to a single line in the program:

```
IF LEFT$(D$(K,N),LEN(K$)) = K$ THEN 290 ELSE 320
```

Amazing!! The entire search-and-display section reduces to just a few lines of BASIC! What else could you do with this program? The techniques shown here would work on any data file. Instead of names and addresses, you might have a parts inventory or lists of business accounts.

But our *search* is over! Time to move on to another example.

31. Well, this ends the current chapter and the book. We trust you have had an enjoyable and rewarding time going through this Self-Teaching Guide. It has been our pleasure to accompany you on this adventure with your TRS-80 computer. We look forward to working with you again in our next book on the TRS-80 machine. In it, we will cover the many BASIC language elements that were beyond the scope of the present book—PEEK and POKE statements, the USR function, disk operations, creating and using large files, and much more.

Goodbye, for now, and happy computing!!

Appendixes

- APPENDIX A Setting Up Your TRS-80
- APPENDIX B The Cassette Recorder
- APPENDIX C Arithmetic
- APPENDIX D Error Messages
- APPENDIX E Print and Graphics Layout Sheets
- APPENDIX F Reserved Words
- APPENDIX G ASCII Codes

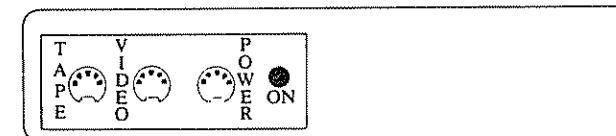
APPENDIX A

Setting Up Your TRS-80

The basic TRS-80 system consists of four components:

- Keyboard/computer
- Power supply
- Video monitor
- Cassette tape recorder

The last three of these units must be connected to the jacks on the rear of the keyboard unit. The correct unit must be connected to the corresponding jack. Each jack is labeled as shown in this sketch.

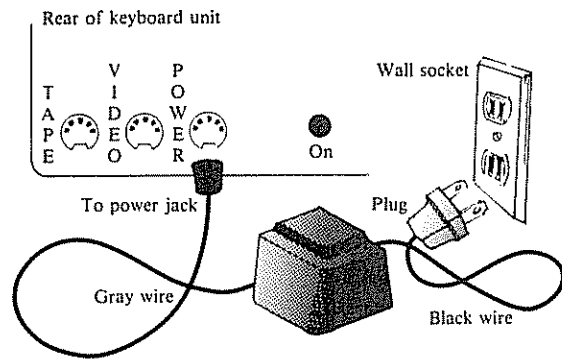


Rear View of the TRS-80 Keyboard Unit

The jacks are notched at the bottom so that the correct connections will be made to the five pins of each connector.

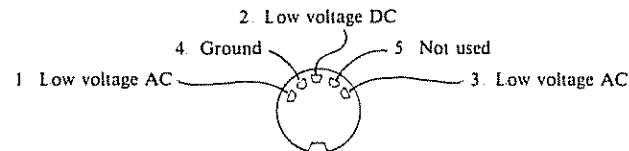
The Power Supply

Two wires come out of the power supply. The gray wire has a connector on the end that is plugged into the jack labeled POWER on the back of the keyboard unit.



Power Supply Connections

I'm sure you have all plugged in lamps, toasters, etc. to a wall socket. You should take special care when plugging in the power supply to the rear of your TRS-80 keyboard unit. The notch on the power connector helps in aligning the five pins. A sketch of the connections is shown below.



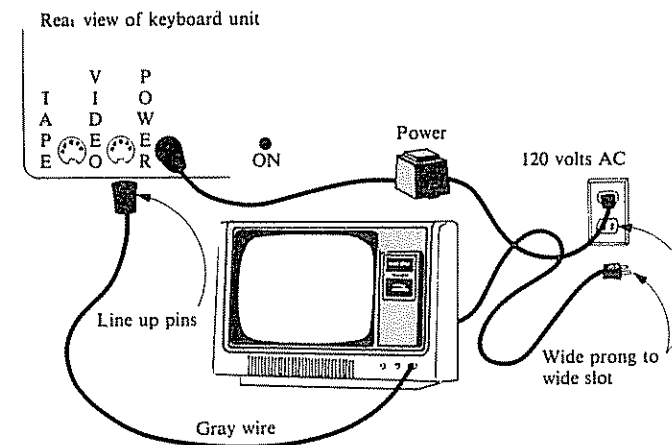
Keyboard Power Jack (as viewed from rear of keyboard)

The TRS-80 needs three voltage levels for its operation: +12 volts, +5 volts, and -5 volts. The +12 and -5 volts are needed for the Random Access Memory (RAM) and the +5 volts is needed almost everywhere. The power supply steps down the AC voltage to voltages that the computer can handle. Inside the power supply unit there is also a fuse connected to the transformer primary winding. This protects the computer from irregularities in power coming from the wall socket.

The most probable cause for a failure in the power supply unit is a blown fuse. Since the unit is sealed, the power supply should be taken to your Radio Shack store if any problems develop.

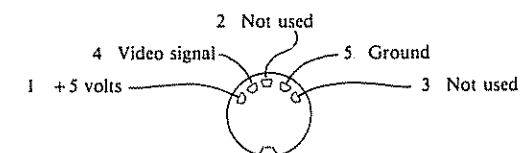
The Video Monitor

The video monitor also has two connectors. The gray wire coming from the front of the monitor is connected to the keyboard unit. The connection is made to the jack labeled VIDEO. The wire that goes to the AC socket has a plug with two prongs, one wider than the other. The wide prong should go to the widest slot of the AC socket. Do not use an extension cord for this connection. Use a direct connection to a wall socket if it is at all possible.



Video Monitor Connections

The gray wire that goes from the monitor to the rear of the keyboard unit also has five pins. The jack on the keyboard unit is shown below.



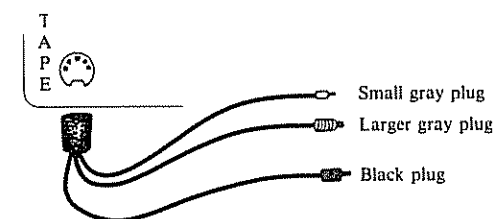
Video Jack (as viewed from rear of keyboard unit)

The Cassette Recorder

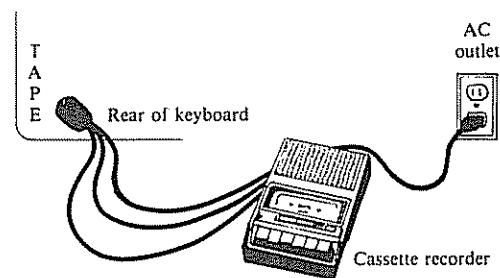
You do not need to connect the cassette recorder unless you are going to record programs or load previously recorded programs. However, if you *are* going to use the recorder, follow these instructions.

The recorder cable has one plug with a five-pin connector on one end, and three separate plugs on the other end.

1. Plug the end with the five-pin connector into the rear of the keyboard unit at the jack labeled TAPE.



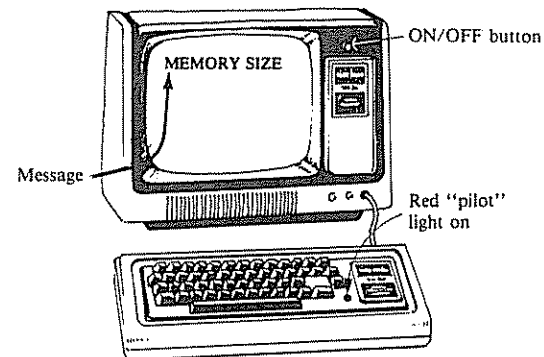
2. Connect the black plug on the other end of the recorder cable to the EAR jack of the tape recorder. This connection provides the output signal from the recorder to the TRS-80 for loading taped programs.
3. Connect the larger gray plug to the AUX jack of the tape recorder. This connection provides the recording signal from the TRS-80 to the recorder for recording programs that you want to save on tape.
4. Connect the smaller gray plug into the REM jack of the tape recorder. This allows the TRS-80 to automatically control the cassette recorder's motor (turns the motor on or off as programs are loaded or saved).
5. There is a dummy plug (plug with no connecting wires) that should be plugged into the MIC jack when you are saving programs on tape. This prevents the microphone from picking up room noises that would spoil the program that you are saving.
6. Finally, connect the recorder's two-prong plug into a 120-volt AC outlet.



Power Up

To power up the computer, all the connections that we have mentioned should be made. Then:

1. Press the power button on the video monitor. Give the video tube a few seconds to warm up.
2. Press the ON button (next to the cable connections on the rear of the keyboard unit). When you have completed these first two steps, the red light on the right side of the keyboard (near the front) should come on. Also, the words MEMORY SIZE? should appear on the video screen.



3. If you were going to program in machine language, this would be your opportunity to protect a section of memory for your machine language program (a program we don't cover in this book). For normal use of BASIC language programs, however, you will not need to protect any memory, so you just press the **ENTER** key. The video display now responds with:

```
RADIO SHACK LEVEL II BASIC  
READY  
>_
```

4. The computer is **READY**. Are you? If so, go to it. Happy programming!!!
In case your TRS-80 doesn't power up the first time, wait a few seconds and try the power up procedure again. Check all the connections to see that they have been properly made.
-

APPENDIX B

The Cassette Recorder

A cassette recorder is provided with the basic TRS-80 system for saving and loading programs that you write. It also is used to load taped programs that you acquire from other sources.

Level II BASIC transfers data from the recorder to the computer (and vice versa) at a rate of 500 BAUD (twice the speed of Level I). The volume setting on the recorder is very critical and should be set at a lower setting for Level II tapes than for Level I. Volume settings are discussed in more detail later in this appendix.

Saving Your Programs

If you have a program in the computer's memory that you would like to save for future use, the recorder must be connected as described in Appendix A. Then:

1. Put the recorder in the RECORD mode by pressing down both RECORD and PLAY buttons. Use a volume setting approximately midway (4-6 on the Radio Shack CTR-41 recorder).

The command `CSAVE` followed by a file (or program) name is used to record programs on tape from the computer's memory. The file name may be any alphanumeric character except quotation marks. The name is enclosed in quotes.

Examples: `CSAVE "1"`
`CSAVE "A"`
`CSAVE "B"`

You should always write file names on the cassette case for later reference. It is also helpful to write the index setting (of the recorder) along with the file name so that the program can be found quickly in the future.

2. On the computer, now type:

```
CSAVE "A"
```

OR whatever file name
you want to use

and press the **ENTER** key.

The recorder motor should be automatically started by the computer, and the program is then recorded. After the recording is completed, the computer will turn off the recorder (hopefully—see “Recorder Problems” in this appendix). The READY message will be displayed on the screen.

```
CSAVE "A"
READY
>_
```

It is a good idea to CSAVE a program more than once on the same tape in case something should happen to one copy. This can be done by leaving the recorder in the RECORD mode and typing:

CSAVE "A" and pressing the **ENTER** key again.

Once again the computer starts the recorder motor, records the program, and turns the recorder motor off. The READY message is again displayed on the video screen. When finished, press the **STOP** button on your recorder.

Checking Recorded Programs

After recording a program, it is advisable to *immediately* check the recording while the original program is still in the computer's memory. Then, if there was an error in the recording, the original can be recorded again. This check can be made by following these steps:

1. Rewind the cassette to the point where the recording of your program started.
2. Press the **PLAY** button on the recorder.
3. On the computer, type:

CLOAD? "A" and press the **ENTER** key.

Note the ? mark

or whatever file
name you are using

The computer will then compare the tape recording with the original program in the computer's memory. If there are any discrepancies, the message: BAD will be displayed on the video screen. In that case, you should CSAVE the program again. If the tapes match correctly, the recording was good, and the READY message is again displayed.

GOOD RECORDING

```
CLOAD? "A"
READY
>_
```

BAD RECORDING

```
CLOAD? "A"
BAD
READY
>_
```

Loading Taped Programs

The volume settings recommended by Radio Shack for loading cassette tapes on a Level II system with the CTR-41 tape recorder are:

User-taped programs 4-6
Radio Shack prerecorded programs 5¹/₂-6¹/₂

For prerecorded tapes by other manufacturers, you will probably have to experiment to find the volume setting at which they will load correctly. (See "Recorder Problems" in this appendix for help.)

To transfer a prerecorded program to the computer's memory:

1. Connect the recorder as discussed in Appendix A.
2. Insert the cassette containing the prerecorded program and rewind the cassette to the beginning of the desired program.
3. Press the **PLAY** button on the recorder.
4. On the computer, type:

```
CLOAD "A" ← or the file name of
              the program that you
              want
```

and press the **ENTER** key.

The computer turns on the recorder's motor. After a few seconds, two asterisks should appear in the upper right corner of the screen. The right asterisk should blink (usually at an irregular rate). When the program has been completely transferred, the computer stops the recorder's motor and gives its **READY** message on the screen.

```
CLOAD "A"
READY
>_
```

If all went well, turn off the recorder. The program is loaded. Type:

```
RUN
```

Happy computing!!

Recorder Problems

The audio cassette recorder provides an inexpensive means to store programs and data files outside the computer. Keep in mind that the recorder was not originally designed for this purpose. It is a cheap alternative to floppy disk drives.

Although far from perfect, it does beat loading programs from the keyboard each time that you want to use them.

Use of the tape recorder for digital purposes requires great patience, understanding, and care. Here are some problems that you may encounter, and some suggestions that may prove helpful. You may not agree with all of them.

- Use high-quality, certified digital tapes. Poor-quality audio tapes may have imperfect magnetic coatings with some spots that would be undetectable when playing music or voices. However, a loss of one bit of data during the save or load of digital data may spoil a good program.
- Keep the heads of your recorder clean and demagnetized. Special tapes are available that can quickly be “run through” the recorder to clean or demagnetize the heads. Special cleaning liquids are also available.
- Be consistent in using a given volume setting when recording and loading your own taped programs.
- Use short tapes and record only one program on each side of the tape.
- Record a given program several times on the tape.

If you follow the above suggestions, only minor problems should arise when loading or saving your own tapes.

Big problems may arise, however, when you try to load tapes from other sources. You must patiently try different volume settings until a good load is accomplished. Recording levels will vary from manufacturer to manufacturer and from tape to tape. Levels have even been known to vary within a given taped program.

- Once you correctly load a tape from an outside source, make a copy of it by saving it with your own volume settings.
- Don’t hesitate to return a program that you have purchased if you cannot get it to load correctly. Not all preprogrammed tapes on the market are perfect, as mass duplication of tapes is as yet an “imperfect art.” The return rate is high.

There are products on the market that enhance the recorded data. They make it possible to load tapes within a wide range of volume settings. Some are done by hardware (a “black box” attached between the computer and recorder) and others are done by software (which must be loaded in from a cassette).

Inside the TRS-80 lives a relay that is supposed to turn the recorder’s motor on or off at the correct time. But relays have been known to stick, so that sometimes the recorder may keep on turning after a program has been successfully loaded. At other times, you may punch the **PLAY** button on the recorder and find that the recorder reaches the beginning of a prerecorded program before you can type in **CLOAD**.

- If this condition persists, don’t hesitate to have your Radio Shack store replace the relay (within the warranty period, if possible).
-

APPENDIX C

Arithmetic

Arithmetic Operators

The symbols used by the TRS-80 differ from the normal arithmetic symbols for division, multiplication, and exponentiation found in a mathematics text.

| | | |
|------------|--------------------|----------------------|
| $+$ | for addition | $5 + 4 = 9$ |
| $-$ | for subtraction | $9 - 6 = 3$ |
| $*$ | for multiplication | $7 * 2 = 14$ |
| $/$ | for division | $14 / 3 = 4.66667$ |
| \uparrow | for exponentiation | $7 \uparrow 3 = 343$ |

Order of Operations

The order in which arithmetic operations are performed by the computer conforms to the normal order used in arithmetic and algebra.

1. 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. Otherwise, the order is from left to right.

| | |
|---------------------------------|---------------------|
| $5*(4 + (3/(2 + 1)) + 1)*(6-5)$ | Innermost first |
| $= 5*(4 + (3/3) + 1)*(6-5)$ | Then next pair out |
| $= 5*(4 + 1 + 1)*(6-5)$ | Now the left one |
| $= 5*6*(6-5)$ | Then the right one |
| $= 5*6*1$ | Then multiplication |
| $= 30$ | |

2. Exponentiation (raising to a power):

$$\begin{array}{ll} 3*2\uparrow 4 & \text{Exponentiation first} \\ = 3*16 & \text{Then multiplication} \\ = 48 & \end{array}$$

3. Negation (-5):

$$\begin{array}{ll} -5\uparrow 2 & \text{Exponentiation first} \\ = -25 & \text{Then negation, as: } -(5\uparrow 2) \text{ not } (-5)\uparrow 2 \\ \text{or} & \\ -5\uparrow 2*3 & \text{Exponentiation first} \\ = -25*3 & \text{Then negation} \\ = -75 & \text{Then multiplication} \end{array}$$

4. Multiplication or division (from left to right):

$$\begin{array}{ll} -5\uparrow 2*3/5 & \text{Exponentiation first} \\ = -25*3/5 & \text{Then negation} \\ = -75/5 & \text{Then multiplication (left)} \\ = -25 & \text{Then division (right)} \end{array}$$

5. Addition or subtraction (from left to right):

$$\begin{array}{ll} -5\uparrow 2-9/3*4+40 & \text{Exponentiation first} \\ = -25-9/3*4+40 & \text{Then negation} \\ = -25-3*4+40 & \text{Then division, multiplication} \\ = -25-12+40 & \text{left to right} \\ = -37+40 & \text{Then addition, subtraction} \\ = 3 & \text{left to right} \end{array}$$

Scientific Notation

Both large and small numbers can be very long, such as:

1234567890000000000

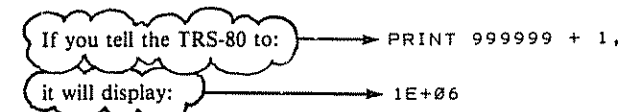
or

0.0000000000123456

Using single-precision arithmetic (the TRS-80 normal mode), results may contain as many places as 39.

You learned in Chapter 2 that the TRS-80 displays results of arithmetic operations with up to six digits, but no more. The largest decimal number that can be expressed with this restriction is 999999. Scientific notation (or floating point notation) is used to print large numbers in a short space.

Large Numbers



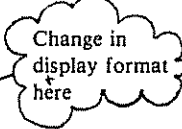
We know that $999999 + 1 = 1000000$. Therefore, $1E + 06$ must be the TRS-80's way to print one million (1000000). This is the way that the TRS-80 displays scientific notation. The normal scientific notation format would be: 1×10^6 . But remember that the TRS-80 cannot write superscripts. By putting the two forms together, we can see the resemblance:

$$\begin{array}{c} 1E + 06 \\ 1 \times 10^6 \end{array}$$

You can see that E stands for exponent, and 06 is the exponent (or power) of ten that 1 must be multiplied by to put the number in decimal notation ($1 \times 10^6 = 1000000$). In other words, move the decimal six places to the right.

To refresh your memory and show how the TRS-80 displays scientific notation, here is a table to show some typical values.

| <i>Decimal Notation</i> | <i>Scientific Notation</i> | <i>TRS-80 Notation</i> |
|-------------------------|----------------------------|------------------------|
| 998 | 9.98×10^2 | 998 |
| 9999 | 9.999×10^3 | 9999 |
| 999999 | 9.99999×10^5 | 999999 |
| 1000000 | 1×10^6 | 1E + 06 |
| 12300000 | 1.23×10^7 | 1.23E + 07 |
| 12345680 | 1.234568×10^7 | 1.23457E + 07 |



Small Numbers

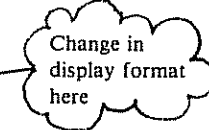
Very small numbers can also become long and cumbersome to work with. Consider: 0.00000000000123. Once again, we can resort to scientific notation to reduce the number to a manageable length. In scientific notation, the number would be: 1.23×10^{-12} . Notice the negative exponent. This means that the decimal point must be moved to the *left* twelve places to put in decimal notation. In other words:

$$1.23 \times 10^{-12} = 1.23 \times \frac{1}{10^{12}} \text{ or } \frac{1.23}{10^{12}}$$

A negative exponent means *division* by powers of ten—move the decimal point left.

A table for small numbers looks like this:

| <i>Decimal Notation</i> | <i>Scientific Notation</i> | <i>TRS-80 Notation</i> |
|-------------------------|----------------------------|------------------------|
| .1 | 1×10^{-1} | .1 |
| .09 | 9×10^{-2} | .09 |
| .01 | 1×10^{-2} | .01 |
| .00999999 | 9.99999×10^{-3} | 9.99999E-03 |
| .0001234 | 1.234×10^{-4} | 1.234E-04 |
| .00001234563 | 1.234563×10^{-5} | 1.23456E-05 |



In addition to displaying some numbers in scientific notation, your TRS-80 will accept numbers for input in that form. Variables may also be assigned in scientific notation.

```
Example: 5 CLS
          10 A = 1.5E+03
          20 B = 1.2376E+05
          30 PRINT A + B
```

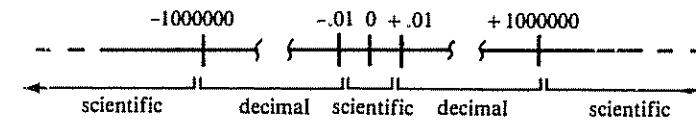
The display: 125260
READY
>_

Notice that the computer accepted the scientific notation variables. However, the result was not large enough to require a change from the normal decimal notation.

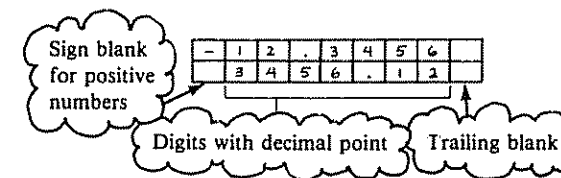
The computer outputs numbers either in decimal or scientific notation, depending on the magnitude of the number. The range of values with the format used to display them is shown below:

| <i>Range of the Displayed Number</i> | <i>Display Format</i> |
|--------------------------------------|-----------------------|
| Number ≥ 1000000 | Scientific notation |
| $.01 \leq \text{number} < 1000000$ | Decimal notation |
| $-.01 < \text{number} < .01$ | Scientific notation |
| $-1000000 < \text{number} \leq -.01$ | Decimal notation |
| Number ≤ -1000000 | Scientific notation |

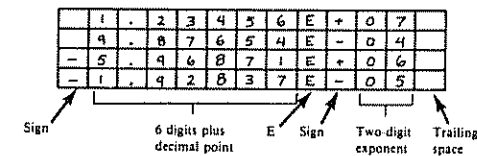
or on the *number line*:



The display format for single-precision numbers allows for six digits, a sign, a decimal point, and a trailing blank space. Typical arrangements are shown below.



This format must allow for a sign for the number, six digits, a decimal point, the letter E, the sign of the exponent, a two-digit exponent, and a trailing space.



Rounding Numbers

When performing arithmetic operations using single-precision arithmetic, the TRS-80 rounds off numbers to six digits. In order to do this, it actually uses the seventh digits of the numbers inside the computer. In other words, single-precision numbers are *stored* in the computer *with seven* digits of precision, but they are *printed* out *with six* digits of precision. For example:

—Inside the TRS-80: $31.08342 + 31.08442 = 62.16784$
but it displays: 62.1678

—and inside the TRS-80: $31.08344 + 31.08442 = 62.16786$
but it displays: 62.1679

Other examples that you can try on your TRS-80:

| <i>Enter</i> | <i>TRS-80 Display</i> |
|---------------------------|-----------------------|
| PRINT 208.1113 + 208.1111 | 416.222 |
| PRINT 208.1113 + 208.1112 | 416.223 |
| PRINT 102.0314 + 102.0310 | 204.062 |
| PRINT 102.0314 + 102.0315 | 204.063 |

Suppose that you want to round a number to a specific place—for example, the tenths place. The decision as to whether to round *up* or round *down* is determined by whether you are *more* or *less* than halfway between two values. For example:

10.83 is closer to 10.8 than to 10.9—round down
 10.84 is closer to 10.8 than to 10.9—round down
 10.86 is closer to 10.9 than to 10.8—round up
 10.87 is closer to 10.9 than to 10.8—round up
 10.85 is halfway—convention used—round up

The value .05 is halfway between one given value in the tenths place and the next higher value in the tenths place:

$$.05 = \frac{1}{2} \text{ of } .1$$

$$\text{or } .5 \times .1 = .05$$

Notice the result of adding .05 to these examples:

$$10.831 + .05 = 10.881$$

$$10.841 + .05 = 10.891$$

$$10.851 + .05 = 10.901$$

$$10.861 + .05 = 10.911$$

The number in the tenths place is now the desired value for the rounded result.

We then go through some arithmetic manipulations to produce the desired result. The complete process is:

1. Add .05 to the original value
2. Multiply by 10
3. Take the INTEger function
4. Divide by 10

This can all be done in one PRINT statement. Examples:

| <i>Statement</i> | <i>Display</i> |
|---------------------------------|----------------|
| PRINT INT((10.831 + .05)*10)/10 | 10.8 |
| PRINT INT((10.841 + .05)*10)/10 | 10.8 |
| PRINT INT((10.851 + .05)*10)/10 | 10.9 |
| PRINT INT((10.861 + .05)*10)/10 | 10.9 |
| PRINT INT((10.871 + .05)*10)/10 | 10.9 |

Since you will want to round off numbers to other places than tenths, here is the method for several places (where A is the number to be rounded):

| <i>Rounding Place</i> | <i>Statement</i> |
|-----------------------|----------------------------------|
| thousandths | PRINT INT((A + .0005)*1000)/1000 |
| hundredths | PRINT INT((A + .005)*100)/100 |
| tenths | PRINT INT((A + .05)*10)/10 |
| ones | PRINT INT((A + .5)*1)/1 |
| tens | PRINT INT((A + 5)*.1)/.1 |
| hundreds | PRINT INT((A + 50)*.01)/.01 |
| thousands | PRINT INT((A + 500)*.001)/.001 |

In general, the statement has the form:

```
PRINT INT((A + B)*C)/C
```

where: A is the original number,
 B is half the unit value of the desired rounded place, and
 C is the reciprocal of the unit value of the desired rounded place.

Typical examples:

| <i>Rounding Place</i> | <i>Statement</i> | <i>Display</i> |
|-----------------------|---|----------------|
| thousandths | PRINT INT((283.7465 + .0005)*1000)/1000 | 283.747 |
| hundredths | PRINT INT((283.7465 + .005)*100)/100 | 283.75 |
| tenths | PRINT INT((283.7465 + .05)*10)/10 | 283.7 |
| ones | PRINT INT((19283.7465 + .5)*1)/1 | 19284 |
| tens | PRINT INT((19283.7465 + 5)*.1)/.1 | 19280 |
| hundreds | PRINT INT((19283.7465 + 50)*.01)/.01 | 19300 |
| thousands | PRINT INT((19283.7465 + 500)*.001)/.001 | 19000 |

APPENDIX D

Error Messages

| <i>Message</i> | <i>Error</i> | <i>Explanation</i> |
|----------------|------------------------|--|
| BS | Subscript out of Range | An attempt was made to assign a matrix element with a subscript beyond the DIMensioned range. |
| CN | Can't Continue | A CONT was issued at a point where no continuable program exists (as after a program was ENDEd or EDITed). |
| DD | Redimensioned Array | An attempt was made to DIMension a matrix that had been previously dimensioned. It is good practice to put all DIM statements at the beginning of your programs. |
| FC | Illegal Function Call | An attempt was made to execute an operation using an illegal parameter. |
| FD | Bad File Data | Data input from an external source (such as tape) was not correct or was in improper sequence, etc. |
| ID | Illegal Direct | The use of INPUT as a direct command. |
| LS | String Too Long | A string variable was assigned a string value that exceeded 255 characters in length. |

| <i>Message</i> | <i>Error</i> | <i>Explanation</i> |
|----------------|----------------------|--|
| L3 | Disk BASIC only | An attempt was made to use a statement, function, or command that is available only when the TRS-80 Mini Disk is connected via the Expansion Interface. |
| MO | Missing Operand | An operation was attempted without providing one of the required operands. |
| NF | NEXT without FOR | NEXT is used without a matching FOR statement. Also occurs if NEXT variable statements are reversed in a nested loop. |
| NR | No RESUME | End of program reached in error-trapping mode. |
| OD | Out of Data | A READ or INPUT # statement was executed with insufficient data available. DATA statement may have been left out or all data may have been read from tape or DATA. |
| OM | Out of Memory | All available memory has been used or reserved. Can be caused by large matrix dimensions or nested branches. |
| OS | Out of String Space | The amount of string space allocated was exceeded. |
| OV | Overflow | A value that was input or was derived is too large or too small for the computer to handle. |
| RG | RETURN without GOSUB | A RETURN statement was encountered before a matching GOSUB was executed. |
| RW | RESUME without error | A RESUME was encountered before ON ERROR GOTO was executed. |
| SN | Syntax Error | This usually is the result of incorrect punctuation, an open parenthesis, an illegal character, or a misspelled command. |

| <i>Message</i> | <i>Error</i> | <i>Explanation</i> |
|----------------|----------------------------|--|
| ST | String Formula Too Complex | A string operation was too complex to handle. Break up the operation into shorter steps. |
| TM | Type Mismatch | An attempt was made to assign a nonstring variable to a string or vice versa. |
| UE | Unprintable Error | An attempt was made to generate an error using an ERROR statement with an invalid code. |
| UL | Undefined Line | An attempt was made to refer or branch to a nonexistent line. |
| /0 | Division by Zero | An attempt was made to divide by zero. |

APPENDIX E

Print and Graphics Layout Sheets

Print and graphics layout sheets are shown proportionally smaller to fit the size of our book pages. You can buy them full size from Radio Shack.

E/Video Display Worksheet

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 | 256 | 257 | 258 | 259 | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 268 | 269 | 270 | 271 | 272 | 273 | 274 | 275 | 276 | 277 | 278 | 279 | 280 | 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 | 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 | 299 | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 | 318 | 319 | 320 | 321 | 322 | 323 | 324 | 325 | 326 | 327 | 328 | 329 | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 348 | 349 | 350 | 351 | 352 | 353 | 354 | 355 | 356 | 357 | 358 | 359 | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 370 | 371 | 372 | 373 | 374 | 375 | 376 | 377 | 378 | 379 | 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 | 389 | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 | 398 | 399 | 400 | 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 410 | 411 | 412 | 413 | 414 | 415 | 416 | 417 | 418 | 419 | 420 | 421 | 422 | 423 | 424 | 425 | 426 | 427 | 428 | 429 | 430 | 431 | 432 | 433 | 434 | 435 | 436 | 437 | 438 | 439 | 440 | 441 | 442 | 443 | 444 | 445 | 446 | 447 | 448 | 449 | 450 | 451 | 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 460 | 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 | 470 | 471 | 472 | 473 | 474 | 475 | 476 | 477 | 478 | 479 | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 | 488 | 489 | 490 | 491 | 492 | 493 | 494 | 495 | 496 | 497 | 498 | 499 | 500 | 501 | 502 | 503 | 504 | 505 | 506 | 507 | 508 | 509 | 510 | 511 | 512 | 513 | 514 | 515 | 516 | 517 | 518 | 519 | 520 | 521 | 522 | 523 | 524 | 525 | 526 | 527 | 528 | 529 | 530 | 531 | 532 | 533 | 534 | 535 | 536 | 537 | 538 | 539 | 540 | 541 | 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 550 | 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 | 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 | 569 | 570 | 571 | 572 | 573 | 574 | 575 | 576 | 577 | 578 | 579 | 580 | 581 | 582 | 583 | 584 | 585 | 586 | 587 | 588 | 589 | 590 | 591 | 592 | 593 | 594 | 595 | 596 | 597 | 598 | 599 | 600 | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 610 | 611 | 612 | 613 | 614 | 615 | 616 | 617 | 618 | 619 | 620 | 621 | 622 | 623 | 624 | 625 | 626 | 627 | 628 | 629 | 630 | 631 | 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 640 | 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 | 650 | 651 | 652 | 653 | 654 | 655 | 656 | 657 | 658 | 659 | 660 | 661 | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 670 | 671 | 672 | 673 | 674 | 675 | 676 | 677 | 678 | 679 | 680 | 681 | 682 | 683 | 684 | 685 | 686 | 687 | 688 | 689 | 690 | 691 | 692 | 693 | 694 | 695 | 696 | 697 | 698 | 699 | 700 | 701 | 702 | 703 | 704 | 705 | 706 | 707 | 708 | 709 | 710 | 711 | 712 | 713 | 714 | 715 | 716 | 717 | 718 | 719 | 720 | 721 | 722 | 723 | 724 | 725 | 726 | 727 | 728 | 729 | 730 | 731 | 732 | 733 | 734 | 735 | 736 | 737 | 738 | 739 | 740 | 741 | 742 | 743 | 744 | 745 | 746 | 747 | 748 | 749 | 750 | 751 | 752 | 753 | 754 | 755 | 756 | 757 | 758 | 759 | 760 | 761 | 762 | 763 | 764 | 765 | 766 | 767 | 768 | 769 | 770 | 771 | 772 | 773 | 774 | 775 | 776 | 777 | 778 | 779 | 780 | 781 | 782 | 783 | 784 | 785 | 786 | 787 | 788 | 789 | 790 | 791 | 792 | 793 | 794 | 795 | 796 | 797 | 798 | 799 | 800 | 801 | 802 | 803 | 804 | 805 | 806 | 807 | 808 | 809 | 810 | 811 | 812 | 813 | 814 | 815 | 816 | 817 | 818 | 819 | 820 | 821 | 822 | 823 | 824 | 825 | 826 | 827 | 828 | 829 | 830 | 831 | 832 | 833 | 834 | 835 | 836 | 837 | 838 | 839 | 840 | 841 | 842 | 843 | 844 | 845 | 846 | 847 | 848 | 849 | 850 | 851 | 852 | 853 | 854 | 855 | 856 | 857 | 858 | 859 | 860 | 861 | 862 | 863 | 864 | 865 | 866 | 867 | 868 | 869 | 870 | 871 | 872 | 873 | 874 | 875 | 876 | 877 | 878 | 879 | 880 | 881 | 882 | 883 | 884 | 885 | 886 | 887 | 888 | 889 | 890 | 891 | 892 | 893 | 894 | 895 | 896 | 897 | 898 | 899 | 900 | 901 | 902 | 903 | 904 | 905 | 906 | 907 | 908 | 909 | 910 | 911 | 912 | 913 | 914 | 915 | 916 | 917 | 918 | 919 | 920 | 921 | 922 | 923 | 924 | 925 | 926 | 927 | 928 | 929 | 930 | 931 | 932 | 933 | 934 | 935 | 936 | 937 | 938 | 939 | 940 | 941 | 942 | 943 | 944 | 945 | 946 | 947 | 948 | 949 | 950 | 951 | 952 | 953 | 954 | 955 | 956 | 957 | 958 | 959 | 960 | 961 | 962 | 963 | 964 | 965 | 966 | 967 | 968 | 969 | 970 | 971 | 972 | 973 | 974 | 975 | 976 | 977 | 978 | 979 | 980 | 981 | 982 | 983 | 984 | 985 | 986 | 987 | 988 | 989 | 990 | 991 | 992 | 993 | 994 | 995 | 996 | 997 | 998 | 999 | 1000 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|

E/1

APPENDIX F

Reserved Words

Although it is convenient to use descriptive words for variable names in a program, some words will produce an error message. You cannot use variable names that contain words with special meaning in BASIC. A complete list of such words is given below.

*Reserved Words**

| | | | |
|--------|---------|--------|----------|
| @ | EDIT | LOAD | RESTORE |
| ABS | ELSE | LOC | RESUME |
| AND | END | LOF | RETURN |
| ASC | ERL | LOG | RIGHT\$ |
| ATN | ERR | MEM | RND |
| CDBL | ERROR | MERGE | SAVE |
| CHR\$ | EXP | MID\$ | SET |
| CINT | FIELD | MKD\$ | SGN |
| CLEAR | FIX | MKIS | SIN |
| CLOSE | FOR | MKS\$ | SQR |
| CLS | FRE | NAME | STEP |
| CMD | GET | NEW | STOP |
| CONT | GOSUB | NEXT | STRING\$ |
| COS | GOTO | NOT | STR\$ |
| CSNG | IF | ON | TAB |
| CVD | INKEY\$ | OPEN | TAN |
| CVI | INP | OUT | THEN |
| CVS | INPUT | PEEK | TIMES |
| DATA | INSTR | POINT | TROFF |
| DEFDBL | INT | POKE | TRON |
| DEFFN | KILL | POS | USING |
| DEFINT | LEFT\$ | PRINT | USR |
| DEFSNG | LET | PUT | VAL |
| DEFUSR | LSET | RANDOM | VARPTR |
| DEFSTR | LEN | READ | |
| DELETE | LINE | REM | |
| DIM | LIST | RESET | |

Examples of improper use of reserved words:

OUTCOME = A + B ← OUT is reserved
BANDY = Q/100 ← AND is reserved
EXPOSE = T + 13 ← POS and EXP are reserved
STABLE = 14 ← TAB is reserved
CONFUSING = A/(A + 3) ← ON, SIN, and USING are reserved
FRED = F*RED ← FRE is reserved

*Many of these words have no function in LEVEL II BASIC; they are reserved for use in LEVEL II DISK BASIC. None of these words can be used inside a variable name.

APPENDIX G

ASCII Codes

| <i>Code</i> | <i>Function or Character</i> | <i>Code</i> | <i>Character</i> |
|-------------|--|-------------|------------------|
| 0-7 | None | 42 | * |
| 8 | Backspaces and erases current character | 43 | + |
| 9 | None | 44 | , |
| 10-13 | Carriage returns | 45 | - |
| 14 | Turns cursor on | 46 | . |
| 15 | Turns cursor off | 47 | / |
| 16-22 | None | 48 | 0 |
| 23 | Converts to 32 character mode | 49 | 1 |
| 24 | Backspace cursor ← | 50 | 2 |
| 25 | Advance cursor → | 51 | 3 |
| 26 | Downward line feed ↓ | 52 | 4 |
| 27 | Upward line feed ↑ | 53 | 5 |
| 28 | Home, return cursor to display position (0,0) | 54 | 6 |
| 29 | Move cursor to beginning of line | 55 | 7 |
| 30 | Erase to end of line | 56 | 8 |
| 31 | Clear to end of frame | 57 | 9 |
| 32 | Space | 58 | : |
| 33 | ! | 59 | ; |
| 34 | ” | 60 | < |
| 35 | # | 61 | = |
| 36 | \$ | 62 | > |
| 37 | % | 63 | ? |
| 38 | & | 64 | @ |
| 39 | ' | 65 | A |
| 40 | (| 66 | B |
| 41 |) | 67 | C |
| | | 68 | D |
| | | 69 | E |
| | | 70 | F |

| <i>Code</i> | <i>Character</i> | <i>Code</i> | <i>Character</i> |
|-------------|------------------|-------------|------------------|
| 71 | G | 86 | V |
| 72 | H | 87 | W |
| 73 | I | 88 | X |
| 74 | J | 89 | Y |
| 75 | K | 90 | Z |
| 76 | L | 91 | ↑ or [|
| 77 | M | 92 | ↓ |
| 78 | N | 93 | ← |
| 79 | O | 94 | → |
| 80 | P | 95 | — |
| 81 | Q | 128 | Space |
| 82 | R | | |
| 83 | S | | |
| 84 | T | | |
| 85 | U | | |

Codes 96-127 provide lower-case codes for the letters whose codes are 64-95.

Index

- Addition, 22
- ASC function, 198
- ASCII code, 198, 203
 - character codes, 204
- Arithmetic, 22, 25
- Array, 214
 - clearing elements, 225
 - strings, 222
 - two dimensional, 233
 - using INPUT, 218
 - using READ and DATA, 216

- BASIC, 3
 - Level II, 4
- Birthday Card program, 138
 - message, 139
- Blink program, 68
- Board Generator program, 304
- BREAK key, 44

- Cassette recorder, 2, 6
- Catenation, 300
- Chaos program, 67
- Chips, 2, 3
- CHR function, 203, 296
 - using immediate mode, 297
- CLEAR key, 14
 - reserving space, 55
- CLEAR N statement, 186
- Computing averages, 117
- Computerized Directory Search, 318
- CONT command, 288
- Countdown-Blastoff program, 99
- Create-A-Character program, 301
- Cursor, 13

- DATA statement, 161
 - format, 163
 - out of data error, 164
 - using FOR-NEXT, 166
 - using mixed numeric data, 172
 - using mixed string data, 172
 - using strings, 171
- Debugging, 282
 - immediate mode, 288
 - review of commands, 291
- Degrees Celsius, 28
- Degrees Fahrenheit, 28
- Diameter of a wheel, 66
 - distance travelled, 66
- DIM statement, 220, 227, 240
- Direct statements, 11, 18
- Division, 24
- Draco program, 193
- Dragon Capture program, 310

- EDIT, 259,
 - cancel-and-restart command, 276
 - cancel-and-exit command, 281
 - change command, 278
 - delete command, 278
 - end-of-line-and-insert command, 269, 273
 - format, 261
 - hack-and-insert command, 271
 - list command, 268, 271
 - save-changes and exit command, 281
 - search-and-kill command, 280
 - search command, 265
 - repeat count, 267
 - review of commands, 290
 - UL error, 262
 - using space bar, 263
- ENTER key, 13

- Errors, 45
 - correcting, 19
 - correcting entire line, 21
- ESP program, 68
- Firefly program, 67
- Flag, 115, 164
- Flash and Move Routine, 306
 - Dragon section, 307
- Floating point numbers, 33, 98
- FOR-NEXT statement, 76, 89
 - counting program 92
 - general form, 91
 - numeric variable, 78
 - time delays, 77, 93
 - using colon, 79
 - using STEP, 98
- Frogs program, 315
 - comments, 317
- Frog Jumping program, 179
- Games, 114
 - number guessing, 114, 120, 121
 - letter guessing, 119
- GOSUB statement, 83
- GOTO statement, 46
- Graphics, 302
- Guess My Letter program, 200
- IF-THEN statement, 111
 - conditions, 112, 114
 - flag, 115
 - general form, 113
- IF-THEN-ELSE statement, 152
- Immediate statements, 11
- INKEY function, 148
 - value of, 149
- INPUT statement, 56, 61
 - assigning values, 65
- Interrupting a program, 150
- Keyboard, 2, 5
- LEN, 190
- LET statement, 52
 - optional, 53, 64
- Letter Guessing Game, 119
- Line number, 40
- LIST, 43
- Loops, 76
- Mandala Ever Chaning program, 144
- Memory, 4, 40
- Metric, 25
- Modular, 296
- Microcomputer, 2
- Multiple statements, 80, 89
- Multiplication, 23
- N factorial, 97
- Number Guessing Game, 114
- Numeric variables, 63, 65
 - using words, 136
- Numerical expression, 22
- Overflow error, 98
- POS, 176
 - using numeric expressions, 178
- Positive-Negative, Zero program, 283, 285
- Powers of numbers, 30
- Power supply, 2, 6
- PRINT statement, 17
 - empty statement, 61
 - LEN, 190
 - POS, 176
 - positions, 85
 - PRINT@, 86, 299
 - quotation marks, 18
 - rounding, 29
 - semicolon spacing, 48
 - strings, 18, 207
 - TAB, 168
 - using parenthesis, 27
 - without quotation marks, 22
- Printing a table, 242
 - totals, 245
- Program, 4, 40
 - entering, 40
 - erase, 41
 - list, 43
 - storing, 44
- Prompt, 14
- RAM, 5
- Random numbers, 106
- Reaction Time program, 153
- READ statement, 161
 - using FOR-NEXT, 166
 - using FOR-NEXT READ, 172
 - using more than one item, 165
 - using mixed numeric data, 172
 - using string data, 172
- Recipe adjustment program, 322
- REMARK statement, 126
- RESET statement, 140
- RESTORE statement, 174
- RETURN statement, 83
- RND function 68, 106, 125
 - using integers, 126
 - using variables, 107
- ROM, 4

-
- SET statement, 133
 - Sequence, 95
 - Shape Matching program, 311, 314
 - Spaceship display, 302
 - Starfall program, 68, 135, 148, 151, 165, 175
 - Stars program, 221
 - Star Twinkling program, 110
 - inside a box, 142
 - Statement, 40
 - Strings, 18, 22, 50
 - catenation, 187, 189
 - CLEAR N, 186
 - comparing, 198, 202
 - LEFT, 191
 - LEN, 190
 - length of, 51, 55
 - MID, 196
 - reserving space, 55
 - RIGHT, 194
 - trailing spaces, 60
 - variable, 52, 60, 118
 - Subroutine, 82
 - time delay, 82
 - Subscripted variable, 214
 - double subscripts, 233
 - errors, 219
 - variables as subscripts, 215
 - Subtraction, 23
 - Syntax error, 16

 - TAB function, 168
 - positions, 169
 - Time delay, 77, 81
 - FOR-NEXT, 93
 - variable, 83
 - Trace, 286
 - TROFF function, 286
 - TRON function, 286
 - TRS-80 Microcomputer System, 2
 - Two dimensional arrays, 233
 - DIM statement, 240
 - using READ and DATA, 235

 - VAL function, 312
 - Variables, 52
 - in mathematical expressions, 64
 - numeric, 63, 65
 - string, 52, 60
 - Video monitor, 2, 5, 13, 298
 - block numbering, 299
-

TRS-80TM Level II BASIC

a self-teaching guide

- No experience needed ■ Complete with index and appendixes
- Learn BASIC terms and commands ■ Create graphic designs
- "Self-tests" check your progress after every chapter

Learn to Program Your TRS-80 with No Previous Computer Experience!

Make the most of your TRS-80! Co-authors Bob Albrecht, Don Inman and Ramon Zamora have combined their extensive backgrounds in the computer field to come up with this self-teaching guide for learning BASIC language for use with your Level II TRS-80. With the help of this step-by-step manual, *you can teach yourself* to read, understand and write programs for your computer *at your own pace!* No matter what your level of experience with computers—from beginner to more advanced hobbyist or professional—you'll find all the information you need to make your computer work for you!

Radio Shack  A Division of Tandy Corporation

TRS-80 Level II BASIC a self-teaching guide

Albrecht
Inman
Zamora

