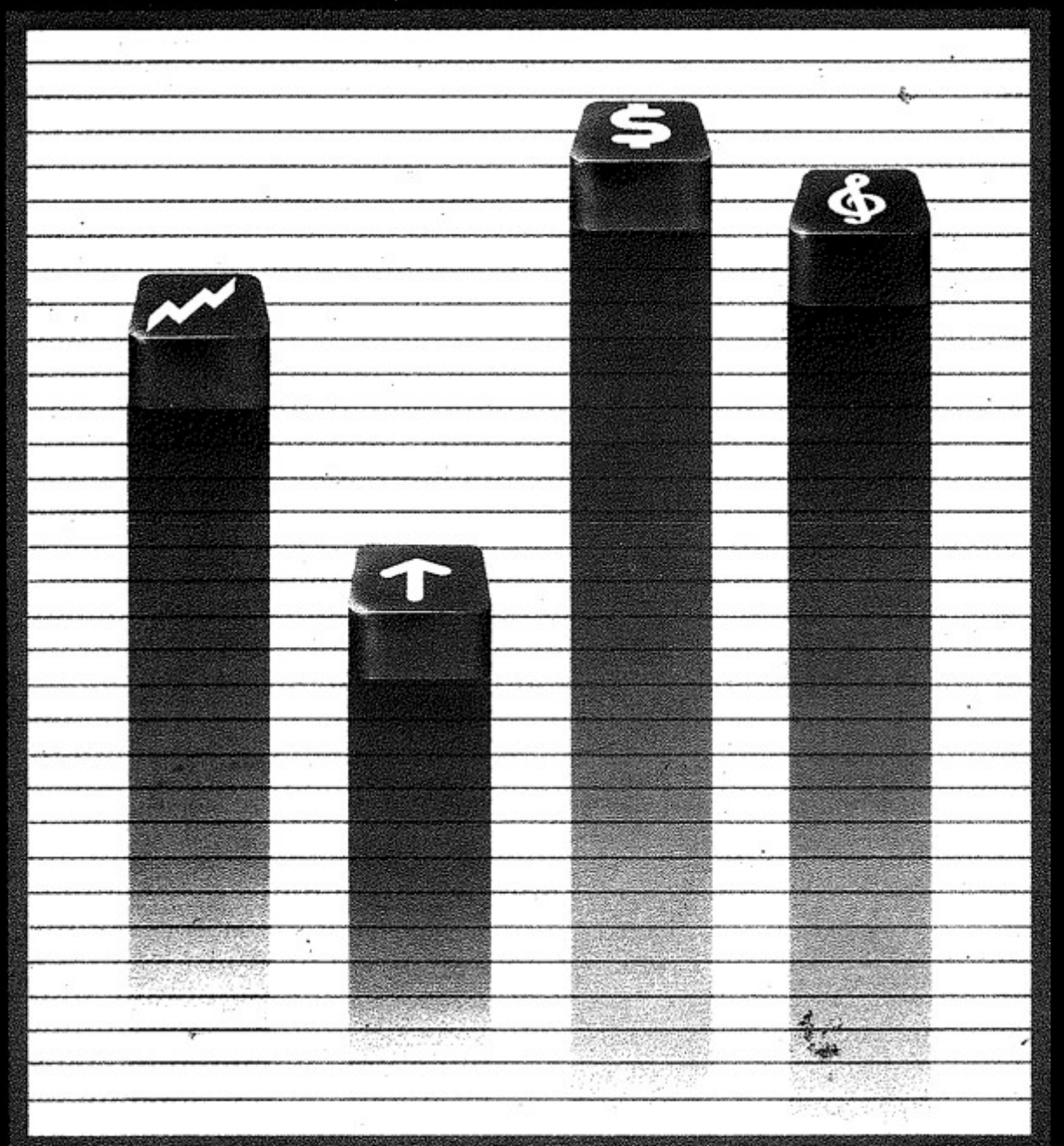
SAMS PRESENTS

22225

Timex Sinclair 2068 Beginner/Intermediate Guide

Fred Blechman



Timex Sinclair 2068 Beginner/ Intermediate Guide

by

Fred Blechman

Howard W. Sams & Co., Inc.

FIRST EDITION
FIRST PRINTING—1983

International Standard Book Number: 0-672-22225-6
Library of Congress Catalog Card Number: 83-50990

Edited by: *Patricia Perry*

Illustrated by: *T. R. Emrick*

Timex Sinclair
2068
Beginner/Intermediate
Guide

Fred Blechman, a former Navy carrier-based fighter pilot, has been writing articles for electronic magazines ever since becoming an amateur radio "ham" (call letters: K6UGT) in 1956. He has had over 250 articles published, with the last 70 articles related to microcomputing. He has also had one book published on programming the Radio Shack TRS-80 Model I microcomputer.

Mr. Blechman studied mechanical and aeronautical engineering at the college level for six years before finding that electronics was his real interest. He built radio-controlled model airplanes in the late 1950s and decided to learn about electronics "after two of my radio-controlled planes flew away!"

He is self-taught in electronics and computer BASIC programming, which, he says "keeps me humble." He has taught a class in programming the ZX81/Timex Sinclair 1000, and has always pursued practical applications of electronics and computing rather than theory, following the KISS ("Keep It Super Simple") approach. He specializes in writing for beginners who want things explained in plain English.

Foreword

Stop right there! I know what you're thinking—*another* book on the Timex Sinclair 2068. It isn't, though, and I'll tell you why if you can spare a moment.

First of all, this book is written by Fred Blechman. In these days of dozens of magazines and thousands of books about small computers it's hard to distinguish one author from another, but you should recognize the name. Fred has written literally hundreds of magazine articles on many, many electronics and computer topics. And he writes with an easy style that is very understandable, an important factor when trying to hack your way through the jungle of small computer buzzwords and jargon. In recent years Fred has become a specialist in small computers, writing up a storm of articles and books that explain to even the most rank beginner how a computer can be used without tears.

Secondly, Fred seems to have extraordinarily good relations with the Timex Computer Corporation. This means that Fred has obtained information on the Timex Sinclair 2068 well in advance of other users. Rather than a hasty few paragraphs added just before publication, this book is *devoted* to the T/S 2068. Beware of other books that reputedly are about the 2068, but show a picture of a Sinclair Spectrum computer on the cover. They are actually reprints of books about a computer that has not been sold in this country. Again, this book was written from scratch to specifically cover the 2068 and only the 2068.

Writers such as Fred are great at their craft of writing, but notoriously poor in promoting themselves. Let me continue in this role of Boswell to round out your picture of an author who can draw on a variety of interests to create the applications in this book. Fred was a carrier-based fighter pilot in the U.S. Navy and has a commercial pilot's license. He's an active amateur radio "ham" (K6UGT) and an avid photographer. He's done many different electronic project designs, most of which have been sold as electronic kits. After 15

years as an aerospace engineer he became a successful entrepreneur, operates several wholesale and retail businesses, and has developed a best-selling package of business programs for Amway product distributors used throughout the world.

With the wide range of interests and experience above, Fred has written a very easy-to-understand book on the T/S 2068. If you've never used a small computer before, you'll find that this book will guide you in a very friendly fashion through the mechanics of operating the T/S 2068 and BASIC programming. If you're an intermediate user, you'll also find a wealth of information in the book—information on the peculiarities of Timex BASIC, the keyboard, and the sophisticated display capabilities. Whether you're a beginner or intermediate user, though, you'll find many different application programs that are not just fluff, but practical, usable programs that illustrate the potential of the T/S 2068.

To my mind, the *Timex Sinclair 2068 Beginner/Intermediate Guide* is *the* book for the T/S 2068. Try it and I think you'll agree!

WILLIAM BARDEN, JR.
Microcomputer Author and Consultant

Preface

The Timex Sinclair 2068 Personal Color Computers are based on the Sinclair Spectrum Color Computer, but with many added features. The physical size, shape, and keyboard of the computer are entirely new. Three new graphic modes, three additional sound channels, bank-switching, joysticks, cartridge capability, and several new commands have been added to the Spectrum design, all of which combine to make the T/S 2068 a very sophisticated machine.

This book, unlike many so-called Timex 2068 books, is not a reprint of a book on the Spectrum. It has been written about the T/S 2068, not the Spectrum.

If this is your first computer, this book, together with the User Manual, will hold your hand. I'll lead you from the mysteries of using programs someone *else* wrote (to do what *they* wanted to do), to the exhilaration of writing programs to do the things *you* want to do! The trip will be tricky. No matter what the fast-talking computer salesperson told you, and no matter how simple the computer ad copy writers say it is, learning to program—even in BASIC—will take time and perseverance. As your "guide" through the software "jungle," I'll be right by your side as we go "In Search of the Lost Byte."

If, on the other hand, you've had BASIC programming experience, you *still* need a guide in this particular jungle. Why? Because the Timex 2068 BASIC is probably quite different in many respects from your past experience—even if you have "graduated" from the T/S 1000 or 1500 or the Sinclair ZX81. Timex also makes several different kinds of watches, but the analog watches with springs and gears are altogether different from their digitals. So, if you have a T/S 2068 computer, you need this book.

"But," you say, "how about the T/S 2068 Computer User Manual? Isn't that good enough?" Sure, it's super. I'd be lost without mine.

However, it covers all the commands and statements with a rather broad brush—it explains what these commands do but it does not teach programming. It can't go into great detail on most things, since it has to concentrate on information rather than application. *This* book's thrust is on applications, but, like all books on computers, it can really only scratch the surface of all the amazing things computers can do. Hopefully, this book will scratch you in a few places you itch!

I'm going to assume you have a Timex 2068 Computer, that you have the User Manual, and that you are somewhat overwhelmed or want more information in some areas. Refer to the User Manual often, since this book does not duplicate the Manual, but rather summarizes or expands its coverage. I'll simplify some sections and illustrate others with actual programs as examples. You will learn by *doing,* by actually keying in each program, reading the line-by-line program explanations, and then making suggested changes. One of the most powerful ways of learning is by example. However, to paraphrase a familiar expression, "I can lead you to the keyboard, but I can't make you program." *You* are the key. Remember, I'm only your guide.

Our "safari" will start (Part I Section A) with setting up camp—getting the computer hooked up, learning the keyboard and describing the layout of the display. Even at this point we'll be putting some ammo in our gunbelts, as it were, by using some simple BASIC programs to illustrate keyboard and display use. Then we'll take a few treks (Section B) into unfriendly territory as we examine the most common BASIC language (a form of Swahili, I believe). Finally, we'll venture into largely unexplored territory (Section C) and revel in the sights and sounds that abound there (was that a ring-tailed pixel or a long-eared sprite?).

By now we've met the natives and found them friendly, so (after some wild ceremonial dances and a few hefty meals of scrumptious alligator stew with barbecued hippo ribs on the side) we get into some meaningful relationships—application programs described in detail in Part II. Actually, most of these programs are not too useful in the jungle, but you should find many of them very useful for home and hobby use just as they are. Also, with your newfound programming skill, they can become "skeletons" for more elaborate customized programs. (Egad! Skeletons in this jungle? Time to split!).

Part III has the trailmarkings, guideposts, and maps—charts, tables, and LISTings—you might need for further exploration.

"All seriousness aside, folks," as Steve Allen would say, I'm going to make this journey as much fun as possible, and you'll find lots of new treasures along the way if you stick in there. I can lead you to the gold, but I can't make you dig.

Ready for the big adventure? Let's go!

FRED BLECHMAN

Acknowledgments

Producing this book to get it into your hands soon after the introduction of the Timex Sinclair 2068 Personal Color Computer involved the effort and cooperation of many people. It was written during the final design and early production stages as the T/S 2068 computers evolved from the Sinclair Spectrum. I thank all those at the Timex Computer Corporation and Sinclair Research who provided me with nonproprietary information on the almost-daily changes and improvements as the design went from "fluid" to "frozen."

In particular, I doubt that this book would be in your hands without the understanding and assistance of George Grimm, Software Development Manager, and Dan Ross, Vice-President of Timex Computer Corporation. George's patience with my numerous calls, coupled with his knowledge and cooperation in providing releasable information in "gray" areas, was critical during the development of this book. Dan Ross's cooperation in providing actual computers and advance information assured accuracy of content.

Sue Mahoney—here's a big kiss for getting an advance copy of the User Manual, and other advance documentation, to me when I needed it so badly!

And to Chuck Durang, who "wrote the book" (The User Manual)—thanks for graciously sharing the pre-publication User Manual Index, and for writing such an attractive and easy-to-follow Manual.

Thanks to Richard Swadley, Acquisitions at Howard W. Sams, for your confidence in my book proposal, and to Bob Manville, Developmental Editor, for bending an ear to some of my suggestions, and steering me along the proper path through all the pitfalls. It was a pleasure working with you!

To Bill Barden, Jr., for whom I have the greatest respect, and whose friendship I treasure, thanks for all the encouragement, the sharing of information—and the great Foreword!

All this would be naught, however, without the endurance, understanding and encouragement of my best friend—my wife, Ev. She's the greatest!

Contents

PART 1—Getting Started

Section A—Up and Running

CHAPTER 1

CHAPTER 2

CHAPTER 3

Section B—The Basics of BASIC

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

Section C—Adding the Frills

CHAPTER 8

CHAPTER 9

CHAPTER 10

PART 2—Some Practical Programs Explained

Section A—Home & Personal Use Programming

CHAPTER 11

CHAPTER 12

CHAPTER 13

Section B—Hobby Programming

CHAPTER 14

CHAPTER 15

CHAPTER 16

CHAPTER 17

APPENDIX A

APPENDIX B

APPENDIX C

APPENDIX D

PART 1
GETTING STARTED

SECTION A
UP & RUNNING

You're the proud new owner of a T/S 2068 Computer and can't wait to get it up and running. But, wait! What are all those other things in the box? Cables, power supply, switch box, and a User Manual. What do you do with them?

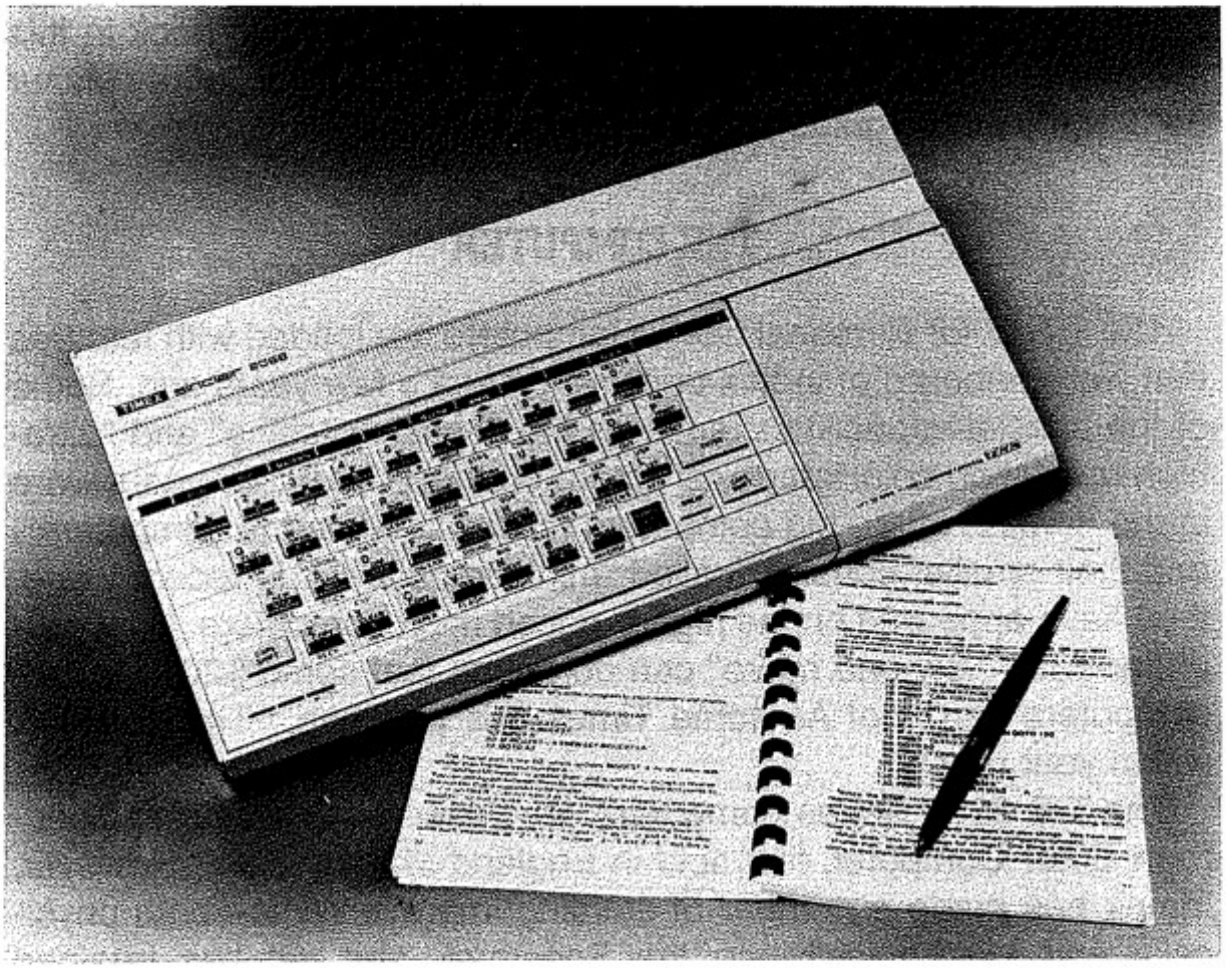
Chapter 1 describes the various computer system components (including some not in the box!) with a discussion of some alternatives.

Chapter 2 takes you on a tour of the complex keyboard (over 220 standard and inverse characters, commands, statements and functions from only 42 keys!) using a sample program to calculate square roots and demonstrate the various color and flashing options.

Chapter 3 describes display formatting for text and graphics with some simple practice programs.

Canteen full? Backpack secure? Machete sharpened? Off we go "In Search of the Lost Byte."

1
The Power Is Within Your Reach



The Timex Sinclair 2068 personal color computer.

INTRODUCTION

Chapter 1 of the User Manual that comes with your T/S 2068 Personal Color Computer does a nice job of showing you how to interconnect the various parts of the computer system—but there is no discussion of what all the parts do, or some of the alternatives. This chapter, rather than repeating the manual's "How To Set Up the Computer," will go into more detail of the components of the system, and some of the precautions. (Out here in the jungle, for example, we have to be sure that our TV/computer switch is set to "computer," or a Tarzan yell on a re-run brings all the chimpanzees into camp!)

Some of the material in this chapter will be for the intermediate reader rather than the beginner. If it gets too tough for you, go on to Chapter 2. However, try to wade through it (even if you don't follow it all) to acquaint yourself with some computer jargon and the features of the fantastic machine you have.

Some of you will want even more detail. Some subjects are covered in more depth in later chapters of this book. However, for those interested in considerably more detail, see the second book in this series, the *T/S 2068 Intermediate/Advanced Guide* by Jeffrey Mazur (who is *my* jungle guide!)

THE COMPUTER

The T/S 2068 Personal Color Computers are loaded with features that were unheard of in this price range just a few years ago. Some of the special capabilities, such as the "bank switching" and "ultra-high resolution color graphics" (more on those later), for years were in the domain of the big computers (called "mainframe computers") and their smaller cousins (called "minicomputers"). In recent years, they have appeared on the higher-priced microcomputers. Now your under-$200 "low-end micro" can do things that some $2000 microcomputers can't do! As some Timex ads proclaim, "The power is within your reach."

The T/S 2068 Series computers are based on the Sinclair Spectrum Color Computer, first sold in England in early 1982. However, Timex has made a number of startling improvements—both physically and electronically—to the Spectrum, and yet has retained downward-compatibility with most Spectrum programs available in England (and certainly those to be available soon in the U.S.). This provides the T/S 2068 with a huge, existing program base—a definite advantage to users, who usually have to wait at least six months after the introduction of a new computer before a good variety of software is available.

Let's take a look at some of the less technical specifications of the T/S 2068 and explain what they mean. Just remember that this is background material which might be of interest to you, but anything you really need to know to operate your T/S 2068 will be covered in later chapters.

Memory

Within the computer there are two main kinds of "memory"—ROM,

You can't change ROM; it contains the "language" and the "operating system" of the computer. The ROM is permanent; when you turn off the computer, the information and instructions it contains stay there.

The RAM is for your use, for you to enter your own programs. The RAM, however, is "volatile," and any information it contains disappears in an instant when the computer is turned off or there's a power failure.

The T/S 2068 has 24K (that's roughly 24 thousand) "bytes" reserved for ROM. Each byte consists of eight memory cells (one memory address) that define a "binary code" to represent and store any decimal number from 0 to 255. These numbers then represent characters or instructions to the computer.

The 24K ROM section of memory contains:

1. the BASIC interpreter, discussed later.
2. a set of routines which provide input and output for display text and graphics, keyboard, printer, sound, and joysticks.
3. channeled input/output to various "dumb" and "intelligent" peripheral devices that can be connected to the computer's "expansion bus" (peripheral connector).

The Timex Command Cartridge

The T/S 2068 contains 48K (49,152 bytes) of on-board RAM for regular program use. By using a "Command Cartridge," a 2½-inch square plastic case which plugs into an interface built into the T/S 2068, you can use a total of 72K bytes. The cartridges contain pre-programmed ROM, with their own operating system included, so the 16K usually devoted to ROM can be switched out. Cartridges need only be inserted into the computer; they don't require loading (which is necessary when using cassette tapes). They operate in the same manner as cartridges for game machines such as the Atari VCS, ColecoVision, Intellivision, and others.

Bank Switching

"Memory" has become somewhat of a numbers game, since an 8-bit computer such as the T/S 2068 (which uses an 8-bit Z-80 microprocessor) can only address 64K (65,536 bytes) at any one time. Ahhh—but there's the rub: "at any one time." The T/S 2068 gets around this with what is known as "bank switching," a feature new to low-end microcomputers, and not available on the Sinclair Spectrum.

The 2068 can actually switch instantly between 255 additional

address up to 16,777,216 bytes (65,536 times 256)—also known as 16 megabytes—of memory, in "chunks" of 8K from any of the 256 banks, for a total at any one instant of 64K. If you want it, however, you'll need to add all that external memory yourself, since the T/S 2068 does not *include* 16 megabytes of RAM.

What's the advantage of all that memory? Well, you probably won't need it, but if you do have programs that need a lot of memory for DATA or files, this information could be held in external RAM for immediate access without using tape recorders or disk drives. If all this is getting beyond you, ignore it for now. You don't need to know any of this to use your T/S 2068 for most things.

Keyboard

In another severe departure from the Spectrum design, Timex greatly enlarged the case for the 2068 and replaced the Spectrum closely spaced "chiclet" keys with real typewriter keys—the same keys used on the Brother EP-20 Personal Electronic Typewriter. The keyboard is standard typewriter size and the full-travel keys are labeled in the standard QWERTY layout. ("QWERTY" refers to the labeling of the first six keys on the top letter-row of a standard typewriter keyboard.)

There are forty-two keys (including a space bar and a CAPS SHIFT key on each end of the keyboard) with automatic repeat (after held down a programmable period of time). Most of the keys do multiple duty, since keyword entry is provided for all commands and statements, and this takes some time to get familiar with. Many keys perform six different functions, and most perform at least five.

> **Note:** Throughout this book computer "keywords" will be capitalized in the text.

There is a RESET key and 16 block-graphic keys, as well as 21 user-defined graphics that can be programmed for the A–U keys. Many special symbols are available, such as a copyright symbol, curly brackets, an "at" sign, an English "pound" sign and—much to the joy of T/S 1000/1500 owners—an exclamation point!

The Display

The display can be programmed to show various "attributes" (characteristics) for each character space in most display "modes."

Eight colors are available for PAPER (background), INK (printing), and BORDER (outside the regular display area). Also, there are two BRIGHTness levels for each color, and FLASHing is available for each character space. INVerse VIDEO provides for a switching of PAPER and INK colors in any character space. All this provides the capability for an endless variety of special effects.

The T/S 2068 has four display modes. (The Spectrum has only one.) The Normal Display Mode, 1, provides 24 rows of 32 characters on a line. Two of these lines are reserved for program entry and editing, or program input. This same mode offers sixteen character space graphic blocks (four sections to each block) or 256 "pixels" (picture elements, or "dots") on a line, with 176 lines from top to bottom.

Display Mode 2 is the 64-column Mode, with 22 rows of 64 characters on a line, and a pixel resolution of 512 by 176.

Display Mode 3 is a second "normal" screen, like Mode 1. By switching rapidly between Modes 1 and 3 you can produce animation.

The fourth, Display Mode 4, is called "ultra-high color resolution." It has the same character and pixel resolution as Mode 1 (32 character lines, 256 pixels on a line), but with an important difference. Each character space contains eight rows of eight pixels each, and in this mode each row of pixels in each character space can be assigned various color parameters (called INK, PAPER, BRIGHTness, and FLASHing). This allows great flexibility in color mixing and other visual effects.

Spectrum BASIC-Plus

The language used in the T/S 2068 is Sinclair Expanded BASIC, sometimes referred to as Spectrum BASIC, but with a number of additional commands not found on the Spectrum. To give you an idea of the power of the 2068, it has more than 50 commands and statements that are not available on the T/S 1000/1500! It even has a number of commands for devices not yet available, such as external disk and disklike storage devices. For the former T/S 1000/1500 user who has been frustrated by not being able to translate programs from other computers, these new commands (such as READ, DATA, RESTORE, and many others) should make the task feasible.

Table A-1 in Appendix A shows the differences between the BASIC language and keys on the T/S 1000/1500 computers and the T/S 2068.

Sound

The T/S 2068 has four "voices," three of which are programmable in eight octaves. On those three channels, the SOUND command allows you to specify tone, duration, amplitude, and various envelope parameters—it's a very sophisticated system. The single-channel BEEP command allows you to specify any of 130 semitones and a duration. Output comes through a built-in speaker. The Spectrum, incidentally, directs only the BEEP command to a very small sounding device. (I plan on programming the 2068 to produce Gene Krupa jungle-drum sounds. Could start a whole new fad here among the natives—"Jungle Boogie"!)

PERIPHERALS

Two joystick ports are included on the T/S 2068. Jacks are provided for a cassette recorder, TV receiver, TV monitor, and a power supply. There's even an on-off switch (a first for a Timex computer). A 64-pin "bus" connector at the rear allows you to connect the Timex 2040 Personal Printer—and other devices. Let's talk about each of these.

Joysticks

On each side of the T/S 2068 there's a joystick port for use with eight-position, industry-standard joysticks. They are used with the STICK command, as described in the T/S 2068 User Manual. In addition to game control, creative programmers will certainly use these joysticks for graphic art generation, as well as for cursor control in word processing, data-base and spreadsheet programs.

Recorder

All the literature these days, including the T/S 2068 User Manual, seems to assume that the user will store programs on a cassette tape recorder. That's probably true for most, because of the convenience, low price, and availability of cassette recorders—but you should know that virtually any kind of tape recorder can be used. There is nothing special about tape cassettes except their low price and convenience. You could use your old reel-to-reel recorder if you like; professional tape duplicators usually use reel-to-reel machines

Don't get too particular, however, about using a fancy high-quality recorder. Sometimes the least expensive recorders have good high-frequency response, even though their low-frequency response is terrible, and high-frequencies are what we want to record. Conversely, the high-frequency "roll-off" built into some hi-fi equipment can prevent a good tape transfer from the T/S 2068. Your best guide is trial and error. Just don't go out and buy a new cassette tape recorder for your 2068—it might not be necessary.

In any case, you must have a recorder that has an earphone output (usually labelled "EAR," "MONITOR," or "SPEAKER") and a microphone input (usually called "MIC" or "MIKE"). The dual audio cable that comes with the T/S 2068 is intended for use with the most common cassette jack—the so-called "miniature phone jack." Adapters are available at electronic stores, however, for mating with virtually every other kind of jack.

Incidentally, if you've used other microcomputers with a recorder, you may be used to a "remote" connection that controls the starting and stopping of the recorder. The T/S 2068 does not have this feature, so you must remember to start and stop the recorder each time you use it. (I've trained Alexander, my pet chimpanzee, to do that—but I can't seem to keep him from chewing on the cassettes!)

TV Receiver—or Monitor?

The "transfer switch box" supplied with the T/S 2068 is intended for use with a regular television receiver to connect the TV antenna terminals to either the regular TV antenna or the signal from the "TV" output jack on the back of the computer. Make sure the switch is in the "Computer" position when using the computer, and that your TV set is tuned to the same channel (2 or 3) as the selector switch on the bottom of the computer.

If you intend to use only Display Modes 1, 3, or 4 (32 characters on a line), a regular TV set will be adequate for a display. You'll probably get a clearer, sharper picture with a black-and-white TV than a color set, and any "color" generated by the computer will appear as shades of gray. Each "color" will be distinguishable from another—which is not true of some computers which display the same shades of gray for several colors. A color display is much more impressive, but may be harder to tune for a clear picture. See Chapter 8, "Color Me Gorgeous!)," for some more information on tuning for color.

Chapter 8 also discusses the need for a "monitor" when using the

olution than can be provided by the narrow 4 MHz bandpass of a regular TV. A monitor is a TV set without a tuner and with circuits designed and aligned for a higher bandpass (at least 6 MHz, but 12 MHz is better). The "MONITOR" jack on the back of the T/S 2068 provides NTSC compatible color video for use with a color video monitor. (The Spectrum doesn't have this jack.)

Chapter 8 also discusses the use of a special color monitor using the pure color signals generated within the computer—the R-G-B signals (red, green, blue)—for use with an "RGB Monitor." This will provide the sharpest picture, but it requires a special cable connected to the T/S 2068 "bus." The Spectrum does not have these signals on its bus. See Chapter 8 for some details on connecting an RGB monitor.

Power Supply

The power supply furnished with the T/S 2068 is a wall-plug type which includes a connector to mate with the "POWER" jack on the back of the computer. There is a danger, here, however, in that the computer does not have a fuse.

In the preparation of this book, I accidentally plugged the wrong power supply into the computer—24 volts AC instead of 15 volts DC! In less than three seconds there was a loud "pop" inside the computer. On investigation I found the voltage regulator inside the computer had literally exploded—and before it blew, it allowed the "execution" of a bunch of other parts. Moral of the story: Make sure you use only the correct power supply voltage and polarity by using the power supply furnished with the T/S 2068!

I suggest you plug the computer power supply into the back of the computer and leave it there. The computer has an on-off switch on its left side, and the power transformer in the wall plug draws practically no power when the computer is turned off. The less often you plug the power supply into the computer, the less often you will take the chance of putting in the wrong plug!

The Printer

The Timex 2040 Personal Printer is a great accessory for use with any of the Timex computers. (It will work with the T/S 1000 and 1500 as well as the 2068.) It is hard to see how, except for playing games, you can derive any real value from your computer without "hard

copy." The Timex computers all provide three very simple commands (LLIST, LPRINT and COPY) so you can LIST your programs, PRINT program results, and COPY every INK dot on the screen. The 2040 does this quickly (about two screen lines per second) on 4.33-inch-wide white thermal paper, using black printing that photocopies nicely.

This printer (unlike the ZX Printer sold in England for use with the Spectrum) has its own power supply, so it draws no power from the computer. However, it was the printer power supply that I accidentally plugged into the computer in the horror story related above. Unbelievably, both power supplies use the identical output connector, even though the printer supply is "deadly" to the computer! I've since marked the printer supply connector with a big, red warning label. Hopefully, later models of the T/S 2068 or printer will have different power connections.

Connecting the printer to the computer is done at the connector on the rear of the computer. With the power "on" for both the computer and the printer "off," slide the printer connector along the computer connector card edge until the "key" in the printer connector mates with the slot in the computer connector and press them together. Note that the printer connector is a "sandwich type" and has provision to accept an additional plug-on device.

The printer has two switches, "OFF" and "ON/ADVANCE." The power to the printer is *not* turned off by the OFF switch, as you might think. It merely allows the computer to bypass any printer commands. It is also used with a printer "self-test," by pressing both switches at the same time.

Unfortunately, it's easy to forget to unplug the printer, since there is no pilot light to tell you it's on. You'll notice, even though you press the OFF switch, the unit stays warm, consuming power, unless it's actually unplugged.

The Timex 2040 produces 32 characters on a line—exactly the same as the screen in 32-character display modes. However, for word processing or any serious use—or to reproduce the 64-character display mode—you'll want to connect a standard 80-column printer and print on regular paper, with multicopy capability. Timex, at this writing, only offers the 2040 Printer. Various other firms have designed printer interfaces for use with the Timex 1000 and 1500 allowing the use of so-called "Centronics parallel" printers. It can be expected that this will also be done for the T/S 2068. For a further discussion of printer interfacing, see the *Timex 2068 Intermediate/Advanced Guide.*

Other Peripherals

Timex has already announced plans to produce a modem for their computers. This will allow you to connect to other computers via the phone lines. With appropriate programming, you'll be able to communicate, to exchange information and even "upload" or "download" programs. You'll be able to connect to "bulletin boards" and large networks and data bases like CompuServe and The Source, leaving messages (electronic mail) or carrying on real-time exchanges in a conference or on a CB simulation. You can also play very complex games on large computers with other participants—or against the mainframe. The modem provides your computer with a whole new horizon.

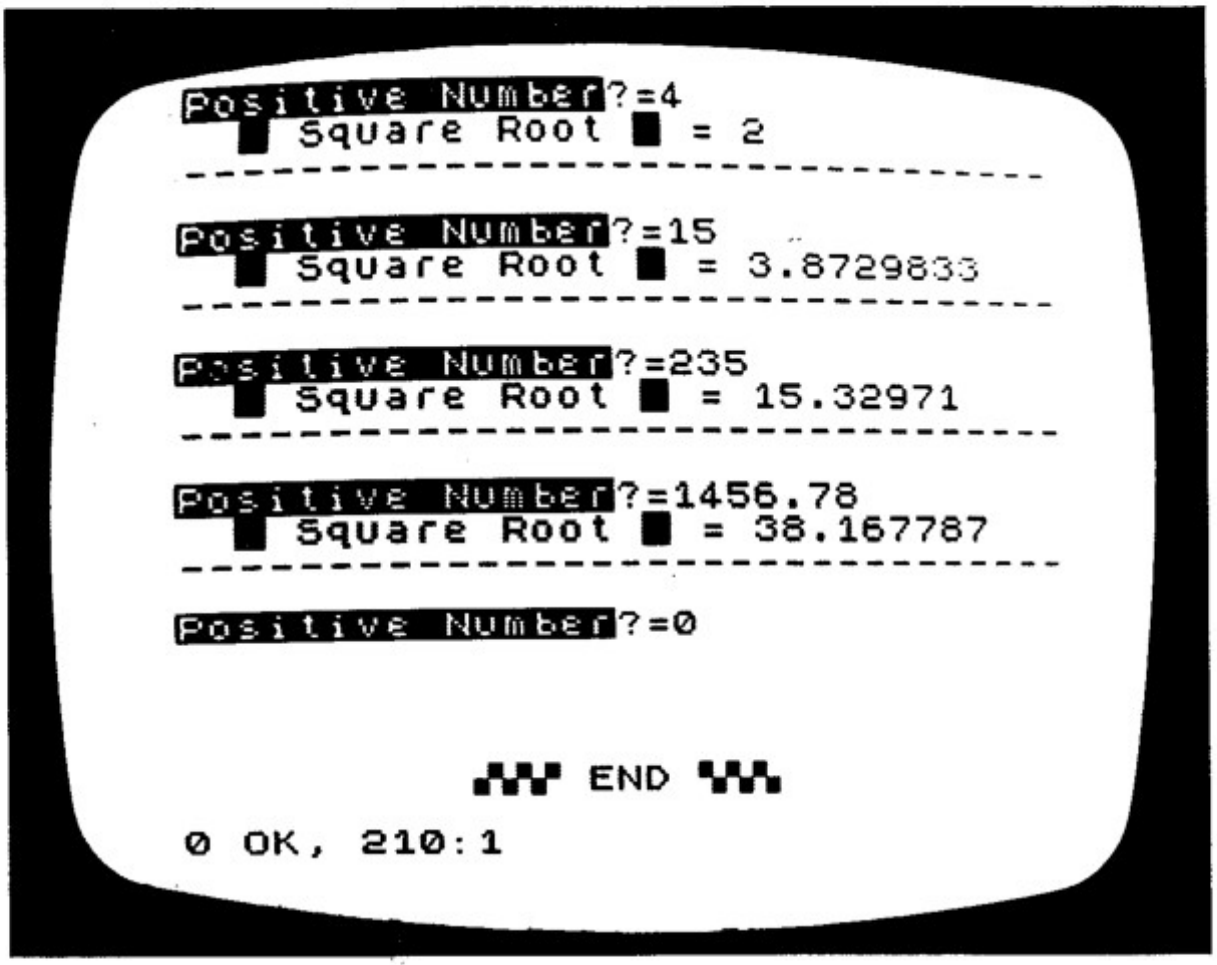
We can also expect some computer wizards to produce "light pens" (to draw directly on the screen or to select items on the screen), "bit pads" (the same thing on a flat surface in front of you), "mice" (cursor positioning devices), and other not-yet-thought-about devices for the T/S 2068. Certainly, the software at first will be games, games, and more games (since the color, sound, and high resolution graphic capabilities are ideal for them), followed by word-processing and business software. (Then a jungle-drum program for Alexander, "The Banana Hop.")

With the incredible memory-addressing capabilities of the bank-switching, the ultrahigh resolution, the 64-character line, the cartridge slot, the sound, the color, and all the rest of the numerous features in the T/S 2068, "the power" is truly "within your reach." Go for it!

QUESTIONS

1. What computer was the original basis for the T/S 2068 design? Is the software for that computer compatible with the 2068?
2. When the computer is turned off, which retains its "memory," ROM or RAM?
3. Normally, how many locations can be addressed by an 8-bit computer like the T/S 2068? Using bank-switching, how many locations can the 2068 address?
4. How much of the 2068 memory space is reserved for ROM?
5. How do you "load" a cartridge into a T/S 2068?
6. What is meant by the term "QWERTY keyboard"?
7. How many regular graphic characters are available from the T/S 2068 keyboard? How many "special" graphic characters?
8. In Normal Display Mode 1, how many characters are there on a line, how many lines, how many pixels per line, and how many lines of pixels?
9. How many sound channels, including the BEEP command, are available on the T/S 2068?

2
Learning the T/S 2068 Keyboard



RUNning the square root program.

INTRODUCTION

If you own a T/S 1000 or 1500, or a Sinclair ZX81, you know how puzzling the keyboard is until you get used to it. Many functions require two keys or a special sequence of keys. Well, the T/S 2068 Personal Computer has a keyboard with two additional keys and about fifty more commands, statements, functions, and symbols. See Table A-1 in Appendix A for a look at these differences, particularly if you have become very familiar with T/S 1000/ZX81 BASIC.

This extraordinary computer has color, sound (four voices), lowercase characters, 16 or 48K RAM built-in (plus banked memory switching and a cartridge slot for even more), high-resolution and

ultrahigh resolution graphics, and flashing characters, just to name some of the additional features. If you like the T/S 1000/1500/ZX81, you'll absolutely *love* the 2068, once you figure out the keyboard.

THE T/S 2068 KEYBOARD

Fig. 2-1 shows the T/S 2068 Series keyboard. The keys push down like a regular typewriter, and are the same keys used on the Brother EP20 Personal Electronic Typewriter. Most keys have three legends on them—one within a black zone on the key, plus additional legends on the keyboard surface above and below. There are also additional color legends above eight keys on the top row. The result is a great mystery to the uninitiated, but is really very simple for those "in the know." With the practice you can get from this chapter, you'll have the keyboard under your control. However, *using* the keyboard is really a lot simpler than *explaining* how to use it, so this chapter appears more complex than it really is.

Sample Program

I'll use a sample program (LISTing 2-1, Fig. 2-2) as a training device to familiarize you with the keyboard. This program does not come even close to using all of the keyboard legends, and it won't teach you how to program, but it does use most of the combinations you'll run into when you enter your own programs. The keystrokes to enter this program into a T/S 2068 will be detailed, but you'll end up being able to determine the square root of a number, with color and sound illustrating some of the programming potential. As an added bonus, I'll explain what each line in the program does, so you'll learn a little T/S 2068 BASIC programming as well.

The Cursor

The "cursor" is a single blinking character found on your screen at the location of your next entry, usually somewhere along the bottom two screen lines. The T/S 2068 has six different cursors to tell you the computer "mode." The K-cursor tells you a "keyword" is expected next. A keyword is a word or abbreviation printed on, above, or below most keys. The L-cursor tells you that a character or symbol is expected next. These will be capital letters if you hold either of the two CAPS SHIFT keys down, or various symbols or

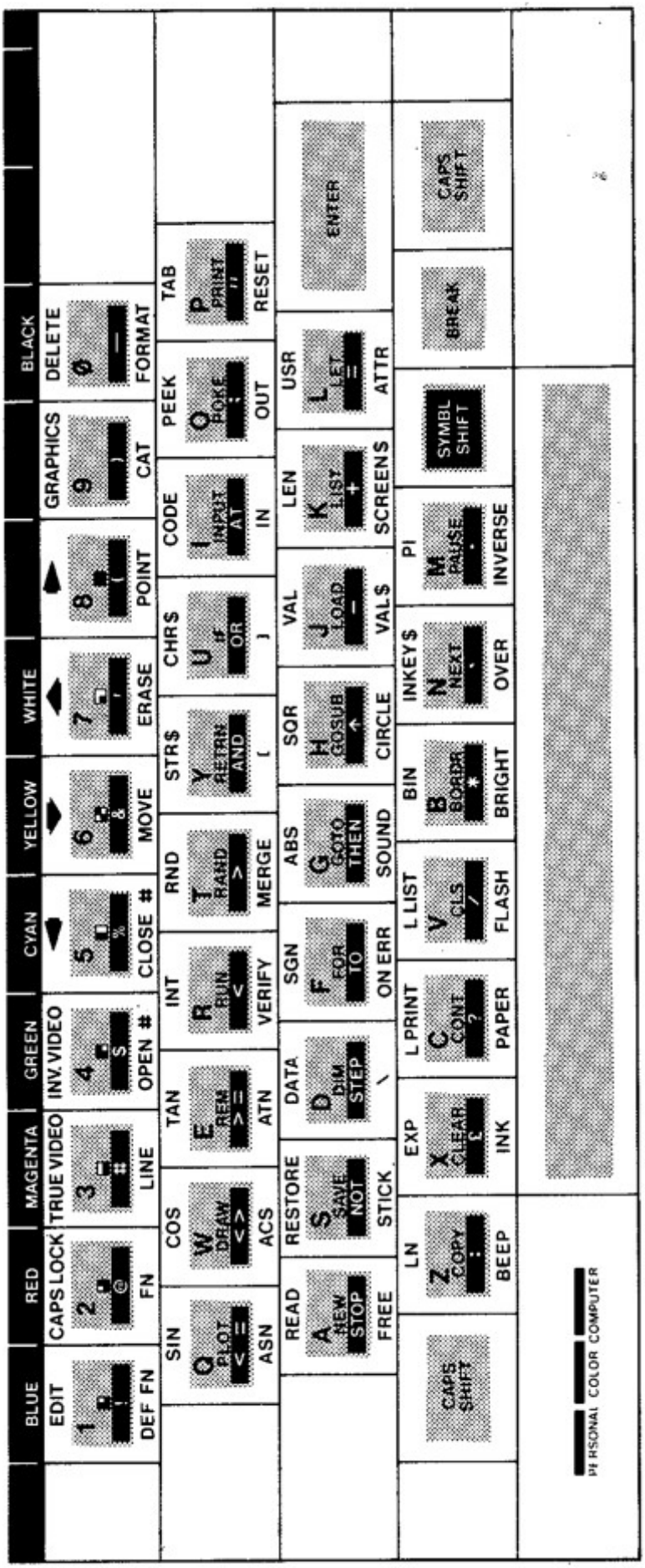


Fig. 2-1. The T/S 2068 keyboard. ***(Courtesy Timex Computer Corp.)***

```
Positive Number?=2
  ▌ Square Root ▐ = 1.4142136
--------------------------------

Positive Number?=5
  ▌ Square Root ▐ = 2.236068
--------------------------------

Positive Number?=9
  ▌ Square Root ▐ = 3
--------------------------------

Positive Number?=3.124
  ▌ Square Root ▐ = 1.7674841
--------------------------------

Positive Number?=785.32
  ▌ Square Root ▐ = 28.023562
--------------------------------

Positive Number?=0
          ▚▚ END ▞▞
```

Fig. 2-2. Typical entries.

LISTing 2-1

```
  10 POKE 23609,100: LET p=7: LE
T i=0: LET b=7
  20 PAPER p: INK i: BORDER b
  30 FLASH 1: PRINT "Positive Nu
mber?=";: FLASH 0
  40 INPUT a: PRINT a
  50 IF a<=0 THEN GO TO 200
  60 LET x=SQR a
  70 PRINT TAB 2;"▌ Square Root
▐ = ";x
  80 PRINT "--------------------
------------": PRINT
 100 LET p=p-1: IF p<0 THEN LET
p=7
 110 LET i=i+1: IF i>7 THEN LET
i=0
 120 LET b=b-1: IF b<0 THEN LET
b=7
 130 BEEP .2,0: BEEP .2,7: GO TO
 20
 200 PAPER 7: INK 0: BORDER 7
 210 PRINT AT 21,10;"▚▚ END ▞▞
"
```

keywords (printed within the black areas of the keys) if you hold the black SYMBL SHIFT key down. The C-cursor tells you the computer is in the CAPS LOCK mode—you get there by holding down either of the two CAPS SHIFT keys and pressing the Number 2 key (CAPS LOCK) once. The E-cursor indicates the next entry will be the legend printed above the keys (unshifted) or below the keys (SYMBL SHIFTed). The G-cursor tells you the computer is in the GRAPHICS mode (CAPS SHIFT and 9 key together) to access the square graphic symbols on keys 1 through 8. The ?-cursor will appear whenever you make an error in a program line entry (or when EDITing a line) and you press the ENTER key. The ?-cursor pops in right where the error is. The line will not be accepted into the program until this error is eliminated, or the line deleted.

The Devious DELETE Key

Confused? Sure you are. But things will become clearer as we enter the program. Before we go any further, however, you should know how to DELETE keyboard entry errors. Normally, if you are within a line being entered at the bottom of the screen, this is easy. Just hold down either CAPS SHIFT key and press the 0 key in the upper right hand corner of the keyboard. Every keypress of the 0 will backspace the cursor, erasing a character or complete keyword.

However, the DELETE function works two ways. If the L-cursor or C-cursor is active, DELETE will backspace, erasing the character or keyword to the left. But, if there is a K-cursor on the screen, then the keyword DELETE appears when you press CAPS SHIFT and 0. This is just fine if you intend to DELETE a block of program lines. That's what the DELETE command, discussed in Chapter 4, is really for. However, if you are using the DELETE function for line EDITing of line numbers or a colon on a multistatement line, then you've got to "outsmart" the DELETE command. You do this by pressing and *holding down* both the CAPS SHIFT and 0 keys until an auto-repeat function takes over. The word DELETE will disappear and you'll be in the backspace mode again.

Also, you must get familiar with the three SHIFT keys on the T/S 2068. The two CAPS SHIFT keys are used to access the legends directly above the top row of keys, and to specify uppercase (capital) letters. The SYMBL SHIFT key is used to access the keywords and symbols within the black area on the keys. When either of the CAPS SHIFT keys and the SYMBL SHIFT key are pressed at the same time, the E-cursor is generated and the next keystroke is the function

above the keys or, if the SYMBL SHIFT key is also held down, the function shown *below* the keys. Although this all seems incredibly complicated, it's surprising how quickly you'll get used to it with some practice.

GETTING SQUARED AWAY!

Let's enter the Square Root program into the T/S 2068. Get the computer up and running following the instructions in Chapter 1 and the User Manual. Your video screen should have only the Sinclair and Timex copyright messages on the two bottom TV screen lines. Follow me as we enter the program line by line and keystroke by keystroke.

Line 10:

Press the 1 key, then the 0 key. Notice how the K-cursor appears, and moves to the right with each keystroke. Now press the letter O key, which has the word POKE printed on it. This keyword now appears on the screen, with a space both before and after it, and the cursor changes to an L, since letters or numbers usually follow a keyword. Next press the 2 key, then the 3, the 6, the 0, and the 9 keys. As you press each key, the number appears on the screen and the cursor moves to the right. Next you need to enter a comma. If you check the N key, you'll find a comma in the black area. Since it's in the black area, you need to press the SYMBL SHIFT key to access it. So, hold down the SYMBL SHIFT key and press the N key and the comma magically appears on the screen. Now press 1, 0, and 0 in sequence, and you have the first part of line 10.

What does POKE 23609,100 do? On the T/S 2068, it provides a short audio tone when you press a key. Want to prove it? Hit the ENTER key and line 10—which is all you've entered so far—will appear at the top of the screen. Now press the R key and the word RUN will appear at the lower left corner of your display. Press the ENTER key and you have RUN a program—albeit a rather short one. What did this "program" do? It entered the POKE into the computer memory—and from now on when you press a key it will BEEP when the computer acknowledges receiving it.

Continuing Line 10

Look at the screen. Blank, except for 0 OK, 10:1 at the bottom.

was the first statement in line 10. That's called a "report," and the T/S 2068 gives very meaningful reports. Now what? Well, you should continue line 10, since there's more to enter. Since line 10 was the last line entered or executed, it is the "current line," and you can EDIT it by simply holding down a CAPS SHIFT key and pressing 1 (which has the word EDIT above it). Line 10 appears at the bottom of the screen, with a blinking K-cursor following the line number.

Since you want to add more characters at the end of this line, you must move the cursor to the end of the line. You do this by holding down a CAPS SHIFT key and pressing the 8 key (which has a right-arrow above it). The K-cursor changes to an L-cursor and moves to the right as you press the 8 key. Hear the beep? Also, notice how the cursor jumps over the whole word POKE since it is a keyword? Move the cursor back and forth along the line with the left-arrow (5) and right-arrow (8) keys, remembering to hold down a CAPS SHIFT key. Finally, move the L-cursor to the right end of the line. Press the SYMBL SHIFT and Z keys and you'll get a colon (:). This tells the computer you want another statement on that line. The cursor, knowing that a keyword must be next, has changed to a K. How's that for smart?

Now press the L key for LET, then the P key. Hmmmm. That's a small p—lowercase. Don't worry about it. This is a "variable," and the T/S 2068 does not distinguish between uppercase and lowercase variables. Now press SYMBL SHIFT and L for the equals sign, then the 7 key. Now SYMBL SHIFT and Z for the colon, then SYMBL SHIFT and L for LET, followed by I, SYMBL SHIFT and L, then 0. Similarly, press the appropriate keys to program :LET b = 7, then press ENTER.

KEYSTROKE NOTATION

Whew! All that for one line? Seems tough at first. Let's summarize by using some simple notation. C-SHIFT will mean CAPS SHIFT, S-SHIFT will mean SYMBL SHIFT. We'll use commas to separate keystrokes. Line 10, using this notation, would be as follows:

```
1,0,0,2,3,6,0,9,S-SHIFT N,1,0,0,S-SHIFT Z,
L,P,S-SHIFT L,7,S-SHIFT Z,L,I,S-SHIFT L,0,S-SHIFT
Z,L,B,S-SHIFT L,7,ENTER
```

You've ENTERed four statements on one line, which, incidentally, is something that would take four program lines on a TS1000/ZX81

You have POKEd for keyboard feedback and set variables p, i, and b to initial values.

Line 20:

Press 2, then 0. Now we get into the use of the E-cursor. Hold down BOTH a C-SHIFT and the S-SHIFT keys and the K-cursor changes to an E-cursor for one keystroke only. Every time you want an E-cursor you must press BOTH shift keys. If you get an E-cursor by mistake, press both shift keys again and it will toggle back to whatever cursor it was. Now you have the E-cursor blinking at you, and you want the word PAPER, which is printed below the C key. "Below" means the S-SHIFT key is needed. Hold down S-SHIFT, press C and you've got PAPER on the screen. Now press P. Here are the rest of the keystrokes for line 20:

```
S-SHIFT Z,C-SHIFT S-SHIFT,S-SHIFT X,I,S-SHIFT Z,B
,B,ENTER
```

This line establishes the initial colors for the PAPER (user screen area), INK (printing), and BORDER (outside user screen area) with the values from line 10. As shown on the keyboard, 0 is black, 7 is white. Note that the legend on the B key says "BORDR," but the actual keyword is BORDER.

Line 30:

This line will print a flashing "prompt" on the screen when the program is run. The prompt is in "inverse video." Start with:

```
3,0,C-SHIFT S-SHIFT,S-SHIFT V,1,S-SHIFT Z,P,S-SHIFT
P
```

Now you need to get into inverse video. Look above key 4. There it is! Hold down the C-SHIFT key, press 4, and you are now in INV.VIDEO, even though nothing seems to have happened. To get back to TRUE VIDEO, you'll use C-SHIFT 3 later on. Here's the rest of line 30:

```
C-SHIFT P,O,S,I,T,I,V,E,SPACE,C-SHIFT
N,U,M,B,E,R,C-SHIFT 3,S-SHIFT C,S-SHIFT P,S-SHIFT
O,S-SHIFT Z,C-SHIFT S-SHIFT,S-SHIFT V,0,ENTER
```

Line 40:

This line allows you, when the program is running, to enter any positive number from the keyboard. Nothing is really new here. The keystrokes are:

4,0,I,A,S-SHIFT Z,P,A

Line 50:

This line lets you end the program by entering 0 or a negative number. The funny left-arrowhead-equals sign means "less than or equal to." The keystrokes are:

5,0,U,A,S-SHIFT Q,0,S-SHIFT G,G,2,0,0,ENTER

Line 60:

This line does the calculation of the square root.

6,0,L,X,S-SHIFT L,C-SHIFT S-SHIFT,H,A,ENTER

Note that the E-cursor is used to access the SQR above the H key, but *not* shifted since it's *above* the key.

Line 70:

This line prints the value of the square root and uses the graphic square that appears on the 8 key. The keystrokes for the beginning of the line are straightforward:

7,0,P,C-SHIFT S-SHIFT,P,2,S-SHIFT O,S-SHIFT P

But how do you get the graphic square on key 8? The graphic blocks shown on keys 1–8 are obtained in the graphics G-cursor mode. The black part of the graphic symbol is INK in the CAPS SHIFT graphic mode. Therefore, to get a solid block, we must use the 8 key with the S-SHIFT in the G-cursor mode. Without the shift key, we would get the inverse—a blank space. Confusing, isn't it? Actually, it appears someone at Timex goofed and inverted the graphic symbols on the keys, since you would expect them to represent the normal black INK on white PAPER. This could be changed in later production runs.

The word GRAPHICS is above the 9 key, so hold down the C-SHIFT key and press 9. The G-cursor appears. Now hold down S-SHIFT, press the 8 key, and there you have the solid block. But now you need to leave the graphics mode. Hold down C-SHIFT and press 9. The cursor returns to an L. Here's the rest of the line:

SPACE,C-SHIFT S,Q,U,A,R,E,SPACE,C-SHIFT
R,O,O,T,SPACE,C-SHIFT 9,S-SHIFT 8,C-SHIFT
9,SPACE,S-SHIFT L,SPACE,S-SHIFT P,S-SHIFT O,X,ENTER

Line 80:

This prints a dotted line across the screen, then a blank line. The keystrokes:

8,0,P,S-SHIFT P,S-SHIFT J (32 times),S-SHIFT
P,S-SHIFT Z,P,ENTER

(Note: With the T/S 2068 you can hold down any key and it will repeat after a short delay. Try doing this to enter the thirty-two dashes.)

Line 100:

What happened to line 90? There isn't any. You don't need to use all line numbers or place any special interval between them. "Modular programming" uses numbers in blocks. In this case lines 10 through 80 have really executed the whole program. Now the program will change the colors and do everything again. (On a black-and-white TV, different colors will appear as different shades of gray.) This program line decrements (decreases the value of) variable p by 1 and checks to see if it has gone below zero. If it has, it is reset to 7.

1,0,0,L,P,S-SHIFT L,P,S-SHIFT J,1,S-SHIFT
Z,U,P,S-SHIFT R,0,S-SHIFT G,L,P,S-SHIFT L,7

Lines 110 and 120:

These lines do the same sort of thing for variables i and b. The combination of lines 100 through 120 changes the PAPER, INK, and BORDER for each square root calculation. The keystrokes are so similar to line 100 that it's not necessary to repeat them.

Line 130:

This line produces a two-tone sound at the end of the calculation and before the colors change. Keystrokes are:

1,3,0,C-SHIFT S-SHIFT,S-SHIFT Z,S-SHIFT M,2,S-SHIFT N,0,S-SHIFT Z,C-SHIFT S-SHIFT,S-SHIFT Z,S-SHIFT M,2,S-SHIFT N,7,S-SHIFT Z,G,2,0,ENTER

The GO TO 20 returns the program for color changes and another calculation.

Line 200:

This line returns the colors to normal (PAPER white, INK black, BORDER white) before ending the program. You should be able to enter this by now without individual keystroke notation.

Line 210:

This line ends the program by printing END in graphic characters on the last user screen-line. The only tricky part here is that the same graphic symbol (key 6) is used both before and after the word END—but if you look close you'll see that they are the *reverse* of each other. This is done by using the S-SHIFT with the 6 key *before* the word END, and without the S-SHIFT *after* END. Type and ENTER this line for your "final exam."

RUNNING THE PROGRAM

Once you've entered the entire program, RUN it and watch the screen colors change with each calculation. Some of the PAPER and INK combinations are not ideal, but at least the program is relatively simple.

You can keep entering numbers all day long, until you exceed the limits of the computer (which is greater than 10 raised to the 38th power) or enter zero or a negative number—in which case program line 50 picks this up and routes the program to END. If you wish to interrupt the program, BREAK will not work, since the INPUT statement in line 40 is flashing an L-cursor waiting for your entry. The BREAK key only makes spaces when a numerical L-prompt is on

the screen. You can, however, press the SYMBL SHIFT and A keys for STOP.

Debugging

If the program doesn't RUN properly, pay attention to the "report" at the bottom of the screen, which will indicate—or at least give you a clue to—your error. You can check each program line with the LIST command on the K key. If you just press K and ENTER, the program will appear on the screen, and it will appear to be the whole program. But see the "scroll?" at the bottom. That means there's more. Press any key (except N, SPACE or STOP) and you'll see the ending quote on line 210.

You can check every program line this way, starting anywhere by using the line number after LIST. For example, LIST 80 would start with line 80. "Debugging" is great for learning programming, and it will keep you humble!

Editing the Bottom Lines

But what if you find an error? Either retype the line or EDIT it. By retyping it with the same line number you actually replace that line in the program when you press ENTER. But many times only one character needs to be changed, and that's when it makes more sense to EDIT. The T/S 2068 program editor is very effective once you get used to it, although it may initially seem strange if you have had experience editing lines on other computers.

Getting in the Mode

If you closely followed the instructions for entering line 10, then you've already had a little practice with the EDIT mode, since line 10 was then the "current line." But once a program is in place, using the EDIT mode takes some further explanation.

The EDIT mode is entered by pressing a CAPS SHIFT key and the 1 key. (See the word EDIT above the 1 key?) When you do this, the "current line" will appear at the bottom of the screen. What is the "current line"? The computer has its own set of rules to determine this, so it's easiest for you just to LIST the desired line number, which then makes *it* the current line. (By the way, in a long listing, this may cause a "scroll?" at the bottom, and you keep scrolling every time

you try to EDIT—until the end of the LISTing, or until you press CAPS SHIFT and BREAK.)

Another way to determine which is the current line is to look at the screen. There is usually (but not always) a "greater than" sign after one of the line numbers. This is the current line.

The Down and Up Arrows

If you want to make the current line another line nearby, you can press CAPS SHIFT and the down or up arrow (key 6 or 7) to move this "current line cursor" to another line, which then makes *that* the current line.

If all else fails, just press CAPS SHIFT and 1 and the current line, whatever it is, will be moved to the bottom of the screen. If it's not the one you want, just press ENTER and nothing has changed—except that you now know the current line!

Left and Right Arrows

OK. You have the line that you want to change at the bottom of the screen. You will need to move the blinking cursor, and you do this with the left and right arrow keys (5 and 8). This works exactly the same way as when correcting a new line before pressing ENTER. Just hold down the CAPS SHIFT and number key, and the cursor will move left with the 5 key and right with the 8 key. To DELETE the character (or keyword) to the left of the cursor, press CAPS SHIFT and 0, but remember that if the cursor is a K-cursor, the DELETE keyword needs to be overridden by holding down both the CAPS SHIFT and 0 keys until the auto-repeat takes over.

CARNEGIE HALL?

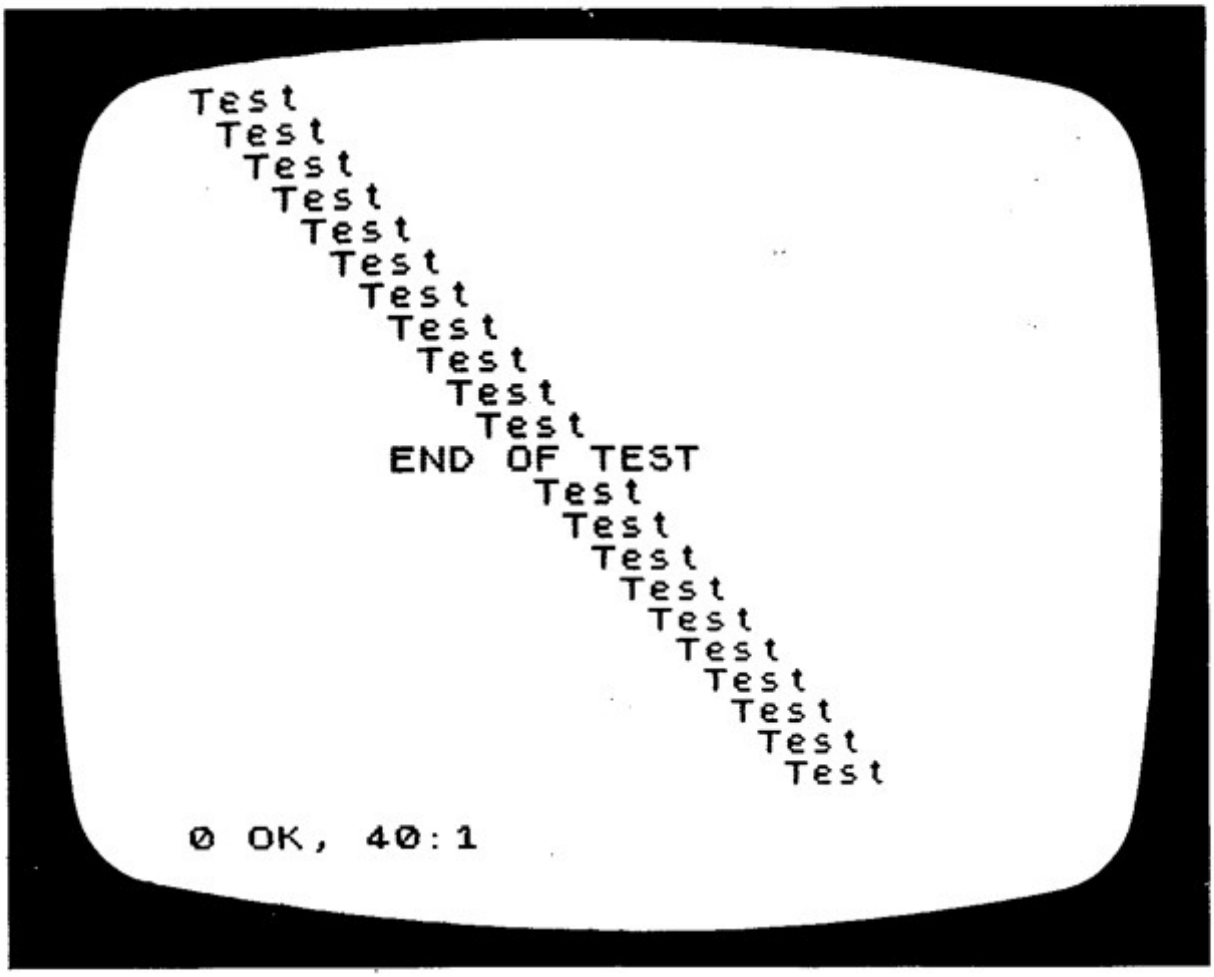
A stranger in New York City was hopelessly lost and asked an obviously cultured man, "How do you get to Carnegie Hall?" The man looked at him square in the eye and said, "Practice, my boy, practice!" And that's what you'll need to get comfortable with the T/S 2068 keyboard.

QUESTIONS

1. How do you access the legends that are within the black areas on the keys?
2. What cursor is necessary to access the SQR statement?

3. How would you access the keyword PAPER?
4. If you LIST a program, what is the "current line"—the first or the last line of the listing?
5. What's the maximum number the T/S 2068 can handle?
6. What BORDER color is the number 6?
7. When a K-cursor is on the screen, what happens when you press the 0 key? How about when you press CAPS SHIFT and 0?
8. What does the ?-cursor indicate?
9. How do you end this program when you're RUNning it?

3
Display Primer



Using PRINT AT.

INTRODUCTION

You've gotten your T/S 2068 hooked up and running, you've practiced using some of the commands, and have even written a few simple programs with INPUT, PRINT, and several other statements. You may have even done some calculations and had the results PRINT on the screen. But you've also discovered that computers seem to have a mind of their own about where they put things on the screen. The purpose of this chapter is to help you get control of your computer, and make it put things where you want them on the screen. You'll also find out how to do some simple graphics.

DISPLAY MODE 1

The display looks like a big blank area when there's nothing on it but the little blinking cursor in the lower left corner. Actually, there are four display modes available with the T/S 2068 Series computers. In this book we'll only be directly involved with Display Mode 1. The other three modes are discussed in detail in the second book in this series, *Timex 2068 Intermediate/Advanced Guide*.

Display Mode 1 provides 24 lines of 32 character locations each, with the bottom two lines reserved for INPUT, EDITing and reports. However, you can also PLOT individual points—called "pixels"—in any of 256 locations horizontally and 176 locations vertically. For those of you weaned on the ZX81 or T/S 1000 or 1500, this is *four times* as many locations in each direction, or *sixteen* times more pixel locations (45056 vs. 2816). All these locations are shown in Fig. 3-1.

Character and Pixel Locations

The first thing to understand is that in Display Mode 1, character locations are each eight pixels across by eight pixels high. That's why there are 256 pixel locations horizontally (32 characters × 8 pixels per character) and 176 pixel locations vertically (22 usable lines × 8 pixels per line). Another thing to understand is that *all* locations start with the first place designated as 0 rather than 1. That's why character locations on Fig. 3-1 go from 0 to 31 horizontally and 0 to 21 vertically, and pixel locations run from 0 to 255 across and 0 to 175 vertically. But that isn't all! Character locations start in the upper left corner, counting across and down. Pixel locations, on the other hand, reflecting the normal Cartesian coordinate standard, start at the lower *left* corner and count across and *up!*

Why do you need to know all of this? You don't—unless you want to control your screen displays. With a few simple commands and Fig. 3-1, you'll become the master. Read on.

To Scroll or Not to Scroll?

Type in this simple program to print on all twenty-two screen lines:

```
10 FOR x=1 TO 22
20 PRINT "Test"
30 NEXT x
```

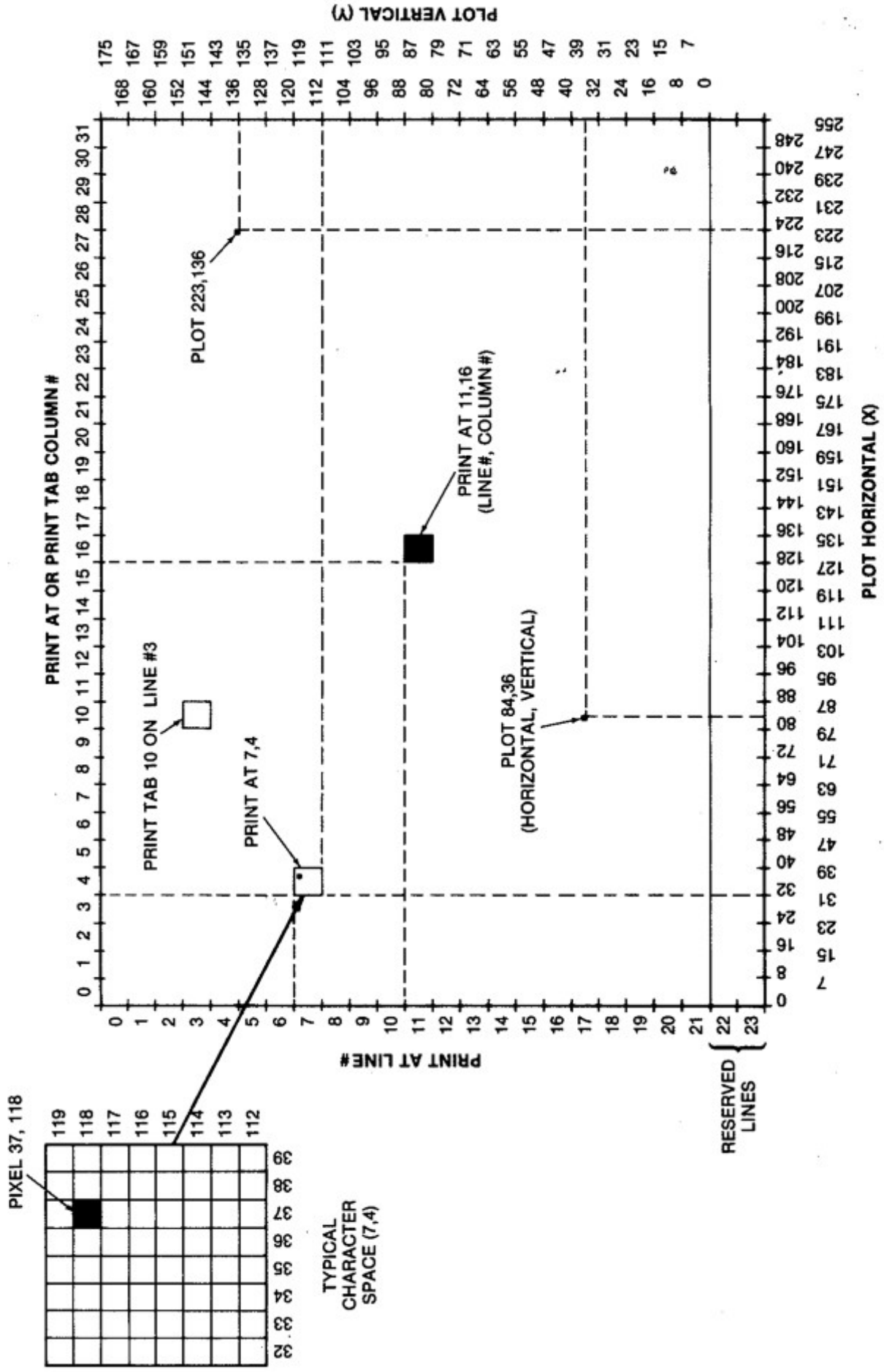


Fig. 3-1. Video display layout.

Of course, FOR, TO, PRINT, and NEXT are keywords, not separate characters. Now, when you RUN the program, the word "Test" (without the quotation marks) appears at the left side of the screen on all twenty-two lines.

Now, just to see what happens when you try to put more than twenty-two lines on the screen at a crack—or whenever the next printed line would fall below the last user screen-line—change the 22 in line 10 to 30. Now RUN and you get "Test" on twenty-two lines, than a blank line and "scroll?". Hmmmm. What to do to get the rest? Just press any key (except N, STOP, or SPACE) and the display continues and then stops with an OK report. Actually, the "scroll?" has been carefully planned to keep you from losing screen data unintentionally by having it scroll up and off the screen. If you don't want the "scroll?" at the end of a full screen, them simply add this line:

```
15 POKE 23692,255
```

Now when you RUN, the program goes to the end before it stops. The POKE simply tells the computer to PRINT 255 more lines before "scroll?". Since you've placed this in the program so that it is operative before the PRINT statement, the program is not even limited to 255 lines. It is reset to 255 every time you go through line 15. This may seem unimportant to you now, but it can be critical when printing long tabulations—and "scroll?" will also stop the printer if you're using one!

PRINT FORMATTING

Using the Comma and Semicolon

Two of the easiest-to-use and most powerful formatting commands are the comma and the semicolon. Just change line 20 by adding a comma after "Test." Now RUN, and you'll get two columns of the word "Test" running down the left and center of the screen, with fifteen in each column. What happens is that the comma at the end of line 20 moves the printing to the next half-screen width. If you use two commas, that moves the next printing a full screen line. As a matter of fact, just to illustrate how commas can be used to space printing several lines apart (with very little memory use), change the 30 in line 10 to 8, delete line 15, and use *six* commas at the end of line 20 instead of one comma. Now the program is simply this:

```
10 FOR x=1 TO 8
20 PRINT "Test",,,,,,
30 NEXT x
```

When this is RUN you get "Test" on display line numbers 0, 3, 6, 9, 12, 15, 18 and 21 (refer to Fig. 3-1 for display line numbers). The six commas at the end of the PRINT statement on line 20 of the program have merely created two blank lines each time. After the last line, the "scroll?" appears, since the commas are trying to print two more blank lines.

To see the effect of a semicolon, just remove the commas from the end of program line 20 and use one semicolon instead. Now RUN—and all the "Test" words are on one line, with no spaces at all between words! The semicolon at the end of a PRINT statement simply freezes the printing cursor at that location unless overridden by PRINT AT or PRINT TAB, which we'll get to soon.

But what about blank spaces? Can they be used to format? Absolutely! Change line 20 of the program to add four blank spaces after the word "Test," but BEFORE the closing quotation mark, like this:

```
20 PRINT "Test    ";
```

When you RUN the program each "Test" is separated by four blank spaces. Any text, including blank spaces, between quotation marks in a PRINT statement, will be printed, even if it forces the printing to the next line. A blank space uses a character space just like *any* other character.

Learn how to properly use commas, semicolons, and blank spaces, and you're on your way to "punctuation power."

Take the TAB Test!

Go back to line 20 in your program and change it to:

```
20 PRINT TAB 8;"Test"
```

The TAB function is above the P key. You need to first access the E-cursor, which you get by pressing one of the CAPS SHIFT keys and the SYMBL SHIFT key together. When you RUN this new line 20 (with the old lines 10 and 30), you'll get eight "Tests" all lined up one above the other, indented in to the eighth character space on each line. Now, change the 8 in line 20 to the letter x. When RUN, each line is indented one more space than the previous one, since

TAB now takes the value of x, which line 10 of the program increases by one each time through the FOR-NEXT loop (lines 10 and 30).

Multiple TAB statements are common. For example, use this for line 20 in the program:

```
20 PRINT "Take"; TAB 8;"the"; TAB 16;"TAB"; TAB
24;"test..."
```

RUN and see how nicely each word is spaced. Can you see the value of this in tabulations? Of course, its *real* value becomes apparent when you use variables. Use NEW (the A key) and ENTER to erase the current program in memory, and then type in this program:

```
10 LET a=1: LET b=2: LET c=3: LET d=4
20 PRINT a; TAB 8;b; TAB 16;c TAB 24;d
30 LET a=a+1: LET b=b+2: LET c=c+3: LET d=d+4
40 GO TO 20
```

RUN and watch the four columns of figures march down the screen, with the first column counting by one, the second by two, the third by three, and the fourth by four—neatly formatted by the TAB statements. Of course, you can combine text with the variables in TAB statements. When the last line prints, the familiar "scroll?" appears. How would you defeat that? Just add:

```
35 POKE 23692,255
```

Using PRINT AT

If you think that TAB is something, wait until you get familiar with PRINT AT! You'll need to refer to Fig. 3-1 again to follow this. PRINT AT simply starts printing at a location down and *across* from the upper left corner. You specify the coordinates with *line* (down) first, and then *column* (across), like this:

```
PRINT AT 20,15
```

This will cause the printing to start on the twenty-first line down and the sixteenth space across. (Remember, zero is a number in computing and is used where you normally would start with one. Also, "lines" are frequently referred to as "rows.") To illustrate how

PRINT AT can be used, delete your existing program with NEW and ENTER this program:

```
10 LET y=0:FOR x=0 TO 21
20 PRINT AT y,x;"Test"
30 LET y=y+1: NEXT x
```

Be sure to use the SYMBL SHIFT and I key to get AT. RUN this and you'll get a string of "Test" words running diagonally down the screen. But to illustrate how PRINT AT can jump around the screen, and even erase previous printing, add line 40:

```
40 PRINT AT 11,7;"END OF TEST!"
```

See how it overprints the twelfth line (screen line #11)? Using blank spaces between the quotes, you can erase anywhere on the screen.

Displaying the Entire Character Set

You might want to see what each number means to the computer when it's used in the "character set." This is accessed with the CHR$ statement. This program can be used to print each CHR$ code on the screen and on the printer. If you don't have a printer, or it's not turned on, the LPRINT commands will just be ignored. Type this in after NEW:

```
10 FOR x=32 TO 255 STEP 2: POKE 23692,255
20 PRINT x;" "; CHR$ x,x+1;" "; CHR$ (x+1)
30 LPRINT x;" "; CHR$ x,x+1;" "; CHR$ (x+1)
40 PRINT : LPRINT : NEXT x
```

A couple of things are significant here. For one thing, most numbers under 32 represent "control codes." Not only will they *not* print out, some will stop the program—so start with 32, which is a blank space. Another significant fact is that the numbers and letters are standard ASCII (pronounced ASK-ee, and stands for American Standard Code for Information Exchange), unlike the T/S 1000/1500/ZX81. This is of prime importance when interfacing the computer to peripherals, such as printers, plotters, modems and so forth. When you RUN the program, the comma in lines 20 and 30 "zones" the printing so you get two columns, each one-half screen width. Notice

the graphics from 128 (blank space, like 32) to 143. What appear to be a repeat of uppercase letters A to U (numbers 144 to 164) are actually locations where you can design and program any character shape to fit into an 8 × 8 matrix. (This is covered in detail in Chapter 9, "Grappling With Graphics.") Notice that the keywords run from 165 to 255; did you realize there were so many?

HIGH RESOLUTION GRAPHICS

There are three ways to produce graphics with the T/S 2068. The "low-resolution graphics" are those available directly from the keyboard—the eight squares on number keys 1–8, and their inverse video counterparts (when the SYMBL SHIFT key is held down, or INV.VIDEO is used). The so-called "high-resolution graphics" are the pixels referred to earlier. The "ultra-high resolution color graphics" are only available in Display Mode 4 and are discussed in detail in the *Timex 2068 Intermediate/Advanced Guide.* In the book you're reading, Chapter 9 goes into detail on the low- and high-resolution graphics. However, since they play such a large part in screen displays, a short "briefing" follows.

Getting Pixel-ated

Fig. 3-1 shows the coordinates used for pixels (picture elements—individual dots). The important thing here is that pixel coordinates are specified just like conventional graphs, with the x-axis horizontal, running left to right, and the y-axis being the left vertical side, running upwards. This is entirely different from the PRINT AT coordinates. It takes some getting used to, but makes sense when you think of it this way: PRINT AT is intended for text, which extends from top to bottom, left to right; pixels, on the other hand, are graphical in nature, and follow x-y graph convention. Furthermore, since there are 64 pixels (8 x 8) in each character space the numbering system is entirely different. To illustrate the use of pixels, delete the existing program with NEW and type in this program:

```
10 LET y=0: FOR x=1 TO 175
20 PLOT x,y
30 LET y=y+1: NEXT x
40 LET y=0: FOR x=1 to 175
50 PLOT INVERSE 1;x,y
```

```
60 LET y=y+1: NEXT x
70 GO TO 10
```

When you RUN this program, a thin line is drawn, one pixel at a time, from the lower left corner of the display to the top of the display (but only 176 pixel locations to the right, not the entire 256 spaces), and then the line is erased, pixel by pixel. (The T/S 2068 does not have the UNPLOT statement used in the TS1000/1500/ZX81, but "PLOT INVERSE 1;" amounts to the same thing, though it's a little trickier to use).

Just for a little fun with sound (although it will slow down the program), add this:

```
45 BEEP .1,x/8
```

OK, Pardner—DRAW!

To give you a taste of the power of just one other graphical command, press the A key (NEW) and ENTER. Now type and ENTER this short program:

```
10 PLOT 0,0: DRAW 175,175
30 PLOT INVERSE 1;0,0: DRAW INVERSE 1;175,175
50 GO TO 10
```

Very simply stated, "PLOT putteth and PLOT INVERSE taketh away." DRAW almost instantly creates a line from the PLOT starting point to an ending point x pixels to the right and y pixels up. Zap! PLOT INVERSE and DRAW INVERSE act like an eraser. More details on these are in Chapter 9. For now, press R for RUN and ENTER and you get a flashing line in exactly the same position as the previous program. Actually, the line is being "drawn" and erased so quickly it appears to be flashing. To slow it down, so you can see how fast the DRAW command works, put in the following two lines:

```
20 PAUSE 30
40 PAUSE 30
```

The PAUSE is just that. The computer counts to itself the number of TV frames going by (60 per second) and does nothing new in between—so PAUSE 30 gives you about a half-second delay. (Don't depend on PAUSE for precise timing, however, since computing time is involved, as well as TV frames.)

QUESTIONS

1. Of the four Display Modes, which is the only one described in this chapter? Where is the information on the other three display modes?
2. In Display Mode 1, how many characters are on a line? How many lines? How many pixels per line? How many lines of pixels?
3. What is "pixels" a contraction of? How many pixels are there in Display Mode 1?
4. What does POKE 23692,255 do?
5. What is the effect of a semicolon at the end of a PRINT statement? How about a comma? How about four commas?
6. Is a "column" horizontal or vertical? How about a "line" or "row"?
7. What line and column will be specified by PRINT AT 3,4?
8. What position on the screen is specified by PLOT 10,20?

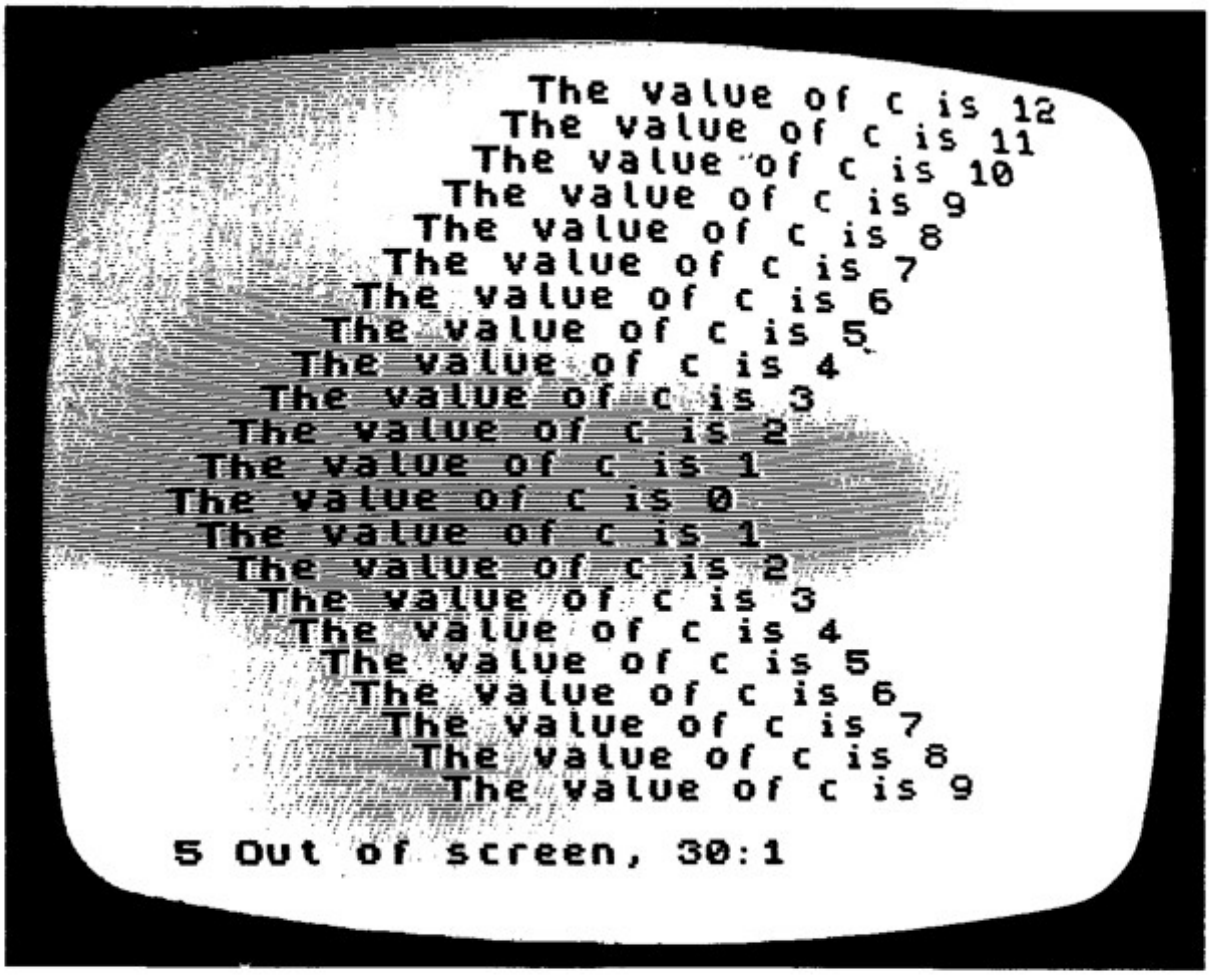
SECTION B
THE BASICS OF BASIC

Now that you've been exposed to some vicious jungle mosquito "bytes," and probably have a touch of "megabyte malaria," it's time you learned a little more about the tactics and strategy of survival in the T/S 2068 software world—the BASIC language (and, in particular, the T/S 2068 dialect). Although sign language will suffice with the local natives, it won't work with computers, so if you are going to communicate effectively with the T/S 2068 you'll have to learn to speak its language.

Chapter 4 covers the most-used statements and commands of Timex 2068 BASIC. Chapter 5 gets into the powerful programming techniques of loops, branching, and subroutines. Chapter 6 covers string handling, mathematics, and many miscellaneous commands. Chapter 7 deals with the sophistication of arrays and data manipulation.

This section of the jungle is full of traps for the unwary. It's a tough jungle. There will be times when you'll want to turn back. Don't, because beyond this jungle (in Section C) we get to "The Valley of Sight and Sound"—color, graphics and music. (Caution: Attempting to fly over this jungle will probably result in your crashing in "The Valley of the Lost"!)

4
Essential Functions, Commands, and Statements



Using variables with PRINT AT.

INTRODUCTION

There are a couple of dozen keywords that you'll use most frequently in programming the T/S 2068, and you'll find these scattered throughout BASIC program LISTings in books and programs you buy on tape. Before going into the less frequently used words, this chapter will get you familiar with some of the most common BASIC words and cover procedures for SAVEing and LOADing programs from tape.

Incidentally, you'll find that I use the words "command," "state-

ment," "keyword," "function," and "instruction" almost interchangeably when referring to BASIC words. However, some purists insist that "command" is the proper term for keyboard entry, and "statement" is the proper term when used within a program. Since these are fine distinctions, and sometimes awkward, I won't be terribly strict in my usage. Out here in the jungle, anything goes! (If you think BASIC is a tough language, try Swahili.)

PROGRAM LISTING AND DELETING

In the course of presenting the material in the preceding chapters, several short programs were used as examples. A program is merely a set of instructions for the computer, arranged in sequence by "line numbers." Whenever a new instruction is placed in a program, it either follows an existing statement ending in a colon, or it begins with a new line number. Line numbers in the T/S 2068 can be any whole number from 0 to 9999. Program lines will automatically arrange themselves in increasing number order, regardless of when the line is typed into the computer. In other words, if you already have program lines 10 and 20 and you wish to add another line in between, you can type in line 15 (or any number from 11 to 19) and it will slide right into position between lines 10 and 20.

Line Numbering

You may wonder why so many BASIC programs you see in books and magazines have line numbers that seem to increment by tens. That's become convention, and it leaves line numbers available for the programmer to use if necessary. Actually, lines can be numbered one apart, but only whole numbers can be used. The computer won't allow a line number such as 10.3 or 11.5. Although the T/S 2068 does not have an automatic line number generator, or a built-in program renumbering command, you can be sure that "utility programs" to perform these kinds of programming tasks will be forthcoming in software. So, for now, when writing your own programs, use line numbers 10 or 20 apart, and give yourself room for the inevitable program fixes and improvements.

LISTing the Program

The lines of the program in memory are called to the screen with the LIST command. If you simply use the keyword LIST (on the

K key) and press ENTER, the resident BASIC program lines will LIST on the screen in number order. If there are more than twenty-two lines (notice, lines—not line numbers—since some program line numbers may use several screen lines) then the query "scroll?" will appear at the bottom of the screen. Just press ENTER, Y, or any key (except N, STOP, SPACE, or BREAK) to continue the LISTing. When you reach the end of the program LISTing, you'll see a screen report of 0 OK, 0:1 at the bottom of the screen.

If you wish to view only a portion of the program LISTing, then use the starting line number after LIST, such as LIST 120, and the screen LISTing will start with that line number. This is particularly useful when you want to EDIT or duplicate a line, since this technique makes the designated line the "current line." See the last part of Chapter 2 for more details on the EDIT function.

DELETEing Program Lines

The DELETE function (CAPS SHIFT and 0 key) on the T/S 2068, as mentioned briefly in Chapter 2, has two distinct uses. When the K-cursor is on the screen, then DELETE is a keyword used to DELETE entire program sections. Suppose, for example, that you wish to DELETE lines 50–90 from your program. You could do so by typing each line number and pressing the ENTER key, thus eliminating each line one at a time. However, you could also use the command DELETE 50,100 and wipe out *all* those lines at once. But be careful, since DELETE ,100 will kill all lines up to and including line 100, and DELETE 50, will eliminate line 50 and all the rest of the lines to the end of the program.

If, however, the L-cursor or C-cursor is on the screen, then pressing the CAPS SHIFT and 0 keys DELETEs the first character to the left of the cursor. The function performed by DELETE depends on the cursor mode. Each time you try to DELETE a character the computer checks to see what cursor mode it's in. If the keyword DELETE appears on the screen, and you are trying to backspace and erase, then you'll need to hold down the CAPS SHIFT and 0 keys until the auto-repeat function takes over and DELETEs the keyword DELETE. This takes nimble fingers until you get used to it.

PRINTING COMMANDS

Probably the most-used keyword in BASIC is PRINT. It has

is simply an instruction to the computer to direct the output—whatever you want PRINTed—to the display. PRINT may be used from the keyboard by pressing the P key when the K-cursor is on the screen. It is usually followed by a letter or letters representing some value (a "variable"), or by a "string" in quotation marks. A "string" is simply one or more characters you wish to appear on the screen. For example, type this on your keyboard and then press ENTER:

```
PRINT "I'm a fantastic computer!"
```

If you can't figure out the keys to use, GOTO Chapter 2, "Learning the Keyboard." Actually, the T/S 2068 is hard to fool, and it won't let you make too many mistakes. For example, if you try to type the word "PRINT" with individual letters, you'll find the first keystroke, P, gives you the whole word automatically (assuming the K-cursor—or a "Report Statement," which actually overlays a K-cursor—is on the screen in the lower left corner).

Anyhow, assuming you also found the quotation marks on the P key, the apostrophe on the 7 key, and the exclamation point on the 1 key, you should be seeing "I'm a fantastic computer!" (without the quotation marks) on the top line of your screen. The point of all this is to show you that if you put anything in quotation marks after a PRINT command, it will appear exactly that way on the screen.

PRINT Formatting

If you use PRINT with nothing after it, it will PRINT a blank line. Normally, after any PRINT statement a "carriage return" (a term passed on from typewriter terminology) is automatically sent to place the next PRINTing at the beginning of the next line. However, a semicolon suppresses the carriage return completely, and the next PRINTing immediately follows the last PRINTing—unless it's at the end of a line, in which case the computer forces the PRINTing to "wrap around" to the beginning of the next line.

If a comma follows the PRINT statement, the following PRINTing will be at the next half-screen width—and that may be the next line. For that reason, multiple commas can be used as "line-feeds." For example, try this:

```
PRINT "testing",,,,,,"commas"
```

The six commas each act like one-half a line-feed character, so six commas act like three line-feeds, and the word "commas"

Similarly, you can use the apostrophe after a PRINT statement to produce line-feeds, one for each. Put six apostrophes in place of the commas in the above PRINT command and you'll see the word "commas" print six lines below the word "testing." While the apostrophe used this way doesn't save memory space, since keywords (or "tokens" as they are frequently called) only take as much memory space as a single character, it can save screen space.

It is important to realize that the comma and apostrophe used *within* quotes after a PRINT command will appear as themselves and will have no effect on PRINT formatting.

PRINT AT and PRINT TAB

The whole subject of screen formatting for characters and graphics was detailed in Chapter 3, "Display Primer." There is a great tendency for beginning computerists to jump ahead of these powerful formatting commands into more sophisticated commands. Actually, almost as important as the information the computer provides is the manner in which the results are presented. If you're not clear on how AT and TAB are used with PRINT, go back to Chapter 2. They are very useful if you want to create professional-looking programs and allow you great flexibility in your screen presentations.

PRINTER COMMANDS

If you have a Timex Sinclair 2040 Personal Printer plugged onto the back of your T/S 2068, there are three more commands available to you which relate to PRINTing and LISTing. The most useful is LLIST (V key in E-cursor mode). This will LIST your entire program (without stopping for "scroll?") The printout will be in the same 32-column format as the screen. If you have interfaced a different printer to the 2068, the printout will probably be different than the screen, since most printers use 40 or 80 or more characters on a line.

The statement LPRINT is normally used within a program to direct output to the printer instead of the display. It is quite common to see a PRINT statement in a program immediately followed by an identical LPRINT statement, so that the user gets output on both the screen and the printer.

The commas and apostrophes, when used for PRINT formatting, do exactly the same on the printer as on the screen. LPRINT TAB is also valid, but LPRINT AT will simply place the printing on the next

printer line. How, then, can you get an accurate printer output of a program that uses a lot of PRINT ATs? That's one of the purposes of the COPY command, which will duplicate on the printer *every dot* on the screen. This is particularly useful for low- and high-resolution graphics. Be aware, however, that the printer is "color blind" and will print only INK dots (even if they are printed on the same color PAPER and invisible on the screen). This means that LISTings in color (yes, each character in a LISTing can be in any of the T/S 2068 PAPER and INK colors), colored graphics and colored text may print differently than they appear on the screen. The printer, to repeat, only "sees" INK dots.

One very nice feature of the T/S 2068 is that it does not normally "lock-up" or "freeze" if you give it a printer command and there's no printer attached. The 2068 simply spits out the information to the rear card edge and then comes back for the next instruction (but a lot faster than when a printer is attached, since the printer can't keep up with the computer output and keeps feeding the 2068 "interrupts").

THE INVARIABLE VARIABLES

There are hardly any computer programs in which variables are not used. There are different kinds of variables—"numeric variables" (sometimes called "simple variables") and "string variables." We'll deal with the string variables in Chapter 7. For now, we've got enough to handle with numeric variables, which, as the name implies, are numbers. But since these numbers change as a result of conditions or calculations, it makes sense to refer to them with some kind of label that doesn't change—so we use "variable names." These usually consist of one or more letters, and some programmers love to assign whole words to variables, like "dollars" instead of "d." I was brought up on a computer that only allowed one-character variables and had very little memory space, so I still tend to stick with that approach.

Using Variables

The T/S 2068 requires the use of the LET or INPUT statements to assign variable names and values. If the program refers to a variable for PRINTing, LPRINTing, or any sort of calculation or comparison before the variable has been assigned by use of INPUT or LET, the

program will stop with an error. Type and ENTER program LISTing 4-1 to illustrate the use of both the LET and INPUT statements (and a couple of new statements as well).

The program starts in line 10 by setting the variable l equal to 0. That's simple enough. LET l=0. But what is "l"? It can be whatever we want it to be, but just in case we forget, a REMark statement follows on line 10 after a colon. The colon is merely a "delimiter" which tells the computer that another separate statement is coming next on this program line instead of being on the next program line. The T/S 2068 (unlike the T/S 1000 or 1500) allows "multiple statement lines" with colons between the statements. The REMark statement, which is totally ignored by the computer, allows the human operator to note anything worth noting. In this case, the variable l is defined as standing for "line."

LISTing 4-1

```
10 LET l=0: REM l is line
20 INPUT "Column? (12 Max.)";c
: REM c is column
30 PRINT AT l,c;"The value of
c is ";c
40 LET l=l+1: GO TO 20
```

Line 20 not only illustrates the INPUT statement, but shows how it can be used to ask a question and define limits. It asks the operator (that's you) to specify a column number up to a maximum of 12, and then assigns that value to the variable c. After the colon, the REMark tells you the c stands for "column."

Line 30 tells the computer to PRINT AT line l and column c, "The value of c is ," and then the value of the variable c. Notice the space after the word "is," since the semicolon following the quotation mark means the next PRINTing will be in the next character location. Without the space after "is" the value of c would be jammed up right against "is." Also notice that the "c" inside the quotation marks is printed exactly that way, while the "c" after the quote tells the computer to print the *value* of the variable we're calling c.

Line 40 adds one to the line variable l so that the next PRINT AT will be on the next screen line, then GOes back TO program line 20 to ask you to INPUT another value for c.

RUN the Program

When you press R for the keyword RUN, and then press ENTER, a number of things happen. For one thing, the screen clears. Internally, all variables are "unassigned," and the program starts at the lowest line number. The INPUT request appears at the bottom line of the screen and everything stops (except the blinking of the L-cursor) until you press a number and hit ENTER. If you try to ENTER a letter, the program stops with an error statement, since the INPUT statement specified a numeric variable, c. You can also press SYMBL SHIFT and A for the STOP command and then press ENTER to actually STOP the program. The BREAK key, normally used with CAPS SHIFT to interrupt a program, will not BREAK an INPUT request. Instead, it just makes a space! Furthermore, you cannot use the CONTinue command, normally used to CONTinue program execution after a BREAK, since BREAK was not invoked.

Enter values from 0 to 12 from the keyboard in any sequence you desire, following each with ENTER. Each time you do this you get a line of printing showing the value you used.

The results of all this, using values for c from 0 to 12 and then back to 3, are shown in Fig. 4-1. The program "crashes" at this point because the value of 1 has increased to 22, a figure larger than the

```
The value of c is 0
 The value of c is 1
  The value of c is 2
   The value of c is 3
    The value of c is 4
     The value of c is 5
      The value of c is 6
       The value of c is 7
        The value of c is 8
         The value of c is 9
          The value of c is 10
           The value of c is 11
            The value of c is 12
           The value of c is 11
          The value of c is 10
         The value of c is 9
        The value of c is 8
       The value of c is 7
      The value of c is 6
     The value of c is 5
    The value of c is 4
   The value of c is 3
```

Fig. 4-1. RUN of LISTing 4-1.

PRINT AT statement allows. (PRINT AT maximums are 21 down and 31 across.)

Incidentally, you can RUN a program starting at any line number by using the line number after the RUN command, but this may "bypass" the setting of some variables and cause a program "crash." Usually this is most useful when you write a program and wish to test portions of the program without going through earlier portions.

TAPE INPUT/OUTPUT

When you turn off the T/S 2068 you "lose" the program in memory. There are various devices available for storing programs on microcomputers, but at this time only two systems are used with the T/S 2068—cassette and cartridge. A third system, the "Micro Drive," is promised later. The Timex Command Cartridges hold ready-to-go programs in a small 2½-inch square housing that plugs into an opening on the right side of the 2068 (with the computer turned off. Never insert or remove cartridges with the power on!). However, you can't save your own programs on these cartridges. The only means of storing your own programs presently available is a standard audio cassette tape recorder.

The 2068 has a cassette "port" that is both fast and efficient, running at "1500 baud"—which means about 150 characters can be transferred to or from the 2068 in one second. In actual use, some time is lost in "leaders" that synchronize program transfer and identify programs, so the actual total time to LOAD or SAVE a program is not strictly based on the number of characters in the program.

You do not need a fancy cassette recorder to SAVE and LOAD programs. As a matter of fact, the cheaper ones seem to work best. However, try to use one with a digital counter since that will allow you to "fast forward" or "rewind" to specific programs on a single tape.

The recorder playback volume is important when LOADing a program from a cassette tape into the 2068. Also, it is very important that the EAR cable be connected to both the recorder and the 2068, and the MIC cable must be disconnected at one end or the other. If both cables are connected, a feedback loop can prevent proper LOADing.

Similarly, when SAVEing a program from the 2068 to the recorder, only connect the MIC cable at both ends and disconnect the EAR

cable at either end. The volume setting when SAVEing is probably not important, since most modern cassette recorders have automatic recording level control. If, by chance, your recorder does control recording volume manually, you'll need to experiment to find the proper volume control setting.

Getting LOADed

Probably before you SAVE your own programs, you'll be LOADing programs from a cassette you purchased or borrowed, so let's cover LOADing first.

First be sure the tape is for the Sinclair Spectrum or the T/S 2068. Tapes made for the T/S 1000 or 1500 (or ZX81) will *not* LOAD into a 2068. The Spectrum, however, uses the same tape format as the 2068, and the language is a major subset of the 2068 language. While some Spectrum programs will not RUN on the 2068, many will RUN normally, and all should LOAD.

The command to LOAD the next program on the cassette is simply LOAD "". Do not put a space between the quotation marks. When you press ENTER, the bottom screen line goes blank. Now turn on the recorder with only the EAR cable connected between the recorder and computer. The sequence is then as follows:

1. The border around the display area starts alternating in color between red and blue as the computer "listens" for the beginning of the program.
2. When the program leader is found, the border changes to narrow horizontal red and blue stripes that move downward rapidly for about two seconds, followed by a burst of blue and yellow stripes. Then the word "Program:" appears, followed by the program name (if it was assigned a file name when it was SAVEd).
3. After a short delay, you'll see a short burst of red and blue stripes, followed by narrow blue and yellow stripes quivering in the border area. This is the actual program information, and it runs until all the information is transferred into computer memory.
4. The report 0 OK, 0:1 then appears on the bottom line if the program has LOADed properly. Don't forget to shut off your recorder. Unlike many computers, the 2068 does *not* control the starting and stopping of the recorder.
5. An error report means the tape did not LOAD properly—try

again, using a different volume level. The correct volume may be higher or lower, and will vary considerably from recorder to recorder.

Generally, you'll have the best success LOADing programs you have SAVEd on the same recorder. Purchased programs are the toughest to LOAD. In that case you're dealing not only with volume differences, but head alignment (called "azimuth") and the general differences in recorder amplifier characteristics that can cause you considerable woes. Fortunately, the tape recording system used by the 2068 is completely different from that used in the 1000 and 1500; it is six times faster, yet far more forgiving.

Finding a Specific Program

If you have several programs on the tape, and you wish to LOAD a particular program, then you must either LOAD each program, one after the other, until you have the one you want, or else use the LOAD command with a "file name." The file name must be exactly the one used when the program was SAVEd, including upper and lowercase characters. For example, if you try LOAD "sample" and the program name is "Sample" (with a capital S), the computer will show the actual program name on the screen, but will not LOAD it. It will go on to the next program, still looking for "sample" with a lowercase s. You can exit from the LOAD command with the BREAK key.

Now you can see the advantage of having a recorder with a digital counter. You can "fast forward" the tape to the program you want. Of course, this assumes you have the discipline to put a label on the tape showing the program locations, or keep a directory in a notebook with this information.

MERGEing Programs

Sometimes you may wish to combine two BASIC programs, putting one after the other. This is particularly useful if you have a library of common "subroutines" on tape—perhaps a BASIC sort or search routine. So long as the line numbers are higher than those in the program already in memory, you can use the MERGE command instead of LOAD. If any line numbers of the MERGE program are the same as the resident program, the MERGE line will replace the resident one. This can cause all kinds of havoc, so first verify that

the line numbers of the MERGE program are all higher. For this reason, it's wise to write subroutines in high line numbers.

What if the MERGE program doesn't have higher numbers? You simply renumber it with the EDIT function, which can also be used to change line numbers.

SAVEing a Program

Now suppose you have a BASIC program in memory you'd like to SAVE on tape. Be sure only the MIC cable is connected at the recorder and the computer. If all you want to do is SAVE the program and the values of the variables, just use SAVE "name" and press ENTER. The screen will prompt you with "Start tape, then press any key." Do that with the recorder in the "record" mode. You'll see patterns of red and blue stripes, then dark blue and yellow stripes in the border area as the program identification and parameters, and then the program and variables, are put on the tape. The report 0 OK 0:1 appears on the bottom of the screen. You must then stop the recorder manually.

Most of the time you'll have no need to SAVE the variables that result from RUNning the program. They are SAVEd automatically and take up both time and tape. To eliminate them, press the X key for CLEAR and then press ENTER before the SAVE command is typed. This resets all variables, arrays, and practically everything else, but does not disturb the BASIC program in memory (and, for you machine language types, does not change RAMTOP protected memory).

If you'd like the program to LOAD and RUN automatically, use SAVE "name" LINE 0 and after it loads, it will start RUNning at the first program line. If, for some reason, you wish to start somewhere within the program, use that line number. The LINE command is obtained by using the 3 key in the E-cursor mode with the SYMBL SHIFT key.

If you just plain don't care about a file name for your program, you can use SAVE " " to SAVE it, but be sure to put a space between the quotation marks, otherwise you'll get an error message when you press ENTER.

It is also possible to SAVE the data in arrays or any block of memory (including the screen display) using the keywords DATA, CODE, and SCREEN$, respectively. These are not commonly used, and the methods for doing so are described in the Timex User Manual.

VERIFYing Your SAVE

The VERIFY command (SYMBL SHIFT and R keys in E-cursor mode) is used to be sure the recorded SAVE is good. Change the EAR and MIC cables so only the EAR is connected, and use VERIFY"" (no space between quotes) after rewinding the tape to the program beginning. The program name and OK report code will confirm a good SAVE.

CLEANING THE SLATE

The command CLS (V key) is handy to clear the screen anytime, either directly from the keyboard or in a program. Frequently, the screen is cluttered with extraneous printing, or you are getting near the bottom of the screen. A CLS wipes the screen clean and the next PRINT will appear on the top line (unless otherwise directed).

The NEW command (on the A key) is used when you want to erase the current BASIC program in memory, but not interfere with special routines you may have reserved above RAMTOP.

(RAMTOP? What's that? That's the second time it's mentioned without explanation. What's going on here? I want my money back!)

RAMTOP is merely the designation for the highest memory address used by the BASIC system. Changing this "ceiling" is primarily of interest to machine-language programmers. The point is that NEW does not change this. When NEW is used, the display part of the screen goes black for a second and the BASIC program is gone. (Actually, it's still there, but it takes special techniques to recover it.)

The CLEAR command (X key) is used with a memory address following it (such as CLEAR 40000) to change RAMTOP. CLEAR used without an address does various other housekeeping tasks, but does not alter RAMTOP.

Other Commands and Statements

There are ninety-one keywords (not counting punctuation or symbols) available in the T/S 2068 BASIC repertoire. We've covered quite a few of the more common ones here, but not enough to allow you to do any significant programming. The emphasis in this chapter has been on operational commands more than programming. The keywords covered in this chapter won't even allow you to calculate

the rate of approach of a charging herd of elephants (which information I need right now, judging from that dust cloud in the distance . . .). However, the following chapters will get into some useful programming (if the elephants don't get here first!).

QUESTIONS

1. What are the minimum and maximum line numbers the T/S 2068 will accept? Can you use decimal line numbers?
2. What is the command to show the program starting at line 50? What are the advantages of this?
3. What is the command to remove line numbers 100 to 200, inclusive, from the resident program?
4. If you use four commas after a PRINT statement, how many screen or printer lines will be skipped? What if you use four apostrophes?
5. What command directs the entire contents of the screen to the printer?
6. What happens if you use LLIST or LPRINT and no printer is attached, or the printer is not on?
7. How would you define a "variable"? What kinds of variables are there?
8. When SAVEing a program on tape, which cable is connected from the recorder to the computer? When LOADing?
9. What command is used to append one program to another? What must you be careful of?

5
Loops, SUBroutines and Branching

Photo 5-1. RUNning the digital timer program.

INTRODUCTION

One of the things a computer excels at is doing the same thing over and over. Unlike us humans, the microcircuitry is actually working all the time, constantly scanning the keyboard and looking through its instructions for something to do. The computer doesn't get more "tired" doing something than doing nothing, so you might as well keep it busy.

In this chapter we'll examine repetitive operations, logical and quantitative comparisons, and routing the instructions based on

computer decisions. Sounds pretty exotic, eh? Well, actually, if you get a good "hang" on the contents of this chapter, you will have made a giant step into BASIC programming. Some of this is more confusing to explain than to do—unlike speaking some of this jungle lingo, which is difficult to do but easy to explain: "Here a grunt, there a grunt, everywhere a grunt, grunt."

HOW TO DO A LOOP

In my younger days, as a Naval Aviator, acrobatic flight was part of our training. A properly executed "loop" brought you right back where you started, even though part of the way you were on your back and the whole world was inverted. "Looping" your computer is no strain for either you or the computer, and you don't even have to wear a seat belt. Actually, like an airplane loop, a computer loop returns you to the same spot except the computer deals with program line numbers instead of points in space.

LISTing 5-1 shows a simple program that counts from 0 to 50 and then back to 0, then does it again. The numbers are displayed on the screen, although they'll be whizzing by so fast you'll need to uncage your eyeballs to identify them. Although this is a purposeless program in terms of accomplishing any real world function, it illustrates a number of new points, and you really should ENTER it into your T/S 2068 to get the full benefit of the following discussion. C'mon, don't be lazy . . . . . type it in and then press the R key for RUN and press ENTER.

Incrementing and Decrementing

It seems that everywhere in the computer world, technical words have supplanted more common terms. We "increment" in computing,

LISTing 5-1

```
 10 LET x=0
 20 PRINT x
 30 LET x=x+1
 35 IF x>50 THEN GO TO 60
 40 POKE 23692,255
 50 GO TO 20
 60 LET x=x-1
 70 IF x<0 THEN GO TO 10
 80 POKE 23692,255
 90 PRINT x
100 GO TO 60
```

but "add" in real life. Similarly, computers "decrement" and humans "subtract."

Line 10 sets the value of the variable x to zero. We could call this variable by any other letter, or by any combination of letters, like "variable," or even "this is a variable." The T/S 2068 has no apparent restrictions on variable labels—but be aware that once you label a variable you must refer to it by that name everywhere it's used. So, we'll keep it short and just use "x."

Incidentally, the 2068 will not distinguish between uppercase and lowercase letters in variable designations; "X" is the same as "x" as far as the 2068 goes. This is not true of all computers, however, so don't get sloppy! In this book, we'll generally use lowercase letters to designate variables to avoid the extra effort of pressing the CAPS SHIFT key. (How lazy can you get?)

Line 20 PRINTs the value of x, which is zero at this point. Since you started the program with RUN, the screen clears and the first PRINTing occurs (unless otherwise specified) in the upper left corner.

Now line 30 says, "LET x=x+1." Huh? Algebra was never like this! Ah, but you weren't using BASIC programming in algebra either. At first this expression "x=x+1" violates all the mathematical training you've ever had, since you were always taught that an equation is equal on *both* sides of the equal sign. After all, that's what "equal" means, doesn't it? No, not in BASIC. You are now "incrementing" and "LET x=x+1" means, to the computer (whether you like it or not) to "make a new value of x equal to the present value plus one." To the left of the equal sign is the new value of the variable; to the right of the equal sign is the "old" value. If you try to get cute and type "LET x+1=x", the computer will not allow it to be ENTERed.

So, the computer has now added one to zero, and x is equal to one. The next line, 35, has several new things. Most noticeable is the strange looking symbol after x, which is simply "greater than" and is a shifted T key.

The keywords IF and THEN (keys U and shifted G) always go together and are used just as you would expect: IF a certain condition exists, THEN do something—which is why these are called "conditional commands" and cause "conditional branching" IF the condition is true.

There's also a "GO TO 60" at the end of the line, which sends the program to line 60—but only IF the specified condition is fulfilled.

Putting it all together, this line says, "IF x is greater than 50 THEN GO TO 60". So, following this instruction, the computer checks the

50), so the entire program line is ignored. It is very important that you realize that once the inspected condition "falls through"—that is, is not true—the computer goes to the next program line and does not do anything else on the IF line. Some computers have an ELSE statement that could appear later on the line and offer an alternative, but the T/S 2068 does not have this feature.

A Friendly POKE

Line 40 is enough to scare the daylights out of any BASIC programmer who has learned that POKE is a command to stir fear in the hearts of the staunchest! Why? Because, used improperly, a POKE to the wrong address will "crash" an otherwise perfect program and can reduce the results of hours of work to just the T/S sign-on copyright notices. (And reduce the programmer to tears!) So, don't POKE "unless POKEn to" (I can't resist a pun, even an awful one like this!) . . . that is, unless POKEing to a known "safe" address. This one is safe and so handy that you should write it down on a piece of paper and tape it to your machine. Really! This POKE23692,255 effectively disables the "scroll?" at the bottom of the screen when you try to PRINT on line 22 (the 23rd line)—provided you put the POKE ahead of a PRINT statement. Technically this POKE tells the computer to PRINT 255 more lines before halting with "scroll?" again.

Unconditional Branching

Line 50 says, in no uncertain terms, to GO TO 20. This, of course, means to GO TO line 20 of the program. No IFs, ANDs or buts, just GO TO 20. Now, actually, if the condition in line 35 had been met, the program would have jumped right over line 50 with a "conditional branching" to line 60. But x still only has a value of one.

Line 20 PRINTs the value of x (which is 1 right now), then line 30 increments by one, then line 35 falls through (since x is still not greater than 50), and the whole process repeats, counting and PRINTing one value at a time. This goes on until x finally reaches the value of 51, at which point line 35 is no longer ignored, and the program jumps to line 60.

The Count Down

Lines 60–100 look strangely like lines 20–50, because they are

rement") and use "less than zero" in line 70 as the conditional branch control. Notice that the PRINT statement is not in exactly the same relationship with the POKE (no GO TO in between), but that the POKE still comes before each PRINT—and that's all that matters. Bear in mind that there are many ways to accomplish the same goal in BASIC, and no two programmers will do it exactly the same way.

Anyhow, if you haven't "bailed out" during all these BASIC acrobatics, you should now have a good feel for how loops and branches work. Now let's look at another way to "loop" in BASIC, and add some other things as well.

FOR-NEXT LOOPS

LISTing 5-2 is a little longer, but type it into memory after using NEW (A key) to clear out the old program. This program illustrates subroutines, PRINT formatting, simple graphics, and the powerful FOR-NEXT statement. Incidentally, the keyboard legend for RETURN, on the Y key, is RETRN.

LISTing 5-2

```
 10 FOR x=0 TO 20
 30 PRINT AT x,x;x: GO SUB 200
 40 NEXT x
 50 GO SUB 300
 60 FOR x=20 TO 0 STEP -1
 80 PRINT AT x,x;x: GO SUB 200
 90 NEXT x
100 GO SUB 300
110 GO TO 10
200 PRINT AT 21,x;"■"
210 RETURN
300 PAUSE 30: CLS
310 RETURN
```

When you RUN this program, at one point in time you'll get the results shown in Fig. 5-1. First the screen clears, then the numbers staircase down the screen as the solid block extends from the left side of the screen to the right. Just as the 20 and the black bar meet, the program stops for about a half-second, the screen clears, and the process repeats—but in reverse! Weird, huh? But simple programs like this, with no apparent real-world value, can be great for learning the principles of programming. Real-world programs tend to

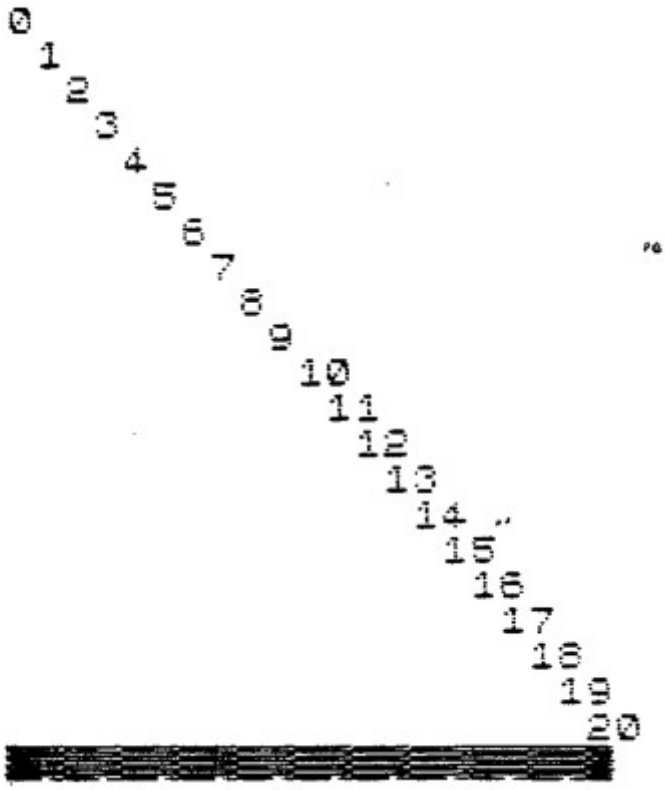


Fig. 5-1. RUN of FOR-NEXT loops.

get cluttered up with real-world needs. Part II of this book is composed of programs that do more purposeful things. At this point, the programming value is more important than the real-world value, so "dumb" programs are actually more useful than "smart" programs.

Line 10 starts a FOR-NEXT loop. Where's the NEXT? It's on line 40. A FOR statement must always have a NEXT somewhere further on in the program, with the same variable assigned, otherwise you'll either get an error message on the bottom of the screen, or the program will not loop back. In this case, line 10 says, "LET the variable x start with a value of 0 and increment to 20 in STEPs of one." Why STEPs of one? Because nothing else is specified, so "plus one" is the default value. If you wanted to increase by some other whole number you would specify, for example, "STEP 2." Don't use fractions or decimals; if you try them, they'll be rounded off to the closest whole number.

What happened to line 20? Where is it written that every number, or every tenth number, or any incremental line number must be used? Use what you like. I may have had something on line 20 that I later decided to remove. (Dead giveaway!)

Anyway, line 30 PRINTs the value of x on screen line x, at x spaces along the line. Check Chapter 3 if you don't remember how PRINT AT works. Since x is equal to zero this first time through the loop, a 0 is PRINTed at space 0 of line 0—the upper left corner of the screen.

SUBroutines

The colon on line 30 means there's another instruction, which says GO SUB 200. This means there's a "subroutine"—a mini-program module—at line 200. Generally, whenever the same programming ("code") can be used more than once in a program, it usually saves time and memory to use those lines in a subroutine. You'll notice that later on line 80 calls this same subroutine.

Line 200 PRINTs a simple graphic block (SYMBL SHIFTed 8 key with G-cursor) on line 21 (the last programmable screen line) at a position along the line determined by the current value of x. Since x is still zero, the block is PRINTED in the first character space on that line.

Line 210 simply instructs the computer to RETURN to where it came from. Whenever you have a GO SUB you must also have a RETURN or the program will just keep going down the LIST—which is not what you want, or you'd just use a GO TO.

Since there's no additional instruction on line 30, the program continues on to line 40 with NEXT x. That simply means "increment the value of x by the amount established by the preceding 'FOR x' statement—unless this makes x greater than the limiting value." In other words, x becomes 1, and the whole procedure repeats until x becomes 21.

Closing the Loop

Each time x increases by one, the PRINT AT statement in line 30 PRINTs the number down one line and over one space to the right, and the subroutine in line 200 prints another graphic block. When x finally becomes 21, it exceeds the "20" limiting value in line 10, so the program falls through to line 50.

Line 50 sends the program to a subroutine at line 300. Here the PAUSE 30 instruction stops everything while the computer counts 30 TV "frames" (about ½ second). The second instruction on that line, CLS, clears the screen. Line 310 RETURNs the program from whence it came—line 50. There's nothing further there, so it goes on to line 60.

Decrementing Loop

Lines 60–100 count down from 20 to 0 in a FOR-NEXT loop, using the same subroutines, and PRINTing everything in reverse order—

bottom of screen to top, right to left. Line 110 starts the whole thing all over again.

You may wonder about the use of the same variable, x, in different parts of the program. Any variable may be used any number of times in a program so long as it only carries one value at a time. Using x throughout this program shortened the program. If, for example, you decided to use y as the variable in the decrementing loop (lines 60–100), then the subroutine at line 200 would not work.

CALCULATED BRANCHING

LISTing 5-3 and Fig. 5-2 illustrate two very handy programming techniques—error-trapping and calculated branching. When typing this program into your T/S 2068 you can take advantage of a very handy feature not generally available in other computers—line duplication. Lines 100–900 are almost identical. Type and ENTER line

LISTing 5-3

```
   5 PRINT "To stop press any le
tter but a."
  10 INPUT "Number? (1-9)";a: PR
INT a
  20 IF a<1 OR a>9 THEN GO TO 10
00
  30 GO TO a*100
 100 PRINT "Your number is 1": G
O TO 10
 200 PRINT "Your number is 2": G
O TO 10
 300 PRINT "Your number is 3": G
O TO 10
 400 PRINT "Your number is 4": G
O TO 10
 500 PRINT "Your number is 5": G
O TO 10
 600 PRINT "Your number is 6": G
O TO 10
 700 PRINT "Your number is 7": G
O TO 10
 800 PRINT "Your number is 8": G
O TO 10
 900 PRINT "Your number is 9": G
O TO 10
1000 PRINT "Number range 1-9! If
[illegible]"
1010 GO TO 10
```

```
To stop press any letter but a.
5
Your number is 5
2
Your number is 2
7
Your number is 7
0
Number range 1-9! try again...
6
Your number is 6
12
Number range 1-9! try again...
6
Your number is 6
3
Your number is 3
55
Number range 1-9! try again...
```

Fig. 5-2. RUN of calculated branching.

100. Now, since this is the last line you ENTERed, it's the "current line." Hold down the CAPS SHIFT key and press the number 1 key (EDIT). This will bring line 100 to the bottom of the screen, with a flashing K-cursor right after the line number.

Now here's the trick. Hold down the CAPS SHIFT key, then press and hold down the 0 key (DELETE). Don't get confused by the attempts the computer makes to put the keyword DELETE on the screen. Once the auto-repeat function is invoked (by holding down both keys for a second or so) the cursor will finally "back up" and delete the line number. Now just type in 200 for the line number, move the cursor past "the number is 1" with the right-arrow key (shifted 8), DELETE the 1, type a 2, and then press ENTER. This all sounds like a real tough job, but you'll find it's really easy, and it allows you to duplicate lines or make minor changes for a new line much more easily than by retyping the line.

Outwitting the INPUT Statement

Once you get this program typed and ENTERed, RUN it. The screen clears and line 5 prints your "escape" message on the screen. The INPUT statement that follows will not respond to a BREAK, and this will remind you how to quit the program. Since this INPUT is expecting a number, you can create an intentional error, which BREAKs the program, by ENTERing any letter—except the letter it expects. Once a value has been assigned to that variable, it

will hold that value until changed. Pressing any other letter gives you a "variable not found" error.

Is There an Error in the House?

Line 10 asks for a number from 1 to 9 and then PRINTs it on the screen. Line 20 examines this INPUT, the variable a, to see if it is the specified range. IF a is less than 1 OR greater than 9 THEN the program GOes TO line 1000. Line 1000 contains the error message (with some reverse video to make it stand out), and then line 1010 returns the program back to the INPUT statement for another try. This process is called "error trapping" and becomes very important in programs that can "crash" with INPUTs that are beyond program or computer limits.

The T/S 2068 also has a built-in ERRor-trapping statement that allows you to keep the program RUNning, even when an ERRor is encountered. The ON ERR GO TO, ON ERR CONT, and ON ERR RESET statements are explained in Appendix A of the T/S 2058 User Manual.

Telling Your Program Where To GO!

We've already covered conditional and unconditional branching, but here line 30 has what I call "calculated branching." Many computers have an ON GO TO statement; the T/S 2068 does not. This technique is the substitute. Just set up line numbers in the program that are multiples of the expected numerical INPUT and use GO TO with the variable multiplied by that factor. For example, if the number 3 is INPUT in response to line 10, then line 30 multiplies this by 100, with a result of 300, and the program branches to line 300. Simple, but very useful.

The GO TO 10 at the end of lines 100–900 is important, or the program will go right down from the selected line and PRINT all the statements that follow it.

ESP TESTER

Here's a little program that uses comparisons and conditions to see if your Extra-Sensory Perception (ESP) is sufficient to determine the next two random numbers the computer will select. Type and ENTER LISTing 5-4, and then RUN it. Fig. 5-3 shows a typical screen using the COPY command for the printer.

LISTing 5-4

```
  10 PRINT "ENTER 9,9 to quit...
.": RANDOMIZE
  20 INPUT "ENTER 2 digits (1-9)
...";a,b: PRINT a,b
  30 IF a*b=81 THEN STOP
  40 LET c=INT (RND*9)+1: LET d=
INT (RND*9)+1
  50 IF a=c AND b=d THEN PRINT a
;" and ";d;" are correct!": GO T
O 20
  60 PRINT "Sorry! Try again..."
  70 PRINT "The computer chose "
;c;" and ";d
  80 GO TO 20
```

```
ENTER 9,9 to quit....
2                 3
Sorry! Try again...
The computer chose 7 and 2
5                 6
Sorry! Try again...
The computer chose 9 and 6
1                 7
Sorry! Try again...
The computer chose 9 and 6
3                 5
Sorry! Try again...
The computer chose 6 and 3
4                 7
Sorry! Try again...
The computer chose 7 and 9
3                 8
Sorry! Try again...
The computer chose 2 and 7
```

Fig. 5-3. RUN of ESP tester.

Pseudo-Random Numbers

Line 10 shows another way to escape the INPUT statement by specifying certain INPUT. Notice the RANDOMIZE (marked RAND on the T key) at the end of line 10. Without that, the computer would select the same "random" numbers in line 40 each time the computer was turned on. This is because the "random numbers" are really from a table built into the computer that provides a pseudo-random sequence. RANDOMIZE "moves up" this table by a number which is based on the time elapsed since the computer was turned on. An internal register in the computer is constantly counting TV frames

so the number which determines where the RND statement starts in the pseudo-random table is always changing. Tricky!

Multiple Inputs

Line 20 asks for two separate INPUTs, with one assigned to variable a and the other to variable b. When you INPUT each number, press ENTER. You'll need to use two numbers and press ENTER twice before line 20 will PRINT the values.

Line 30 checks to see if you want to end the program by using a nine for each INPUT. IF a multiplied by b equals 81, the program STOPs. Otherwise, line 40 selects RaNDom values for new variables c and d, with the values from 1 to 9. The INTeger statement assures that the number will be a whole number, rounded down from the RaNDom value. The "+1" assures that each number will have a value of at least one.

Now line 50 compares the various values. IF a is equal to c, and b is equal to d (both conditions must be true), you have ESP! Usually, you'll fail this test and lines 60–80 tell you what numbers the computer chose and give you another try. Try this 100 times and you will probably not pick the correct numbers even once.

Other Conditions

Look at the Q, W, and E keys, in the black "shifted" zone, and you'll see some more conditional symbols. The one on the Q key means "less than or equal to" and the one on the E key means "greater than or equal to." But what about that strange symbol on the W key. It means, simply, "not equal to." Incidentally, like all keywords on the T/S 2068, they must be entered with the single-key, and not as combinations of other keys.

DIGITAL TIMER

This program provides a way for you to replace your portable $25 digital alarm/chronograph/wristwatch with your $200+ plugged-into-the-wall computer system! Not only that, but this program is nowhere near as accurate as your wristwatch . . . but your watch won't teach you about nested FOR-NEXT loops!

Type and ENTER LISTing 5-5. Look at all those FOR-NEXT loops! There are four of them, "nested" together, plus an incrementing var-

LISTing 5-5

```
 100 REM * Digital Timer Program
 *
 110 REM * (C) Copyright Fred Bl
echman 1983 *
 200 PRINT AT 8,3;"D I G I T A L
  T I M E R"
 210 PRINT AT 11,0;"Hours  Minut
es  Seconds  Tenths"
 220 PRINT AT 18,4;"Press A-key
to stop."
 300 LET h=0
 310 FOR m=0 TO 59
 320 FOR s=0 TO 59
 330 FOR t=0 TO 9
 400 LET a$=INKEY$
 410 IF a$="a" THEN STOP
 420 FOR d=1 TO 7: NEXT d
 430 PRINT AT 12,2;h;"        ";m
;"        ";s;"        .";t;" "
 500 NEXT t
 510 NEXT s
 520 NEXT m
 600 LET h=h+1
 610 GO TO 310
```

iable. To add to the fun, there's an INKEY$ (called "INKEY-string") for a STOP command. Yet the whole program is very simple.

Initialization

"Initialization" is another computer "buzzword"; it simply means "getting started." This program gets started by identifying itself (lines 100–200) and then PRINTing headings (line 210) and the STOP instruction (line 220). Lines 300–330 establish initial values for variables h ("hours," using LET) and m, s, and t (for "minutes," "seconds," and "tenths") using FOR-NEXT loops.

Line 400 uses INKEY$, a keyboard scanning instruction. It says, in effect, "LET a$ equal any key pressed at this instant." In other words, it scans the keyboard to see if a key is pressed, and if it is, it gives a$ that designation. Hmmmmm. What's a$? We'll get to that in Chapter 7. For now, trust me! Line 410 looks to see if you pressed the A key (which looks the same as lowercase in this function). If so, the program STOPs, and you read elapsed time.

Line 420 is a "delay loop," with the computer counting to itself seven times. Strange behavior—but it takes an increment of time, and creates a delay to provide accuracy in counting tenths of a

second. This line becomes our "fine-tuning" speed control, as you'll see.

Let Me Count the Ways

Line 430 formats and PRINTs the starting values of zero for h, m, s, and t. Now comes the fun. Line 500 increments the value of t, working with the "inside" loop in line 330, until t exceeds 9. Notice that each time through the loop, lines 400–430 are active. The screen is "updated" so fast and so often that the only thing that looks like it's changing is the "Tenths" digit, but the entire line is actually being rewritten ten times a second.

When t equals 10, the program falls through to line 510 which counts seconds, together with line 320, until s equals 60. At that point lines 520 and 310 update minutes. Eventually, when m is 60, hours are increased by one, and line 610 goes back to counting minutes, seconds and tenths.

You are bound to be confused, but if you study this and follow the logic, you'll suddenly understand it and have control of one of the most powerful BASIC tools—FOR-NEXT loops in combination. Fig. 5-4 shows a typical RUN.

```
D I G I T A L  T I M E R

Hours   Minutes   Seconds   Tenths
  0         1         32       .7


Press A-key to stop..
```

Fig. 5-4. RUN of digital timer.

Accuracy

If you've figured this out, you'll see that the delay loop in line 420 controls how fast the "Tenths" count. You can change the 7 to an 8 to slow down the counting, or to a 6 to speed up your "timer." Unfortunately, however, the local ambient temperature and the heat generated within the computer itself both affect the internal "clock"

speed, so its long-term accuracy is not great. Another example of "computer overkill!"

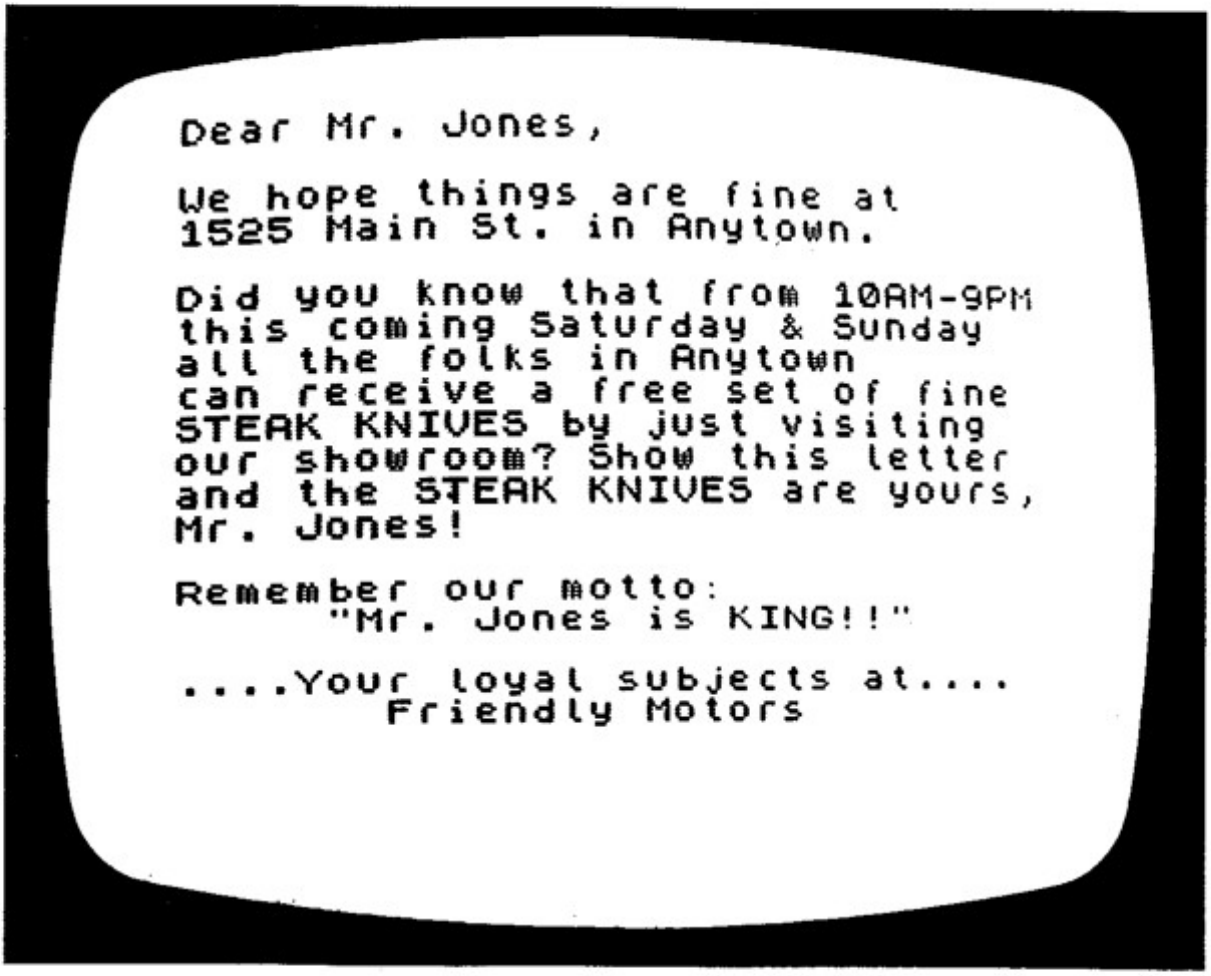
QUESTIONS

1. How do you stop the screen from interrupting a screen display with the "scroll?" message?
2. Write a FOR statement to have variable x start at 50 and count to 100 by 5. Is this incrementing or decrementing?
3. What is the symbol for "greater than"? "Lesser than"? How about "not equal to"?
4. What keyword must be used with FOR? With IF? With GO SUB?
5. When are subroutines generally used?
6. What value will be PRINTed by this command?

```
FOR x=5 TO 25 STEP 5: NEXT x: PRINT x
```

7. How do you escape from a numerical INPUT statement?
8. Do nested FOR-NEXT loops work from the inside out, or from the outside in?
9. Why is the Digital Timer Program not accurate?

6
Strings and Many Other Things



RUNning the form letter program.

INTRODUCTION

Now that you've been exposed to the essential functions and commands, with screen formatting, loops, subroutines and branching under your belt, how much more can there be? The answer: lots more! BASIC programming offers the creative programmer an almost endless variety of means to accomplish a given task—or, putting it another way, "many ways to do the same thing." We have not yet covered some of the most powerful commands and techniques. In this chapter we'll start bridging the gap between the basics of BASIC and real-world applications. Strings, and various string-handling techniques, could almost themselves be the subject of a

book of their own. We'll cover them in some depth, and also discuss mathematical functions, and a few commands generally reserved for the mystical world of "machine language programming." See the next book in this series, the *Timex 2068 Intermediate/Advanced Guide,* for more details on some of these subjects.

THINGS ABOUT STRINGS

Firstly, just what is a "string" in BASIC programming? It's any combination of characters that includes more than just numbers. For example, "This is a string" is a string. So is "So is this!", and so is "34%", since it contains a character—the percent sign—that is not a number. Notice that a string is always enclosed in quotation marks, both before and after the string characters. Actually, even a number is considered to be a string if it is placed between quotation marks. The T/S 2068 is very strict about strings being enclosed in quotation marks wherever they are used, unlike some computers that permit closing quotes to be dropped, and sometimes (in DATA statements—next chapter) don't require quotes at all. The T/S wants quotes around strings, and don't you forget it!

String Variables

Just as LET a=1 establishes a value for the numerical variable a, you can create string variables by adding the $ sign to a single letter. This means a$ (called "a-string," but sometimes referred to as "a-dollar sign") to z$ can be assigned strings. (What? Only 26 string variables? No. In the next chapter you'll see a way, using arrays, to get all the additional string variables you'll ever want!)

For example, LET a$="Joe Doaks" will assign this string to a$. Anytime from then on that you wish to refer to this string, you simply use a$. Don't forget the quotes before and after the string.

Rather than go into more explanation, LISTing 6-1 shows you a simple "form-letter" that plugs strings into text. Once you've assigned the strings in lines 30–60, they can be used anywhere else in the program. Fig. 6-1 shows the output to the printer. Notice how a$–d$ are used to address the letter, then used again at the end to print a mailing label? See how a$, b$, and c$ are used within the text? To do the letter for another "prospect" just change the strings in lines 30–60.

LISTing 6-1

```
  10 REM © Fred Blechman 1983
  20 REM * Form Letter" *
  30 LET a$="Jones"
  40 LET b$="1525 Main St."
  50 LET c$="Anytown"
  60 LET d$="CA  93546"
  70 PRINT
 100 PRINT "Dear Mr. ";a$;",": P
RINT
 110 PRINT "We hope things are f
ine at"
 120 PRINT b$;" in ";c$;".": PRI
NT
 130 PRINT "Did you know that fr
om 10AM-9PM"
 135 PRINT "this coming Saturday
 & Sunday"
 140 PRINT "all the folks in ";c
$
 150 PRINT "can receive a free s
et of fine"
 160 PRINT "STEAK KNIVES by just
 visiting"
 170 PRINT "our showroom? Show t
his letter"
 180 PRINT "and the STEAK KNIVES
 are yours,"
 190 PRINT "Mr. ";a$;"!": PRINT
 200 PRINT "Remember our motto:"
 210 PRINT "     ""Mr. ";a$;" is
 KING!!""": PRINT
 220 PRINT "....Your loyal subje
cts at...."
 230 PRINT "          Friendly Moto
rs"
 300 LPRINT a$: LPRINT b$: LPRIN
T c$: LPRINT d$: LPRINT
 310 COPY
 320 LPRINT
 330 LPRINT a$: LPRINT b$: LPRIN
T c$: LPRINT d$: LPRINT
```

Line 310 copies the entire letter from the screen, and line 330 prints the mailing label. Actually, by using READ/DATA statements (which are discussed in the next chapter) you could program a whole mailing list into memory and have the printer grind it out without your intervention.

Two things in this program require explanation. In line 10, you see the copyright symbol. Where did that come from? It's not on the keyboard! It really is—hidden under the keyword RESET on the P

```
Jones
1525 Main St.
Anytown
CA  93546


Dear Mr. Jones,

We hope things are fine at
1525 Main St. in Anytown.

Did you know that from 10AM-9PM
this coming Saturday & Sunday
all the folks in Anytown
can receive a free set of fine
STEAK KNIVES by just visiting
our showroom? Show this letter
and the STEAK KNIVES are yours,
Mr. Jones!

Remember our motto:
      "Mr. Jones is KING!!"

....Your loyal subjects at....
        Friendly Motors



Jones
1525 Main St.
Anytown
CA  93546
```

Fig. 6-1. Form letter.

key. To access it, you must first have an L-cursor (not a K-cursor or you'll end up RESETting the computer peripherals). Now go to an E-cursor by pressing the CAPS SHIFT and SYMBL SHIFT keys, then hold down the SYMBL SHIFT key and press the P key. The copyright symbol will appear on your screen. Similarly, under the words ON ERR and SOUND (beneath the F and G keys) you'll find curly left and right braces, obtained in the same manner—but only when the L-cursor precedes the E-cursor.

The other little "trick" in this program is using quotes within quotes. Normally, a quotation mark is used to start and end a string. But sometimes you want quotation marks to enclose screen or printed text. The T/S 2068 allows you to do this by being persistent. You must use two sets of quotation marks before and after the expression you want quoted. Line 210 shows this. Neat, huh? Another plus for Timex BASIC!

String Operations

This is an area of BASIC where T/S BASIC is quite different from other BASICs, but is truly powerful. Most other BASICs either have very limited string handling or require a bunch of different commands to define parts of strings. With the T/S 2068 you use parentheses and the keyword TO to select portions of a string. For example:

```
LET a$="This is a string."
```

There are 17 characters in this string, including the spaces and period, since all characters count. To extract the word "This" from the string, we could say:

```
LET b$=a$(1 TO 4)      or      LET b$=a$(TO 4)
```

If an initial number is not specified within the parentheses, then 1 is assumed. That gets the leading characters. But how about specific characters within the string? To extract " is " (with space before and after):

```
LET c$=a$(5 TO 8)
```

To extract the single character "a":

```
LET d$=a$(9)
```

And, finally, the final word "string." (with space before and period at the end):

```
LET e$=a$(10 TO 17)      or      LET e$=a$(10 TO )
```

Notice that here the last number can be omitted and the end of the string is used.

String Concatenation

There's another computer buzzword, "concatenation," which merely means "add together" or "combine." This is done by adding the strings together with plus signs, like this:

```
PRINT b$+c$+d$+e$
```

That's getting "This is a string." the hard way, but it illustrates string slicing. LISTing 6-2 summarizes the above, and Fig. 6-2 shows you the screen output.

LISTing 6-2

```
 10 LET a$="This is a string."
 20 LET b$=a$(1 TO 4)
 30 PRINT b$
 40 LET b$=a$( TO 4)
 50 PRINT b$
 60 LET c$=a$(5 TO 8)
 70 PRINT c$
 80 LET d$=a$(9)
 90 PRINT d$
100 LET e$=a$(10 TO 17)
110 PRINT e$
120 LET e$=a$(10 TO )
130 PRINT e$
140 PRINT b$+c$+d$+e$
```

```
This
This
 is
a
 string.
 string.
This is a string.
```

Fig. 6-2. String concatenation.

Why are "This" and "string." printed twice? Because these strings are assigned two different ways (lines 20 and 40 for the first one, and lines 100 and 120 for the second one) and each is PRINTed. But notice that only the *last* values of b$ and e$ are PRINTed in line 140. The computer only "remembers" the last value assigned to any variable.

LEFT$, MID$ and RIGHT$, commonly used in other BASIC dialects, are handled on the T/S 2068 this way:

```
LEFT$(A$,N) = a$( TO n)
MID$(A$,M,N) = a$(m TO m+n-1)
RIGHT$(A$,N) = a$(LEN a$-n+1 TO )
```

String handling is an art in itself, and the parentheses can contain calculations of character positions, too. To aid you in string operations, there are some additional statements—LEN, STR$, VAL, VAL$, CODE, and CHR$.

String LENgth

If you have a$ = "This is a string." in memory, and you want the LENgth of the string, you could PRINT LEN a$ and you'd see the value 17 on your screen. LEN counts all characters, including blank spaces and punctuation.

Converting Numbers to Strings and Back Again

Sometimes it's necessary to treat a number as a string. The STR$ command does this. For example:

```
LET x$ = STR$ 1.234
```

This makes x$ = "1.234." However, you can't calculate with strings, so VAL x$, in this case, would equal 1.234—a number. In other words, STR$ makes a number a string, and VAL makes a string a number. As is usually the case, you can use these commands with parentheses containing numerical calculations where appropriate. For example, type NEW, then ENTER, then the following:

```
10 LET a$ = STR$ (5*10)
20 LET b$ = STR$ (50/5)
30 PRINT (VAL a$*VAL b$ + 1.5*3)
```

When you RUN this, a$ equals "50," b$ equals "10," and line 30 calculates 50 times 10 for a VALue of 500—then adds the result of 1.5 times 3, which is 4.5, for a total of 500 plus 4.5, or 504.5. As useless as this may seem at the moment, there are situations where it can be handy.

There is also a VAL$ command which processes a string and ends up with a string. It, in effect, removes a string within a string—a rare use, not worth discussing here.

ASCII CODE AND CHR$

The keyword CODE has two uses. We mentioned in Chapter 4 that it can be used with the SAVE command. With strings, however, CODE gives the ASCII CODE of the first character in a string. ASCII Code? What's that? It stands for American Standard Codes for Information Exchange, which define the characters and punctuation used by most popular computers. (The T/S 1000 and 1500 do not use

ASCII codes.) The Character Set of the 2068 is shown in Appendix B of the T/S 2068 User Manual. It is conventional ASCII for the characters and punctuation but deviates (as most computers do) for graphic, keyword, and control codes.

So, for example, if a$="TEST," and you ask the computer to PRINT CODE a$, the display shows an 84—which is the ASCII CODE for a capital T. If a$ were "test," then CODE a$ would return 116 for a lowercase t. This command, CODE, is sometimes useful in alphabetical search and sort routines.

The keyword CHR$ is your way of calling a particular character from the character set. Type PRINT CHR$ 84, then press ENTER, and a capital T will appear on your screen. You must be careful, however, since most of the numbers below CHR$ 32 are either not used, or are "control codes" that can cause errors. In Chapter 3 we had a program to PRINT the CHR$s for values from 32 to 255. You might want to go back and look at that again. Also, as mentioned before, these CHR$s are shown in the Appendix B Character Set of your T/S 2000 User Manual. Notice that the graphic blocks on keys 1–8 (and their reverse) are CHR$ numbers 128–143.

How do you use this information? Some sophisticated programming makes use of CHR$ for doing "string packing," a method of achieving high speed graphics. You probably won't use CHR$ much, but if you ever need to specify a particular character outside of keyboard use, this is the way to do it.

String Comparisons

Just as numbers can be compared, so can strings. Actually, when comparing strings, the ASCII values of the characters are compared, so it's really a number comparison after all. By now you should know, if you haven't already realized it, that computers do everything by numbers—and, in fact, really only use "0" and "1." Amazing, isn't it?

Type NEW, ENTER and then this program:

```
10 INPUT"Type and ENTER the password (S-h-h-h)";a$
20 IF a$="password" THEN GO TO 40
30 PRINT"IMPOSTER! THROW HIM IN IRONS!": STOP
40 PRINT"You may pass, Brother.....": STOP
```

Line 10 lets you try to enter the password. Line 20 compares what you ENTERed with "password," using the CHR$ value of each char-

acter. Unless all match, the program falls through to line 30 and STOPs. If you have used exactly the right password, letter for letter, line 20 sends the program to line 40 for a successful passage. Note, when you try this, that if you use capital letters in place of lower case in your INPUT, you will not "compare." You may, of course, change the "password" in line 20 to anything you like, including graphic codes.

String comparisons can also be used with the "greater than," "less than," and the other conditional expressions. Again, the CHR$ values are added together for evaluation. Thus, the word (excuse the expression) "Apple" (a value of 498) is "less than" the word "apple" (a value of 530) since the lowercase a has a higher value than the capital A! This is all of extreme importance in sort and search functions.

Because it's important that you understand this concept, it would be a good idea for you to ENTER LISTing 6-3 into your T/S 2068 and RUN it. Try different five-character words in lines 10 and 110 in place of "Apple" and "apple." You can use longer words if you change the 5 in line 30 to the length of the words you use (although the words in both lines 10 and 110 should be the same length for this simple program.)

LISTing 6-3

```
 10 LET a$="Apple": LET f=0
 20 LET v=0
 30 FOR x=1 TO 5
 40 LET c=CODE a$(x)
 50 LET v=v+c
 60 NEXT x
 70 PRINT a$;" has a value of "
;v
 80 IF f=1 THEN STOP
100 LET f=1
110 LET a$="apple"
120 GO TO 20
```

You should be able to follow this program but the variable "f" in lines 10, 80 and 100 may puzzle you. This is a "flag" to STOP the program, and it allows some of the program (lines 20–80) to be used twice. The first time through, f = 0 (from line 10), so line 80 is ignored. But line 100 changes the value of f to 1, and after the second time through lines 20–70, line 80 sees the value of f equal to 1 and STOPs the program.

MATHEMATICAL FUNCTIONS

Calculators and computers love to "number crunch," and the T/S 2068 has plenty of commands to do just that. If you're "into math," you'll probably find the following too elementary, so skip it. If, however, you're still struggling with the square root program in Chapter 2, then this section will help you get a feeling for the incredible power Timex has put in your hands (just like their ads say!).

Simple Stuff

Operations like addition, subtraction, multiplication and division are straightforward—except that you may have a little trouble getting used to the asterisk (*) as a multiplication sign instead of x. The reason for this, of course, is that the letter x, as are all letters, is used as a variable. The slash sign (/) is used for division instead of the horizontal bar or colon used in "longhand" math. The parentheses are used as brackets; multiple levels are allowed, but always be sure you have exactly the same number of left-parenthesis symbols as right-parenthesis symbols in a program instruction.

The computer establishes priorities when doing calculations, starting with the innermost parentheses and working outward, always doing multiplication and division before addition and subtraction, and working left to right. If some equations are confusing to you, they may also be confusing to the computer, so don't be afraid to use parentheses to clarify. For example, what does T equal in this equation?

```
LET T=15+3*5
```

The computer will evaluate T as equal to 30. It first multiplies 3 by 5 (=15) and then adds 15. Remember, even though it works left to right, it does multiplication before division. On the other hand, you might add 15 and 3 to get 18 and then multiply that by 5 for a result of 90. Which is right? It depends on what you really meant to do. However, the fact is, the equation is technically incorrect since it is open to interpretation. Rewrite it this way and there can be only one answer:

```
LET T=(15+3)*5
```

Now the T/S 2068, seeing parentheses, does the calculation inside first, coming up with 18, then multiplies that by 5 to get the result of 90—with no question about it.

The Trig Functions

Trigonometric functions are fully supported by the T/S 2068. The explanations of SIN (sine), COS (cosine), TAN (tangent), ASN (arc sine), ACS (arc cosine), and ATN (arc tangent) can be found in any trigonometry book. It's sufficient to say here that examples of the use of some of them will be found in Chapter 13, "Bio-Graphs & You," and Chapter 17, "Long-Distance Navigator." When using these commands, you must realize that the computer expresses them in "radians" rather than the "degrees" you are probably used to. The conversion is simple:

```
degrees=radians*(180/PI)
     (1 radian=57.29578 degrees)
radians=degrees*(PI/180)
     (1 degree=.017453293 radians)
PI=3.1415926535+
```

Fortunately, PI is a value held in the computer by pressing the M key when in the E-cursor mode, and it's actually more accurate to use 180/PI compared to 57.29578, since the computer internally calculates to a greater number of places than it displays.

MISCELLANEOUS MATHEMATICS

The SGN (sign) function (above the F key) placed before a number (such as SGN −2.345) returns a 1 if the number is above zero, a 0 if the number is zero, and a −1 if the number is below zero. The ABSolute function (above the G key) strips a negative sign from a number and makes it positive. For example, ABS −2.345 will equal 2.345.

The Handy INTeger

The INTeger function (above the R key) is really quite useful. It rounds any number down to the next whole number. For example, INT 3.556 will equal 3, even though normal "rounding" would move it up to 4. The INTeger of 3.999 is still 3. INTeger always rounds down, even in the negative numbers. INT −3.123 becomes a −4.

A very useful subroutine for getting two-place decimals is to take the number, multiply it by 100, add .5, divide by 100, and then take the INTeger. For example:

```
10 LET x=RND*100
20 PRINT x,
30 PRINT INT (x*100+.5)/100
40 GO TO 10
```

Line 10 generates a random number from 0 to 99 and line 20 PRINTs the number. Notice the comma at the end of line 20, which means the next PRINTing will be halfway across the screen. Line 30 multiplies the number by 100, adds .5, divides the total by 100, then takes the INTeger and PRINTs it. Line 40 GOes back TO do it again. A typical screen of results (before stopped with "scroll?") is shown in Fig. 6-3. Notice how the numbers are "rounded" in the conventional manner, to two decimal places (although trailing zeros are dropped.)

```
16.993713       16.99
74.623108       74.62
96.762085       96.76
57.159424       57.16
87.005615       87.01
25.434875       25.43
7.699585        7.7
77.574158       77.57
18.086243       18.09
56.561279       56.56
42.144775       42.14
60.923767       60.92
69.326782       69.33
99.543762       99.54
65.782166       65.78
33.700562       33.7
27.618882       27.62
71.348572       71.35
51.174927       51.17
38.174438       38.17
63.153076       63.15
36.521912       36.52
```

Fig. 6-3. Two-place decimals.

Keep this simple INTeger statement in mind the next time you're working with financial programs and want to round to cents. Incidentally, changing both 100's in line 30 to 1000 gives you *three* decimal places. Isn't that neat?

Getting to the Root of Things

The SQR function (above the H key) gives you the square root of a number. If you ever struggled through doing this the longhand way, you'll really appreciate SQR. See Chapter 2 for a program using

SQR. If you're interested in getting other roots, like the cube root, 4th root, 5th root, or whatever, it's really easy on a computer. You simply use the "raise to a power" function (the up-arrow on the H key), and use a fractional exponent. For example, to find the 8th root of 256, type and ENTER:

```
PRINT 256 ↑ (1/8)
```

Be sure to put the fractional exponent in parentheses or the computer, in this case, will simply divide 256 by 8 and give you an answer of 32. The answer, of course, is that the 8th root of 256 is 2. Why? Work it backwards. Two to the eighth power is 256. (When playing with math, always test out functions for which you know the answers.)

Natural Logs

The LN and EXP functions (above the Z and X keys) are related to logarithmic and exponential functions, somewhat beyond the intent of this book. EXP (1) has a value of 2.7182818284+. This is used in calculations using LN, natural logarithms to the base e. If you will be using these, you already know how.

Defining Functions

It's handy at times to take a formula and DEFine it as a special variable. You do this with DEF FN (below the 1 key), then press the FN (below the 2 key) to use it. Although whole books have been written mostly about the imaginative uses of DEF FN, we'll only give a couple of simple examples here.

Say you wanted to DEFine the following formula:

```
x=(10*a+5)/10
```

You then take any variable name, say f, and DEF FN f(a)=(10*a+5)/10. Now, anytime you want a value for x you use FN f(a). In this example, if a=30, then FN f(a)=30.5. This sample program will give you some practice. First type NEW and ENTER:

```
10 DEF FN f(a)=(10*a+5)/10
20 INPUT a
30 PRINT FN f(a)
40 GO TO 20
```

Here's another example, for PLOTting a SINe wave on the display. This draws the curve across the entire height and width of the display.

```
10 DEF FN f(a) = SIN (a*PI/180)
20 FOR a = 0 TO 360
30 PLOT a*(255/360),87*FN f(a) + 87
40 NEXT a
```

There's a lot more to DEF FN and FN than meets the eye, and they can be used in powerful ways by experienced programmers to make programs "faster and better."

PEEKING AND POKEING AROUND

Two commands found frequently in BASIC programs—but seldom explained—are PEEK and POKE. The commands are directly related to the internal memory of the computer, and while PEEKing around at various memory "addresses" can't hurt, you can easily crash a program (but not harm the computer) with a POKE in the wrong spot.

PEEK (above the A key) will return the decimal number in any decimal address, and is of particular value to machine language programmers to identify the location of various "pointers" and parts of a program. Jeff Mazur will go into this in detail in the *Timex 2068 Intermediate/Advanced Guide.*

POKE and USR

POKE, also mostly of value to advanced programmers, has a number of uses for the BASIC programmer. For one thing, as you'll see in Chapter 9, "Grappling With Graphics," the POKE and USR statements are used to create special graphic characters. POKE can also be used with a BASIC program to insert machine language subroutines into the computer's memory at specific locations.

When this is done, the USR command sends the computer to the address of that routine. USR 32400, for example, would send the program to decimal address 32400, where you might have, perhaps, a renumber program in machine language.

There are, however, a few POKEs you can use safely and easily. Appendix D of the T/S 2068 User Manual consists of several pages

of "system variable" addresses, many of which can be changed to suit your wishes. Here are two:

POKE 23609,100 . . . Beeps when key pressed
POKE 23692,255 . . . Use before every PRINT to defeat "scroll?"

THE INS AND OUTS

Just as there are 65,536 (256 × 256) possible memory addresses with an 8-bit microprocessor like the Z-80 used in the T/S 2068, there are also Input/Output (called "I/O") "ports," numbered form 0 to 255, with each having numbers from 0 to 255 following. For example, OUT 255,0 addresses a particular port (in this case, the normal video mode). Although some OUT commands can be used to control internal computer activities (like the colors, display mode, and sound) most of the OUT commands are used to control external peripherals, such as printer, interfaces, additional memory and other devices.

The IN command, also used with a number from 0 to 255, provides status information to the computer. As implied by their titles, IN is an INput, OUT is an OUTput. These are also sometimes likened to reading (IN) and writing (OUT).

QUESTIONS

1. What's the difference between a "string" and a number?
2. How do you access the copyright symbol? The left curly brace? The right curly brace?
3. How do you put quotation marks within a PRINT statement?
4. If a$ = "Question," what will PRINT a$(3 TO 5) display?
5. What is the ASCII CODE for the capital letter X? How about a question mark?
6. Which word would come first in a sort by ascending value, "Test" or "test"?
7. What sign is used by the computer to indicate multiplication? Raising to a power?
8. What are the INTegers of 234.499, 234.501, −12.51, −12.49?
9. Can you harm anything with a PEEK? How about POKE?

7
Program Storage and Manipulation

RUNning the club roster program.

INTRODUCTION

One of the big advantages of a computer over a file card system is its ability to store and manipulate DATA. What is DATA? Information, such as names, addresses, numbers—just about anything. Sometimes this information is entered into the computer from the keyboard with the INPUT statement. By using the LET statements within a program, you can assign information to variables held in memory, and you can LOAD information into the computer from a cassette tape recorder. In this chapter we'll discuss another way of

putting DATA into the T/S 2068, by making this information part of the program itself. We'll also get into a whole new concept of assigning and manipulating variables, the mysterious "array."

READ AND DATA STATEMENTS

If you'll take the time to understand the READ, DATA, and RESTORE statements available on the T/S 2068 (and most microcomputers, but not on the T/S 1000 or 1500) you will have a powerful means of accomplishing tasks frequently delegated to external disk files. Presently, disk drives are not available for the T/S 2068. Although cassette files can be generated, they are slow and clumsy to use. You'll be able to search DATA for just the information you want much more quickly than card files, and you can PRINT the results.

The DATA statement, as its name implies, stores information. It does this in a very simple *sequential* format, with line numbers, *anywhere* within a program. A DATA statement starts right after the line number with the keyword DATA (above the D key). While you can have any number of items in a DATA statement, it's a good idea to keep DATA lines no longer than two screen lines so they may be edited easily, if necessary.

The information held in DATA statements is either numbers or strings. Numbers are typed in the normal way, but strings must be enclosed in parentheses. An example will provide the best explanation of how DATA statements work. With no program in memory, type and ENTER this (READ is above the A key):

```
10 READ a
20 PRINT a
30 IF a=9999 THEN STOP
40 GO TO 10
100 DATA 1.5,13,4*3.2,13.45,17,2.55,99.9,18.7,-23.45,0
110 DATA 34.56,-25.98,3.5*a,17.44,52.8,13.45
200 DATA 9999
```

Although the DATA statements are shown at the end of this short program, they can be anywhere: beginning, middle, end, or even split up. However, when you RUN the program (or use CLEAR), the "DATA pointer" will start at the first DATA entry, wherever it is.

In this program, the screen PRINTS each number one at a time, in the exact sequence shown in the DATA statements. Line 10

READs the first DATA item, 1.5, and assigns this to variable a. Line 20 PRINTs the value of a, which has just been read as 1.5. Since line 30 is not satisfied (a is not equal to 9999) the program falls through to line 40, which returns to line 10 for another READ. It seems that the program can never get beyond line 40 to line 100, yet it is actually nibbling away at the DATA statements one item at a time. The second time through, a is assigned the number 13, which gets PRINTed, and then back for another READ. This keeps going on until line 100 is "used," then line 110, then finally a is assigned 9999 in line 200, and the program stops.

Several things are worth noting about this simple program. For one thing, every DATA item is separated by a comma—absolutely necessary! The comma is known as a "delimiter" which tells the computer where to stop nibbling at the DATA. Notice that there is no comma after the word DATA. Notice also that only numbers or calculations (positive or negative) are used in DATA when the READ statement asks for a numerical variable. The only reason that the 3.5*a item in line 110 works is that a has a numerical value from the previous READ.

The 9999 in line 200 is the STOP signal. This is usually put on a line number beyond the regular DATA statements to allow more DATA lines in between if necessary. It should be the last DATA statement and does not need to be a separate line number, but could be at the end of the last DATA line.

What about strings? If the DATA is in strings, then the READ statement must use a string variable. Change line 10 in the above program to a$—and it "crashes" immediately when you try to run it. The error statement at the bottom of the screen says, "C Nonsense in BASIC, 10:1". Why? Because the numbers after the first DATA are not in quotation marks. The T/S 2068 demands that strings be enclosed in quotes.

To salvage this program, you must put quotation marks around every number, even the 9999 in line 30. This is too much trouble. Just remember that READing a string takes quotes around each DATA item.

Finding the Month

To give you a better idea of how READ and DATA can be used to advantage, type NEW, press ENTER, and then type and ENTER the program in LISTing 7-1. When you RUN this program it will ask you

LISTing 7-1

```
  10 REM * © Fred Blechman 1983
  20 PRINT INK 2;"Date:MM/DD/YY"
,,
  30 INPUT m,d,y
  40 READ a,m$
  50 IF a=m THEN GO TO 70
  60 GO TO 40
  70 PRINT "The date is:"
  80 PRINT INK 4;m$;" ";d;", 19"
;y
  90 RESTORE : PRINT : GO TO 20
 100 DATA 1,"January",2,"Februar
y",3,"March",4,"April",5,"May",6
,"June"
 110 DATA 7,"July",8,"August",9,
"September",10,"October",11,"Nov
ember",12,"December"
```

(in red INK) to type the numbers for a month, day, and year. The format, MM/DD/YY, is specified—two digits for each entry. Actually, you can get by with single digits for month and day—since the computer drops leading zeros—but not for year.

Anyway, you must press ENTER after each entry, since the INPUT statement in line 30 requests three INPUTs. (Some computers allow a comma to separate multiple INPUTs. The T/S 2068 requires ENTER.)

Line 40 now READs two pieces of information from the first DATA line—the number 1 and the string "January"—and calls these a and m$. Notice that "January" is in quotation marks, since it's a string. Next, line 50 compares this number to the first number you INPUT, which was assigned to variable m in line 30. If they aren't the same, the program GOes TO line 40 and picks up the next two DATA items. The program keeps looping back until READ finds a equal to m, then the program GOes TO line 70.

But wait a minute! Suppose that never happens? Suppose you INPUT some number other than 1 to 12? Well, you cheated! All months are numbered 1 to 12! However, this program assumes you are honest and will make no mistakes, so there is no "error trapping" and a number less than 1 or greater than 12 results in an "E Out of DATA, 40:1" error.

Let's go back to line 70 and 80, which print the month, day, and year in conventional form in green INK. A sample screen output is shown in Fig. 7-1. This simple sequential "search" program is

```
Date:MM/DD/YY
The date is:
March 5, 1978

Date:MM/DD/YY
The date is:
February 22, 1956

Date:MM/DD/YY
The date is:
July 23, 1927

Date:MM/DD/YY
The date is:
December 25, 1983

Date:MM/DD/YY
The date is:
October 13, 1922

Date:MM/DD/YY
```

Fig. 7-1. Date converter.

remarkably swift for a small amount of DATA items. However, with 100 or more DATA items, the search time becomes obvious for items far down the "list."

Line 90 starts with a new statement, RESTORE (above the S key). This resets the DATA pointer back to the beginning of the DATA, regardless of where it was just previously. A modified form of RESTORE, not used here, is to specify a DATA line number for the reset point, such as RESTORE 110.

PRINT merely puts a blank line on the screen and GO TO 20 asks for another date.

Club Rosters

LISTing 7-2 and a typical output, Fig. 7-2, show a little more sophistication in the use of READ and DATA statements. With some imagination, you'll be able to use these statements in lots of programs of your own.

Line 20 reminds you to turn on the printer if you want the output printed. Unlike many other computers, which will "freeze" if the printer is addressed but not connected, the T/S 2068 allows the program to continue to the next statement.

Line 30 PRINTs the title, and line 40 tells you what an asterisk after a name indicates. Line 100 asks you which club roster you want. Notice that the INPUT follows on the next line, since you want

LISTing 7-2

```
  10 REM © Fred Blechman 1983
  20 PRINT : PRINT "Turn printer
 on if wanted..."
  30 PRINT : PRINT TAB 8;"CLUB R
OSTERS"
  40 PRINT : PRINT "    (* = Memb
ership Fee Due!)"
 100 PRINT : PRINT "Club Name?";
 110 INPUT n$: PRINT n$: LPRINT
n$
 115 IF n$=" STOP " THEN STOP
 120 READ a$,b$,f
 130 IF a$="end" THEN RESTORE :
PRINT : LPRINT " ": GO TO 100
 200 IF b$=n$ THEN PRINT a$;
 205 IF b$=n$ THEN LPRINT a$;
 210 IF b$=n$ AND f=1 THEN PRINT
 " *",
 215 IF b$=n$ AND f=1 THEN LPRIN
T " *",: GO TO 230
 220 IF b$=n$ THEN PRINT " ",
 225 IF b$=n$ THEN LPRINT " ",
 230 GO TO 120
1000 DATA "Adams","Stamp",0
1010 DATA "Billings","Coin",1
1020 DATA "Chambers","Computer",
0
1030 DATA "Dunn","Computer",0
1040 DATA "Evans","Stamp",1
1050 DATA "Farley","Tennis",1
1060 DATA "Gaines","Tennis",0
1070 DATA "Harrison","Coin",1
1080 DATA "Jenkins","Coin",0
1090 DATA "Jordon","Stamp",1
1100 DATA "Klein","Computer",1
1110 DATA "Krueger","Computer",0
1120 DATA "Larson","Tennis",1
1130 DATA "Mason","Computer",1
1140 DATA "Norris","Computer",0
1150 DATA "Prentiss","Tennis",1
1160 DATA "Roth","Computer",1
1170 DATA "Stevens","Coin",0
1180 DATA "G.Thompson","Computer
",0
1190 DATA "R.Thompson","Computer
",1
1200 DATA "Valdez","Tennis",0
2000 DATA "end","end",0
```

the "Club Name?" question to remain on the screen. When using text with an INPUT statement, it disappears when you provide the INPUT. The semicolon at the end of line 100 means the next PRINTing will follow the existing PRINTing.

```
Turn printer on if wanted...

          CLUB ROSTERS

     (* = Membership Fee Due!)

Club Name?Tennis
Farley *              Gaines
Larson *              Prentiss *
Valdez

Club Name?Computer
Chambers              Dunn
Klein *               Krueger
Mason *               Norris
Roth *                G.Thompson
R.Thompson *

Club Name?STOP
```

Fig. 7-2. Club roster.

Line 110 asks you to INPUT a string. The flashing cursor in the lower left corner will be enclosed in quotation marks to indicate that a string INPUT is expected. This string will be assigned to n$, PRINTed on the screen, and then PRINTed on the printer (if it's connected and on). Let's say, for the purpose of illustration, you type and ENTER "Coin" with a capital C. (You do not need to type the quotes, since the INPUT prompt already has them.)

Line 115 looks to see IF you used STOP for the INPUT. IF you did, the program STOPs. Otherwise, it goes on to line 20 and READs three items from the first DATA statement. It READs "Adams" as a$, "Stamp" as b$, and 0 for f. Line 130 looks to see if "end" was assigned to a$. If so, RESTORE moves the DATA pointer back to the beginning, a blank line is printed on both screen and printer, then the program returns to ask for another Club Name. Notice that the LPRINT has a quote, space, quote after it. A plain LPRINT is ignored by the printer, but even one space forces a line-feed. Anyhow, since a$ is "Adams" at the moment, line 130 is ignored.

Lines 200–225 are only active IF b$ (in this case, "Stamp") is equal to your INPUT "Coin", assigned to n$ in line 110. They are not equal, so line 230 goes back to 120 for another READ. This time "Billings", "Coin", and "1" are assigned to a$, b$, and f.

Now lines 200–225 are active, since b$ is equal to n$, "Coin." Line 200 PRINTs Billings on the screen, and line 205 does the same on the printer. Line 210 finds f is equal to 1 (Membership fee due!) so it prints an asterisk on the screen after Billings, and line 215 does the

same on the printer. The GO TO 230 at the end of line 215 sends the program around lines 220 and 225, which are only used if the "membership flag," f, is a zero.

The commas at the end of lines 210–225 provide for two names on a line by "zone printing" to the second half of the screen line. Line 230 goes back to line 120 for yet another READ, and this continues until the "end" of DATA, line 2000.

Your Own Modifications

Membership status is merely the change of the 1 or 0 at the end of the DATA line for that member. Other clubs could be added. Members are sandwiched alphabetically between the existing DATA statements by using the appropriate line number, since the computer automatically "sorts" line numbers on entry. With program modifications you could PRINT the entire list, or just those whose fees are due. There are all kinds of uses for which a program like this could be used—and the DATA is always within the program itself.

HOORAY FOR ARRAYS!

One of the most confusing BASIC concepts for many beginners is "arrays." It certainly was for me! Maybe what got me was the dictionary definition of an array as "an orderly collection or series of things imposingly displayed." Huh? Then someone said, "Array. Oh, that's just the BASIC word for matrices." Double "huh?" Yet, I had a feeling if I could ever understand arrays they would be extremely useful. Suddenly, one day the dawn broke and the clouds cleared, and I was right—arrays are an incredible addition to your programming "arsenal." I even found out that they kept alligators away. That's right! I never saw an alligator at home in California when I was programming! (For some reason, however, out here in this rotten jungle, despite all my programming, I see those critters all the time.)

Single-Dimension Array

You can think of an array as a bunch of "pigeon holes" arranged in an organized fashion—like safe deposit boxes in a bank, or post office boxes. Each box has a special designation within the entire array, which itself has a name based on a single variable. Parentheses are used and called "sub" when described. For example, a(x) is

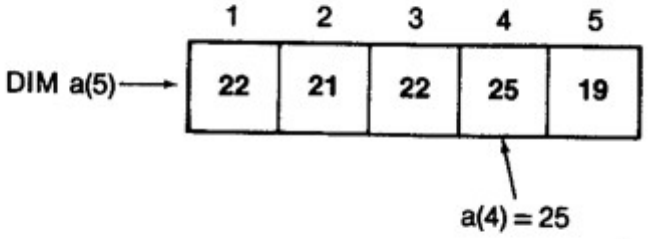


Fig. 7-3. Single-dimension array.

called "a sub-x," a single element of the "a array." Fig. 7-3 will help clarify this.

A DIMension statement must be used to define an array, and this is usually done at the beginning of a program (especially since the DIM statement does some other things you might not want done later on). For example, DIM a(5) tells the computer to set aside space for an array you will call by the letter a, and that it will have five "columns." You do not need to use all the spaces, but Fig. 7-3 shows all the columns filled. The 25, since it is in the fourth column, would be referred to as a(4).

How do the numbers get in those boxes? Try this, with no other program in memory:

```
10 DIM a(5)
20 FOR x=1 TO 5
30 INPUT "Enter number";a(x)
40 NEXT x
```

Each time the "prompt" in the lower left corner asks for a number, type and ENTER the numbers shown: 22, 21, 22, 25, 19. Now, type and ENTER PRINT a(4) and the number 25 should appear on the screen. Try PRINTing the other array box contents.

An easier way to put the numbers in the array is this:

```
10 DIM a(5)
20 FOR x=1 TO 5
30 READ a(x)
40 NEXT x
50 DATA 22,21,22,25,19
```

Arrays can also be SAVEd on tape with the SAVE "filename" DATA a() command and LOADed with the LOAD "filename" DATA a() command.

Two-Dimension Array

Now that you've got single dimensional arrays figured out, the next step is to the much more useful two-dimension array. This much

more closely resembles an arrangement of post office or safe deposit boxes. We assign the number of "rows" (horizontal) and "columns" (vertical) in a DIM statement, in row, column order. For example, DIM a(4,5) defines an array with four horizontal rows and five vertical columns, as shown in Fig. 7-4.

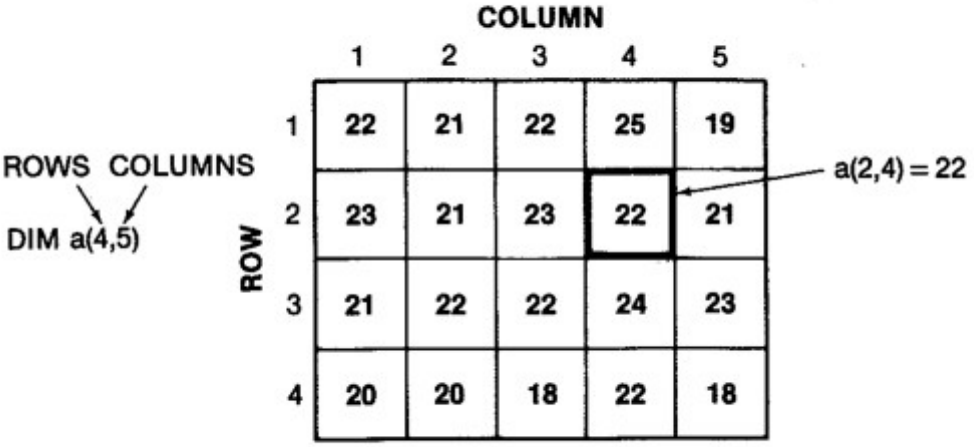


Fig. 7-4. Two-dimension array.

In computereese, "columns" are always the name for the vertical lines, and it's easy to remember that if you think of the huge "columns" on Greek architecture.

The contents of any box in the array can be found by using the array designation. For example, in Fig. 7-4, a(2,4) is 22—the value in the second row, fourth column. Now, really, isn't that easy?

LISTing 7-3 not only loads the array of Fig. 7-4 into memory, it also examines the array and PRINTS the results—an example of array manipulation—as shown in Fig. 7-5.

Let's assume you're a school administrator and you want to know the average attendance for certain classes. Each column in the array represents one of five different classes, and each row represents one week. This program first DIMensions the array (line 100), and then fills the array with the number of students in each class each week (lines 110, 120, 500–530). Lines 200–220 PRINT the array on the screen, for reference. Lines 300–340 add the number of students in each class for the four weeks, divide the total of each by four, and then PRINT the average number of students for each class.

This kind of a program forms the basis for the many "electronic spreadsheet" programs that are so popular. They are, of course, far more sophisticated than this simple program, but this one serves to illustrate that you can examine and manipulate array contents. You can multiply, divide, and all the rest, and you can even move the figures around within the array.

LISTing 7-3

```
 10 REM © Fred Blechman 1983
 20 REM * Array Manipulation *
100 DIM a(4,5)
110 FOR x=1 TO 4: FOR y=1 TO 5
120 READ a(x,y): NEXT y: NEXT x
200 FOR x=1 TO 4: FOR y=1 TO 5
210 PRINT AT x,y*5;a(x,y);" ";
220 NEXT y: NEXT x
300 PRINT : PRINT : LET t=0
310 FOR y=1 TO 5: FOR x=1 TO 4
320 LET t=t+a(x,y): NEXT x
330 PRINT "Class ";y;" average:
";t/4: LET t=0
340 NEXT y
500 DATA 22,21,22,25,19
510 DATA 23,21,23,22,21
520 DATA 21,22,22,24,23
530 DATA 20,20,18,22,18
```

```
22   21   22   25   19
23   21   23   22   21
21   22   22   24   23
20   20   18   22   18

Class 1 average: 21.5
Class 2 average: 21
Class 3 average: 21.25
Class 4 average: 23.25
Class 5 average: 20.25
```

Fig. 7-5. Class attendance.

Multidimensional Arrays

It doesn't end with two dimensions. You can visualize three easily, as shown in Fig. 7-6. Here the numbers could represent additional class attendance information, with Level 1 being January; Level 2, February; and Level 3, March. You could average the figures for all classes for the first week of each month, or any class for three months, or whatever. Are you beginning to see the power of arrays?

Large arrays use *lots* of RAM (user memory). However, within the limitations of memory and your ability to cope with the visualization required, you can have arrays with any number of dimensions. You'll seldom see more than three, however.

There's another point that shouldn't be missed. Arrays are a great place to store variables. If the letters a–z are not sufficient to store your variables, you can store them in an array—which gives you a virtually endless supply of variable names. For example, a(12) or

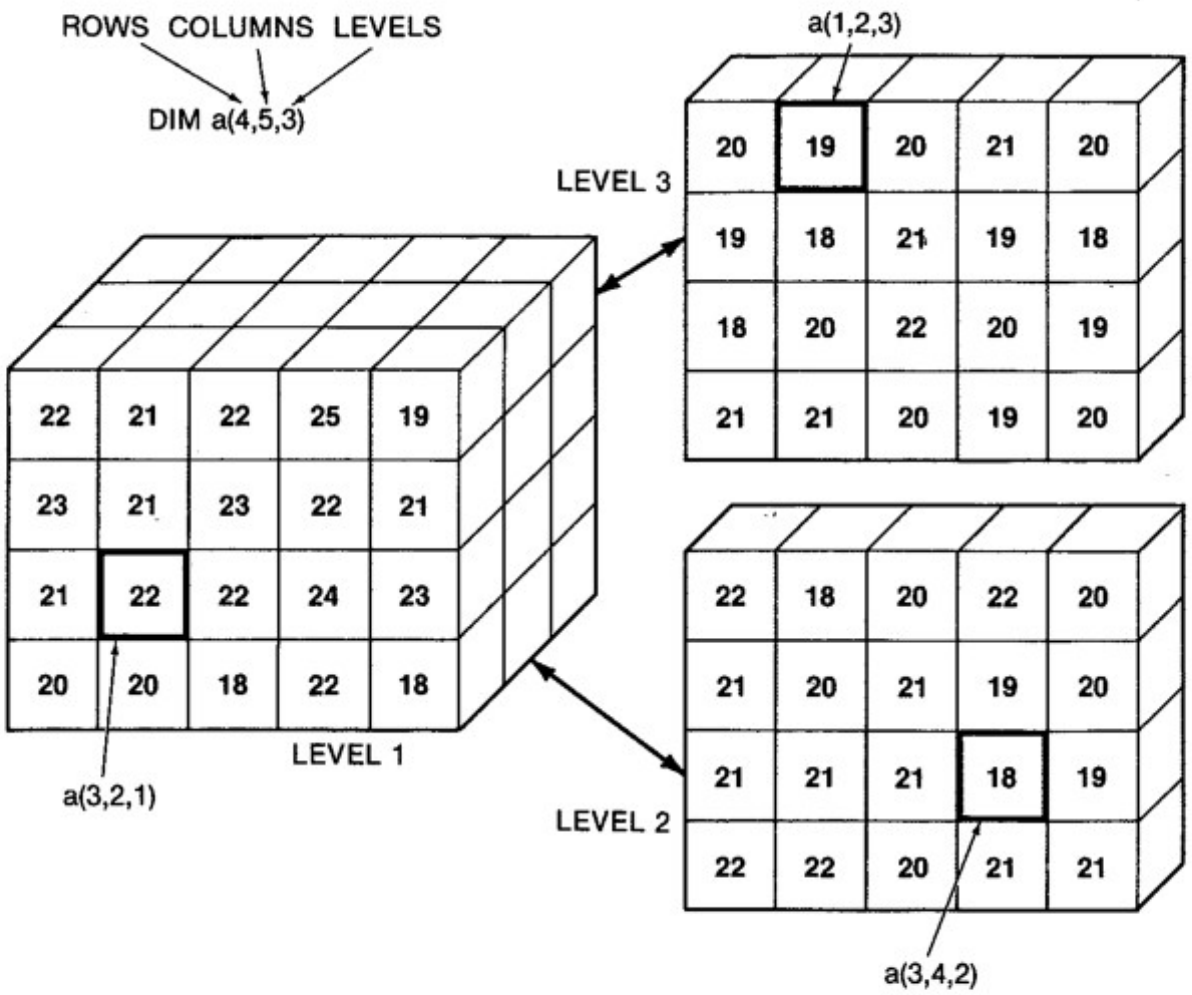


Fig. 7-6. Three-dimension array.

b(6,4) may hold any number you like, and you can pull out, use, change, and reinsert that number in the same (or a different) location. It's all kind of mind boggling when you begin to see its potential.

STRING ARRAYS

We've just discussed arrays which use only numbers. "String arrays," which use any characters, are similar, but have some definite, specific differences. For one thing, you can't perform calculations on strings, but you can juggle them around for sorting. The well-known "bubble sort," and others, use arrays to sort strings.

Another big difference in string arrays is that you must define the maximum string length—anything larger will be "truncated" (cut off), and anything shorter will be followed by blanks to the specified length. Again, the DIM statement is used, but this time with a string symbol. For example, DIM a$(20,10) defines a single dimension string array of twenty columns, each with ten spaces.

Since a$(12), for example, would pull the entire string from the twelfth array location, how does the computer distinguish this from meaning the twelfth *character* of some other a$? It can't, and that's why you can't have simple string variables and array string variables with the same name! In other words, a$ cannot be used if you're going to use an a$ array.

Name and Age Array

Numerical and string arrays can be used in combination, as shown in LISTing 7-4 and its output, Fig. 7-7.

LISTing 7-4

```
 10 REM © Fred Blechman 1983
 20 REM * String Array *
 30 DIM n$(12,10): DIM a(12)
100 FOR x=1 TO 12
110 READ n$(x),a(x)
120 NEXT x
130 PRINT "   Name      Age",
140 PRINT "   Name      Age",
200 FOR x=1 TO 12
210 PRINT n$(x);" ";a(x),
220 NEXT x
300 PRINT " ": PRINT "Last name
s only:"
310 FOR x=1 TO 12
320 PRINT n$(x,3 TO ),
330 NEXT x
500 DATA "A.Allison",34,"R.Bake
r",23,"A.Cornwall",37,"C.Finkel"
,28
510 DATA "S.Garrard",31,"H.Jenk
ins",42,"D.Lawrence",27,"G.Nicol
as",33
520 DATA "S.Pulver",31,"H.Simmo
ns",45,"M.Turnbull",38,"S.Vincen
t",39
```

Line 30 DIMensions two distinct arrays, both single dimension. The n$-array holds 12 names, each with a maximum of 10 characters. The a-array holds 12 ages. Lines 100–120 load the arrays, using the DATA in lines 500–520. It is necessary for the strings to be enclosed in quotation marks, for a comma to be between DATA items, and for the items to be in the proper sequence. You cannot load a string into a numerical array, or a number into a string array (unless it is enclosed in quotation marks—which means it's not a

```
   Name      Age        Name      Age
A.Allison     34    R.Baker        23
A.Cornwall    37    C.Finkel       28
S.Garrard     31    H.Jenkins      42
D.Lawrence    27    G.Nicolas      33
S.Pulver      31    H.Simmons      45
M.Turnbull    38    S.Vincent      39

Last names only:
Allison             Baker
Cornwall            Finkel
Garrard             Jenkins
Lawrence            Nicolas
Pulver              Simmons
Turnbull            Vincent
```

Fig. 7-7. String array.

number anymore, but a string). If you try, you'll get a "C Nonsense in BASIC" error.

Lines 130 and 140 PRINT a heading, and lines 200–220 PRINT the contents of the arrays. Lines 300–330 perform string slicing of the names and cut off the initials to PRINT only the last names.

Multidimensional String Arrays

As with numerical arrays, the number of dimensions you can have in a string array is limited only by the computer's memory and your ability to keep track of the notation. In a multidimensional string array, the last number is the number of characters per string. For example, DIM c$(2,3,5,10) establishes an array with 2 rows, 3 columns, 5 levels and 10 characters per array location.

Advanced BASIC programming books cover sorts and searches using arrays, but be careful! Most BASICs don't assign specific string lengths to arrays (the T/S 2068 does), and this can have a definite effect on searches and sorts (which use string comparisons) unless taken into account.

Once again I encourage you to spend some time with READ, DATA, RESTORE, and arrays if you plan on doing any programming. They are very powerful!

QUESTIONS

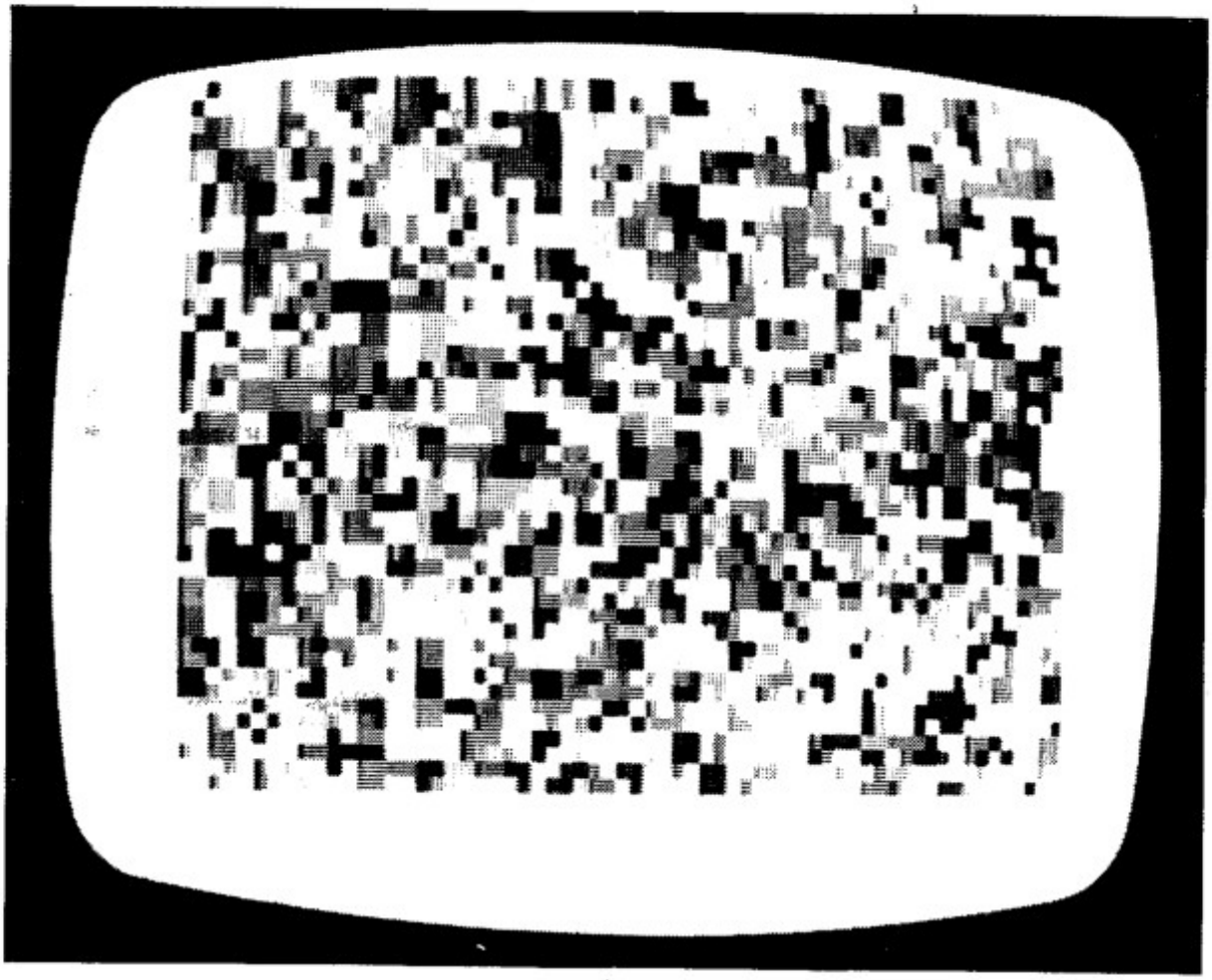
1. Are DATA statements sequential or random?
2. Can DATA statements be anywhere within a program? Must they all be together?

3. Can RESTORE be directed to a particular DATA line?
4. Can numerical DATA include calculations?
5. Are commas ever used in DATA statements? What for? How about quotation marks?
6. Is READ used with numbers or strings?
7. How many DIMensions are allowed for numerical arrays? String arrays?
8. Describe this array: c$(3,4,5,6)
9. Can strings from arrays be "sliced" like any other strings?

SECTION C
ADDING THE FRILLS

Now that you've made it through to "The Valley of Sight and Sound," you've come to where all the fun is! Gather all the natives around and dazzle them with the colors your T/S 2068 can produce, using the programs in Chapter 8. Then add the graphics (Chapter 9) and sound (Chapter 10) for a good ole' "jungle jamboree"!

8
Color Me Gorgeous!



The color mosaic, as seen on a black-and-white screen.

INTRODUCTION

One of the big new features of the T/S 2068 Series Personal Computers, compared to the T/S 1000 and T/S 1550, is C-O-L-O-R! We've dabbled a little with color in previous chapters, but now we're ready to go into some more depth. While color is not particularly important for most simple computer functions, it has become, together with sound, almost a necessity in games. Our emphasis in this book is on practical uses—not games—but there's no doubt that color can enhance almost any program. This chapter will equip you with enough facts about the color capabilities of the 2068 to allow you to become somewhat of a "pixelated Picasso."

THE COLOR SIGNALS

Although all the promotional information on the T/S 2068 mentions that it has eight colors, there is little mention of the two distinct BRIGHTness levels which make these eight colors actually appear to be 16 colors.

First, however, we must clarify what can be colored and what cannot, and how well these colors can be viewed. For example, if you're using a black-and-white TV set as a display, then you will see the colors as different shades of gray—and even these won't show up too well unless you tune the brightness and contrast controls for the best distinction. (These control settings will probably not be the same ones used to get the best TV picture.)

Using a Color TV Receiver

If you're using a color TV set, you'll be feeding the modulated RF signal from the 2068 into the antenna switch box supplied with the 2068. This is certainly usable for the regular 32-characters-per-line display, and even high resolution graphics, but won't be good enough for Display Mode 2, which displays 64 characters on a line. Display Mode 2 is discussed in detail in the follow-up to this book, "The Timex 2068 Intermediate/Advanced Guide" by Jeffrey Mazur. When using a color TV receiver, you'll be distracted by "crawlies," wiggly lines that pervade the picture and are particularly noticeable on borders between colors. You'll learn to live with crawlies (like I've learned to live with mosquitos in this cursed jungle hut).

It's important that the antenna switch box has the switch in the "Computer" position and not the "TV" position. Also, be sure the channel selector is properly set, and then tune for the clearest picture.

Color Monitor

If you use a color *monitor,* you'll have an improved display. Connect the video cable supplied with the computer from the computer's MONITOR output (on the back of the 2068) directly to the color monitor's video input. An adapter plug may be required if the monitor doesn't have an RCA-type video input jack. The antenna box is not used with a monitor. Using a monitor, rather than a TV set, won't absolutely eliminate crawlies, but it greatly cuts down their population and size!

RGB Monitor

Now, if you're really looking not only for the best picture, but the sharpest, most accurate colors, then you'll want to use an "RGB Monitor." That's not a brand name. It's a special type of color monitor, and instead of using the standard NTSC composite color video signal, it processes individual red, green, and blue signals directly—hence the name "RGB." To access these signals on the 2068, you'll need a cable (and possibly some interface circuitry) to connect the proper "fingers" on the 64-pin card-edge connector on the back of the T/S 2068 to the RGB monitor input.

Don't fool around with this unless you know what you're doing, but the RGB signals are there for those with RGB monitors. For details on interfacing with an RGB monitor, see the *Timex 2068 Intermediate/Advanced Guide* mentioned earlier.

THE COLOR PALETTE

There are various commands/statements that affect the color of the display and each picture element (pixel) on the display. Fig. 8-1 shows the various areas of color and the associated BASIC control words. The BORDER, for example, surrounds the usable area of the display, and is controlled directly with the BORDER number. This can be done from the keyboard (which is then called a "command") or within a program (when it's called a "statement").

PAPER and INK

The usable area has a background color called "PAPER," on which the printing occurs in an "INK" color, which certainly makes sense. So, whenever you PRINT anything, it's printed in the INK color on the PAPER background of the usable area (22 lines) surrounded by the BORDER color. The additional two display lines, normally used for EDIT, INPUT, and program line entry are the current or previous BORDER color.

Each character space (and in this book we're only covering 32-character lines) has a PAPER color, and every pixel in that space ($8 \times 8 = 64$) has that color unless PRINT, DRAW, PLOT, INVERSE, or OVER has caused that pixel to assume the INK color. The most important thing to remember is that there is only one PAPER color and one INK color for the entire character space at any one time.

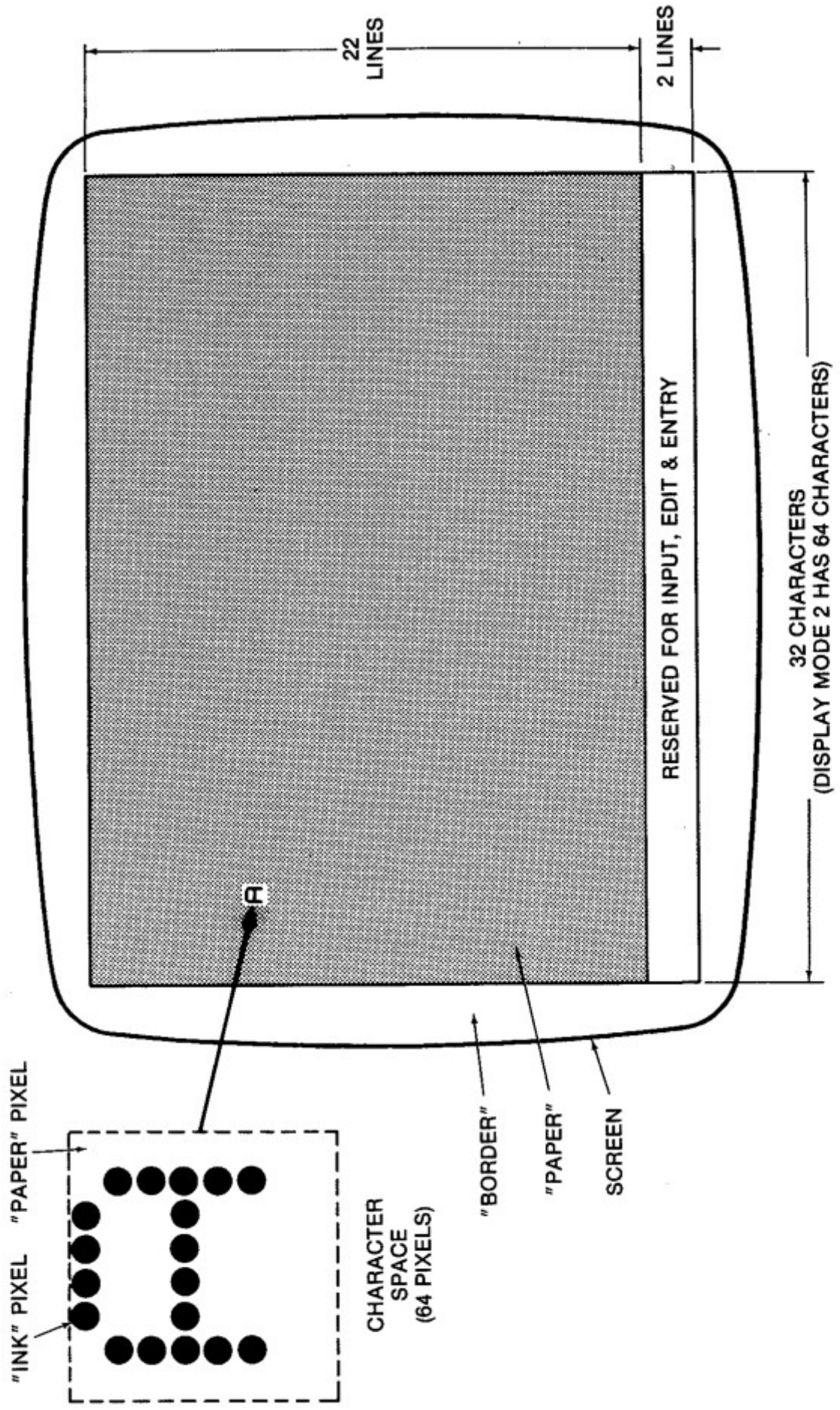


Fig. 8-1. Display modes 1, 3, and 4.

Each color can change, but it affects every pixel in that character space. In other words, you cannot control the color of individual pixels within a character space. (In the *Timex 2068 Intermediate/ Advanced Guide* you'll find that individual rows of eight pixels within a character space can have different colors, but that's only in Display Mode 4.)

The other thing to understand is that each pixel can be in only one of two states—either the PAPER color (in which case it is sometimes referred to as "off") or the INK color ("on").

What Are the Colors?

Your T/S 2068 keyboard shows the colors above eight of the keys on the top row. It is not coincidental that these colors are referred to within the T/S 2068 by the numbers of those keys. For example BORDER 3 is magenta, and BORDER 6 is yellow. The same system applies to all the color commands, so you don't need to memorize the color codes or look in a book—they're right on the keyboard.

Type and ENTER LISTing 8-1 for the color "palette"—the eight different colors, and their two BRIGHTness levels. Line 110 starts a FOR-NEXT loop, sets the variable p equal to 0 to start and counts up to a value of 7. This line also sets PAPER to the value of p. That means the next PRINT statement will change the PAPER color. Notice that no INK color is specified, mainly because we won't be using any INK, just PAPER, for this program.

LISTing 8-1

```
100 REM * Color Palette *
110 FOR p=0 TO 7: PAPER p
120 PRINT ,: BRIGHT 1: PRINT ,:
BRIGHT 0
130 PRINT ,: BRIGHT 1: PRINT ,:
BRIGHT 0
140 NEXT p
```

Line 120 PRINTs 16 blank spaces across the screen (see the comma?), changes the BRIGHTness level to "1," or "on," PRINTs another 16 spaces for the rest of the screen line, and then goes back to BRIGHT 0 ("off"). Line 130 does exactly the same thing, but on the next screen line. Don't forget the commas after both PRINT statements in each line, or you'll get line-feeds you don't want. Line 40, of course, simply goes back to increase the value of p in line 110

if it has not yet reached 7. The result is you get two screen lines in each of the seven colors.

Notice that the left side of the screen shows the regular colors in the sequence shown above the number keys, and the right side shows the same colors, but BRIGHTer. On a properly adjusted black and white TV, you'll be able to distinguish not only the colors but the brightness levels, except for the black. On a color TV or NTSC monitor, this "palette" will allow you to adjust for the proper colors (the yellow, for example, should not be orange), but the brightness differences might be hard to detect in the low-number colors. An RGB monitor should show it all!

PAPER AND INK

LISTing 8-2 changes the colors of the PAPER, the INK, and even the BORDER. Line 110 sets the PAPER color in a loop, and line 120 sets the INK and BORDER color in another "nested" loop. In other words, the INK/BORDER loop performs the line 130 PRINT statement eight times (for i values of 0 to 7, with p equal to 0), and then the NEXT p in line 140 does it all over again, but with p equal to 1. This continues until p finally exceeds 7, at which point the program "falls through the loop" to line 150, and the PAPER and INK colors are reset.

LISTing 8-2

```
100 REM * Paper & Ink *
110 FOR p=0 TO 7: PAPER p
120 FOR i=0 TO 7: INK i: BORDER
i
130 PRINT "PAPER ";p;"/INK ";i,
140 NEXT i: NEXT p
150 PAPER 7: INK 0
```

Which INK on Which PAPER?

RUN this program (pressing Y or ENTER when "scroll?" appears at the bottom of the screen) and you'll see the many PAPER and INK combinations available (64, in this program)—and how many of them are almost unreadable! Obviously, in every case where PAPER and INK are the same color, you can't see the PRINTing, even though it's there. Table 8-1 shows the suggested color combinations

for best readability. Actually, if you don't need color for your application, it's vary hard to beat black INK on white PAPER for legibility.

Table 8-1. Best Paper and Ink Combinations for Maximum Legibility

Paper Color & No.		Ink Color No.
Black	0	3, 4, 5, 6 or 7
Blue	1	4, 5, 6 or 7
Red	2	6 or 7
Magenta	3	0, 6 or 7
Green	4	0 or 1
Cyan	5	0 or 1
Yellow	6	0, 1 or 2
White	7	0, 1, 2 or 3

To FLASH or Not To FLASH?

Another attribute that affects a character space is FLASH. Just to add excitement to this program, add line 125:

```
125 FLASH 1
```

Also, add FLASH 0 at the end of line 150 to reset the FLASH to "off" at the end of the program, so you can look at a LISTing without going batty. Now, RUN the program and you'll see each color FLASHing—but so much is going on, it's hard to tell exactly what's happening. If you watch closely, you'll see that each section of the screen changes its PAPER color to its INK color when FLASHING. Then it goes back to normal—all in about a half a second.

Use Y or ENTER to continue to the end of the program when "scroll?" appears at the bottom of the screen, and then press the L key for LIST. The program LISTing will appear, pushing up the colors—which keep on FLASHing. So, even if you move a FLASHing character, it keeps on FLASHing until you use FLASH 0 either from the keyboard or in the program. Another way of saying this is that the attributes (PAPER, INK, BRIGHT and FLASH) apply to the character when it was originated or last changed, not the screen location.

Attributes "8" and "9"

The number "8" may be used with PAPER, INK, FLASH, and BRIGHT, but not with BORDER. The "8" is used as a precaution to

keep from changing the existing attribute. PAPER 8, INK 8, FLASH 8, or BRIGHT 8 simply leave these attributes in the character attribute file as they were—no change. This is called a "transparent" instruction, since it allows the original attribute to continue. If you're putting something on the screen and you don't want to change its PAPER, INK, FLASH or BRIGHT mode, then use "8" (without the quotes, such as INK 8).

The "9" is much more useful, but can only be used with PAPER and INK. If INK 9 is used, it assures you that any PRINT instruction will be in either black or white INK, to make it most visible against the PAPER color. For PAPER colors black, blue, red, and magenta (0-3), white INK is used. For PAPER colors green, cyan, yellow, or white (4-7), black INK is used. Similarly, PAPER 9 will provide white PAPER for INK 0-3 and black PAPER for INK 4-7.

Try It Yourself

So you don't believe the T/S 2068 is really that smart? Well, here's a "four-liner" (which could be a "one-liner") to prove it. Use NEW to clear memory, then type, ENTER, and RUN this short program:

```
10 FOR x=0 TO 7
20 PAPER x: INK 9
30 PRINT "Testing....."
40 NEXT x
```

Pretty smart machine, eh? The commands DRAW, PLOT, INVERSE, and OVER can all have an effect on color as well, but they'll be covered in the next chapter, "Grappling With Graphics."

THE MANDATORY MOSAIC

One cannot cover the subject of using color on computers without presenting a randomly generated mosaic pattern. You'll find mine, simple as it is, in LISTing 8-3. Line 20 uses the RANDOMIZE statement (shown as RAND on the T key) all this does is assure that you will never get the same pattern twice.

Line 30 picks RaNDom values (actually, pseudo-RaNDom values, but we decided to avoid that distinction) for variables x and y. The value of x can fall anywhere from zero to 21, and y can be up to 31.

LISTing 8-3

```
10 REM * Color Mozaic *
20 RANDOMIZE 0
30 LET x=RND*21: LET y=RND*31
40 LET z=RND*15+128
45 IF z<129 THEN LET z=129
50 LET p=RND*7: LET i=RND*7
60 PAPER p: INK i: PRINT AT x,
y;CHR$ z
70 GO TO 30
```

Line 40 sets variable z equal to a number from zero to 15, plus 128, or a range of 128 to 143. You'll see why in a moment.

Let's skip line 45 for now. We'll come back to it. Line 50 sets variable p and i each to a number from 0 to 7. Line 60 does the real work, by printing a graphic character with PAPER color p, and using INK color i, at a screen location x rows down and y character spaces to the right of the upper left hand corner. Huh? Read it again, and if you're still confused, check Chapter 3 "Display Primer" to refresh yourself on PRINT AT coordinates.

But what "graphic character" does it print? Variable z in CHR$ determines that. Variable z can be any number (line 40) from 128 to 143. If you'll look in your T/S 2068 Manual, you'll find that the T/S 2068 character set assigns these numbers to the low-resolution graphics characters. So CHR$ z simply prints that character at the PRINT AT position specified by x and y. Line 70 then routes the program to line 30 to do the whole thing again. After about 10 minutes the screen is completely full of a graphic quiltwork in all the colors. It looks like an infrared photo of the San Fernando Valley in California on a summer day, with red (hot) houses and blue (cool) pools.

Oh, you still want to know about line 45, huh? Well, it just so happens that PAPER, INK, PRINT AT, and CHR$ are all smart enough to round random numbers up or down to the nearest integer. In other words, 13.4 becomes 13, but 13.5 becomes 14. This is no problem with PAPER, INK or PRINT AT in this program (it could be in others, however, so be aware of this odd characteristic), but it could produce a z value of 128. This is a blank space character that pokes holes in the mosaic, so line 45 eliminates this chance.

Fig. 8-2 shows how the Timex 2040 Personal Printer "sees" a partially complete mosaic. Since it's "color blind," it only prints INK dots, regardless of color (even if they are the same color as PAPER dots, and therefore "invisible" on the screen!).

Fig. 8-2. How the printer sees the mosaic (INK only).

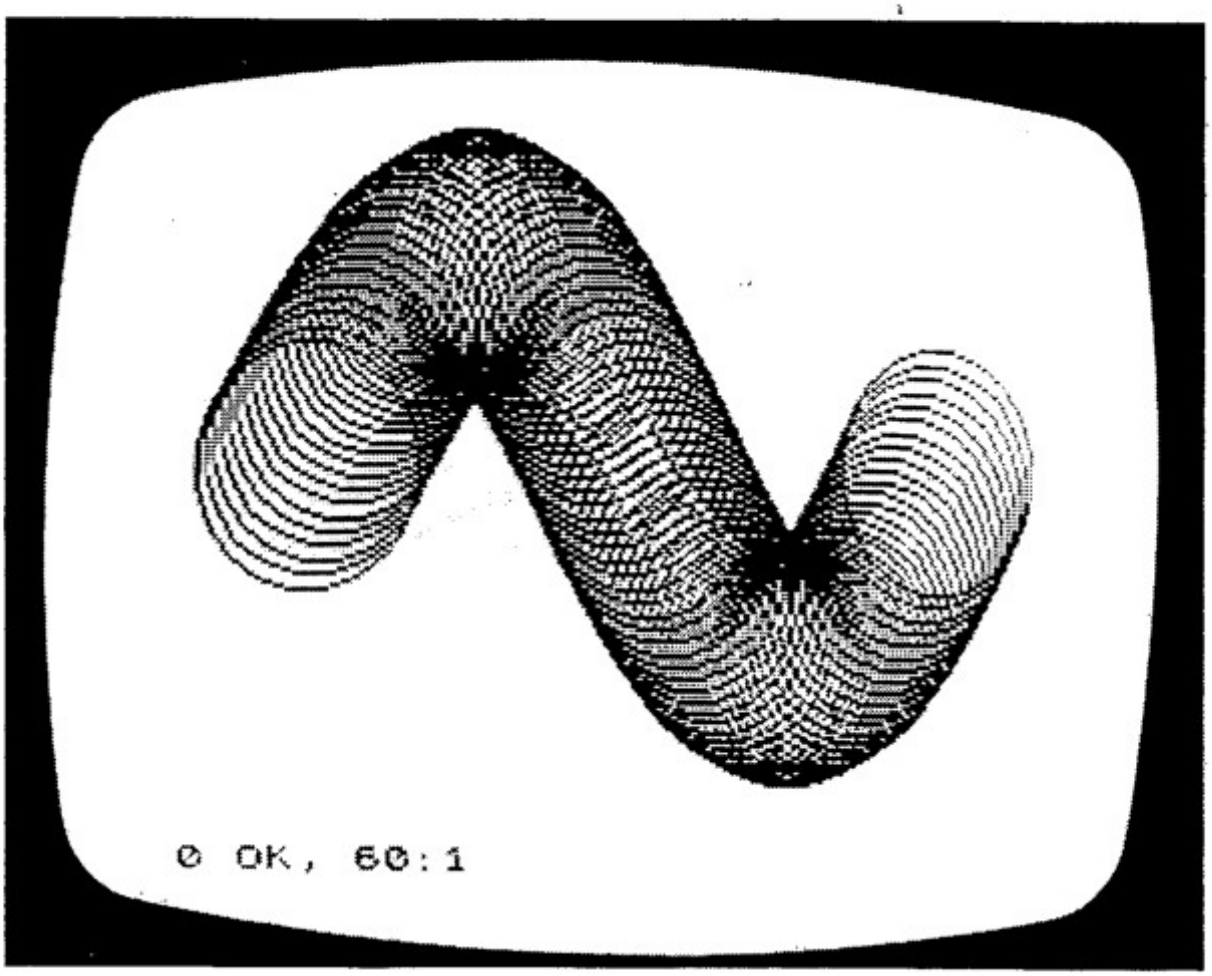
What If *You're* Color Blind?

If you are part of the population that is color-blind, it doesn't matter anyhow . . . . . In fact, this whole chapter could have been skipped! ("Now he tells me . . . . .")

QUESTIONS

1. How many colors can the T/S 2068 produce? How many brightness levels?
2. How do the colors appear on a black-and-white TV?
3. Display Mode 1 has how many character spaces on a line? How many pixels on a line?
4. How many PAPER colors can there be in any one character space at any one time? INK colors?
5. How many different colored pixels can appear at one time in a given character space? How many pixels in a character space?
6. How many BORDER colors are available? Is BORDER 8 valid?
7. A character space that normally has yellow PAPER and red INK is set to FLASH. What are its FLASH colors for PAPER and INK?
8. Describe CHR$ number 133.
9. What is the range of numbers of RND*15? Could it ever be rounded up to 16?

9
Grappling With Graphics



"A SINe of spring."

INTRODUCTION

In Chapter 3 you were introduced to some of the graphics available in Display Mode 1. The other three Display Modes will be covered in the *Timex 2068 Intermediate/Advanced Guide* as well as ultrahigh color resolution graphics. In this volume, we are concerned only with block graphics, special characters, high resolution pixel graphics, and geometric shapes—which we'll cover in this chapter.

Graphics hold a fascination all their own but require a special kind of patience and perseverance. If you find yourself not particularly interested in computer "art," it might be best to jump over this chap-

ter, and come back to it another time when your creative juices are flowing. However, if you intend to do any game programming, this chapter is really a "must." As we say out here in the jungle, "You can eat an elephant—one bite at a time!" We'll try to ingest graphics "one *byte* at a time."

CHARACTERS & PIXELS

In Chapter 3 we discussed Display Mode 1. Take a look at Fig. 3-1 just to refresh your memory. There are 32 character positions on each of 22 lines, for a total of 704 character positions. The additional two lines below these (screen lines 22 and 23) are reserved for the computer to use during line entry or EDITing, or INPUT statements, so we don't consider them part of the user-available display.

Each character position is actually made up of 64 picture elements, called "pixels" or "dots." The tendency is to call these individual elements "pixels" when they appear on a screen where they make up a "picture," and "dots" when they are printed out from a printer, since that term more accurately describes how they appear on paper.

Playing With Blocks

Before we get too deeply involved with pixels, let's first make sure you understand the low-resolution or "block" graphics available directly from the keyboard. On keys 1–8 you'll see small square symbols. These shapes are used in programs in a PRINT or PRINT AT statement, using the GRAPHICS mode.

The *light* part of the symbol is INK, not PAPER as you might expect—since INK is usually *darker* than PAPER.

> Note: This is true in early models of the 2068. These symbols could be reversed in later production. How can you tell which you have? If the 8 key shows a black block, then the light part of the graphic block symbols on your keyboard represents INK, and the black is PAPER. To reverse these on the screen, use the SYMBL SHIFT key when in the G-cursor mode, or go into INV.VIDEO before entering the G-cursor mode.

For example, suppose you want to PRINT INK in the upper right corner of a character space. The symbol on the 1 key is light (INK)

in the upper right corner. First use CAPS SHIFT and 9 to get into the GRAPHICS mode. This will put a G-cursor on your screen. Now if you press the 1 key, that symbol will appear on your screen. To get the reverse of that symbol, hold down the SYMBL SHIFT key when you press the 1 key.

Block graphics may also be obtained using the CHR$ function, as shown in the "Color Mosaic" program in Chapter 8. Appendix B shows each of the block graphic characters (there are a total of sixteen, counting the "blank") and several ways to access them. Note that Table B-1 shows the INK portions of each block in black, the conventional way to think of them (though this is the opposite of the T/S 2068 keyboard symbols).

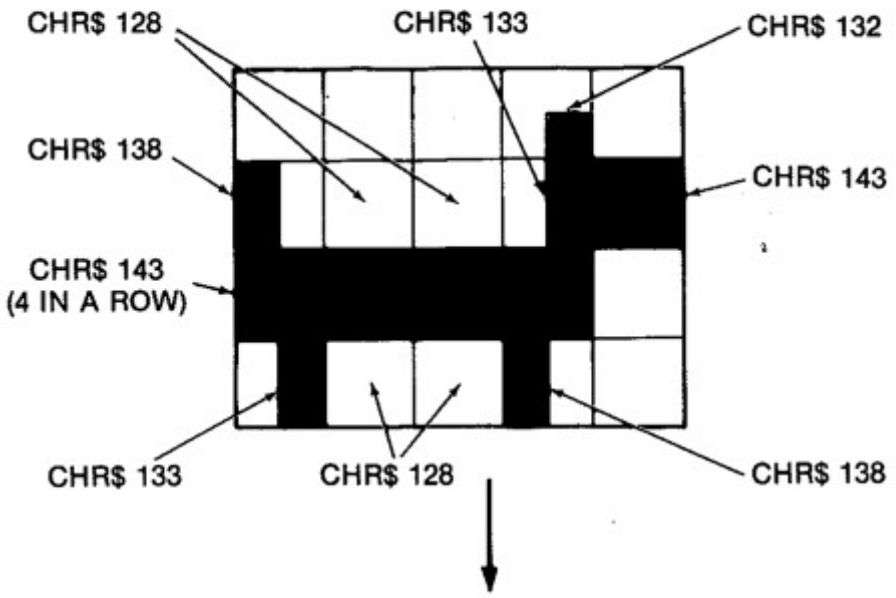
Beware! Vicious Doggie!!

Low-resolution graphics are handy for borders to "dress up" displays and can be "concatenated" (added together) to provide symbolic forms. Fig. 9-1, a RUN of program LISTING 9-1, shows a "graphic dog" made up of various low resolution graphic blocks. LISTing 9-1 not only draws the "dog" on your screen, but moves it along the top lines in slow animation, and "barks" as it goes.

The program is not complex, but illustrates several programming techniques. Line 5 puts a warning on the screen. Line 10 starts a FOR-NEXT loop, used with line 120, to "move" the dog across the screen. Actually, it establishes print locations along the line for the graphic blocks. Lines 20–50 actually "assemble" the dog from different blocks on one screen line at a time, using PRINT AT statements. Line 20 PRINTs on the first screen line (0), starting at the fourth character location (0+3, since y is equal to 0 for this first time through the loop). What does it print? CHR$ 132, which you can see from Appendix B is the lower right corner of a block—and in this case is the dog's ears!

Lines 30, 40, and 50 PRINT AT, on successive lines, the CHR$ blocks that assemble the rest of the dog. Line 30 forms the tail and head, line 40 the body, and line 50 the legs.

Line 60 checks to see if the "dog" has reached the right side of the screen; if it has, the program stops with the dog displayed. Line 70 does the "barking" by PRINTing "ARF!" at RaNDom locations below the title. The RND*14+7 provides a PRINT AT line number from 7–21, and the RND*27 gives numbers for the PRINT AT location along the line from 0–27 leaving room for the four characters of "ARF!"



```
  BEWARE! VICIOUS DOGGIE!!

ARF!
                        ARF!

   ARF!
ARF!RF!
    ARARF!ARF!   ARF!
                       ARFARF!
                     ARF!
          ARF!!ARF!      ARF!
                     ARF!   AARF!
ARF!!            ARF!          ARF!
   AARF!         ARF!

                     ARF!
```

Fig. 9-1. Vicious doggie!

Lines 80–110 "erase" the figure by printing blank spaces in those locations where the dog has been placed. Now line 120 increments the value of y by one and sends the program back to line 10, which moves the dog forward (to the right) one space and draws it again.

Incidentally, the limit of 27 in line 10 keeps the program from coming up with an "out of range" error, which occurs if you try to draw the dog off the right side of the screen. Since the dog is five spaces long, and the maximum right space is number 31, we sub-

LISTing 9-1

```
   5 PRINT AT 5,4;"BEWARE! VICIO
US DOGGIE!!"
  10 FOR y=0 TO 27
  20 PRINT AT 0,y+3;CHR$ 132
  30 PRINT AT 1,y;CHR$ 138+CHR$
128+CHR$ 128+CHR$ 133+CHR$ 143
  40 PRINT AT 2,y;CHR$ 143+CHR$
143+CHR$ 143+CHR$ 143
  50 PRINT AT 3,y;CHR$ 133+CHR$
128+CHR$ 128+CHR$ 138
  60 IF y=27 THEN STOP
  70 PRINT AT RND*14+7,RND*27;"A
RF!"
  80 PRINT AT 0,y;"      "
  90 PRINT AT 1,y;"      "
 100 PRINT AT 2,y;"      "
 110 PRINT AT 3,y;"      "
 120 NEXT y
```

tract four spaces from 31 to get 27. (27+28+29+30+31 equals five spaces.)

Logo Design

There's another way to use block graphics effectively, as shown in Fig. 9-2. Here, the programming was done by placing graphic blocks in PRINT AT statements directly from the keyboard, as shown in LISTing 9-2. Notice how regular text can be mixed with the graphics (lines 20, 30, 40, 600, 610 and 710). Even a CIRCLE is drawn (we'll

Fig. 9-2. A company logo, using block graphics. (Amway is a registered trademark of the Amway Corporation.)

LISTing 9-2

```
  10  REM  * Amway Logo * © Fred
 Blechman 1983
  20 CLS : PRINT AT 5,10;"C O M
P U T E R"
  30 PRINT AT 7,10;"P R O G R A
M S"
  40 PRINT AT 9,14;"F O R"
  50 PRINT AT 4,1;  "    ▄▄"
  60 PRINT AT 5,1;  "   ▐  ▌"
  70 PRINT AT 6,1;  "  ▐    ▌"
  80 PRINT AT 7,1;  "  ▌    ▐"
  90 PRINT AT 8,1;  "  ▌    ▐"
 100 PRINT AT 9,1;  " ▐      ▌"
 110 PRINT AT 10,1; " ▐      ▌"
 120 PRINT AT 11,1; " ▌▀▀▀▀▀▀▐"
 130 PRINT AT 12,1; " ▌▀▀▀▀▀▀▐"
 140 PRINT AT 13,1; " ▌      ▐"
 150 PRINT AT 14,1; "▐        ▌"
 160 PRINT AT 15,1; "▐        ▌"
 170 REM
 200 PRINT AT 11,8; "▐▌  ▐▌"
 210 PRINT AT 12,8; "▐▌  ▐▌"
 220 PRINT AT 13,8; "▐▌  ▐▌"
 230 PRINT AT 14,8; "▐▌  ▐▌"
 240 PRINT AT 15,8; "▐▌  ▐▌"
 250 REM
 300 PRINT AT 11,13; "▐▌    ▐▌"
 310 PRINT AT 12,13; "▐▌    ▐▌"
 320 PRINT AT 13,13; " ▐▌  ▐▌"
 330 PRINT AT 14,13; " ▐▌  ▐▌"
 340 PRINT AT 15,13; "  ▀▀  ▀▀"
 350 REM
 400 PRINT AT 11,20; " ▄▄▄▄  "
 410 PRINT AT 12,20; " ▐    ▌"
 420 PRINT AT 13,20; " ▐    ▌"
 430 PRINT AT 14,20; " ▐▄▄▄▄▌"
 440 PRINT AT 15,20; " ▀▀▀▀▀▀"
 450 REM
 500 PRINT AT 11,25;    "▐▌  ▐▌"
 510 PRINT AT 12,25;    "▐▌  ▐▌"
 520 PRINT AT 13,25;    " ▐▌▐▌"
 530 PRINT AT 14,25;    " ▐▌▐▌"
 540 PRINT AT 15,25;    "  ▐▌"
 550 PRINT AT 16,25;    "  ▐▌"
 560 PRINT AT 17,22; " ▄▄▄▄▌"
 570 PRINT AT 18,22; " ▐  ▐▌"
 580 PRINT AT 19,22; " ▀▀▀▀"
 600 PRINT AT 17,5;"P R O D U C
T";
 610 PRINT AT 19,0;"D I S T R I
B U T O R S";
 700 CIRCLE 227,27,8
 710 PRINT AT 18,28;"R"
 720 FOR x=0 TO 7
 730 BORDER x
 740 FOR d=1 TO 50: NEXT d: NEXT
 x: GO TO 720
```

be getting to that soon) and, as an attention-getting device, the border continuously changes colors after the logo is drawn.

In order to do this for your own design, use the video layout shown on page 152 of the T/S 2068 User Manual as a worksheet. Lay out your design lightly in pencil until it looks like what you want, then enter the proper blocks and text into a program. LISTing 9-2 is an example for you to use as a guide, but other approaches could be used. Any way you do it, it's tedious, but the results are gratifying. Of course, color and FLASHing can be added, using the information from Chapter 8.

SPECIAL CHARACTERS

Chapter 3 and Chapter 8 got into the definition of "pixels" ("picture elements," or individual screen dots); now get ready for the "advanced course"! Here you'll learn how to create up to 21 "special characters" of your own design and access them directly from the keyboard. This is not simple to explain, but it *is* simple to do.

"Comes the Resolution!"

The T/S 2068 Personal Computer allows you to program every single pixel in each character location. Since each character space is eight pixels across and eight pixels high, that means there are 64 pixels in each (and every) of the 704 character spaces in the regular display area of the screen (not even including the two lower lines used for EDIT, INPUT and line entry). That's 64 times 704, or 45056 individual pixels—and that's just in the high resolution modes. In the ultra-high resolution modes, covered in the next book *(Timex 2068 Intermediate/Advanced Guide),* there are 90112 programmable pixels!

How Many Special Characters?

You can assign any or all of the letter keys on your keyboard from A to U (that's 21 of them) to contain "special characters," or characters you design yourself. Now, this will really startle you, but it's a fact: The characters you design may have any of over 18 billion-billion shapes!

This is hard to believe, but here's the "proof." If you have a box divided into two parts, there are four possible light/dark combina-

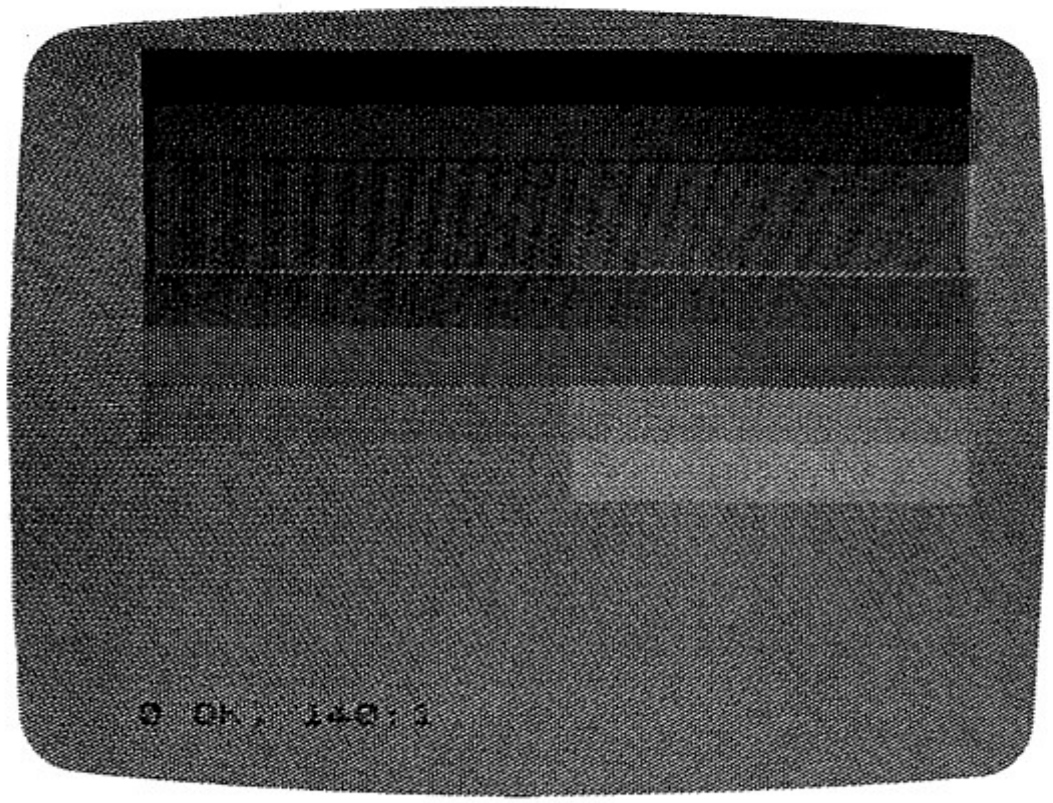


Photo 1. The eight available colors, shown in both brightness levels.

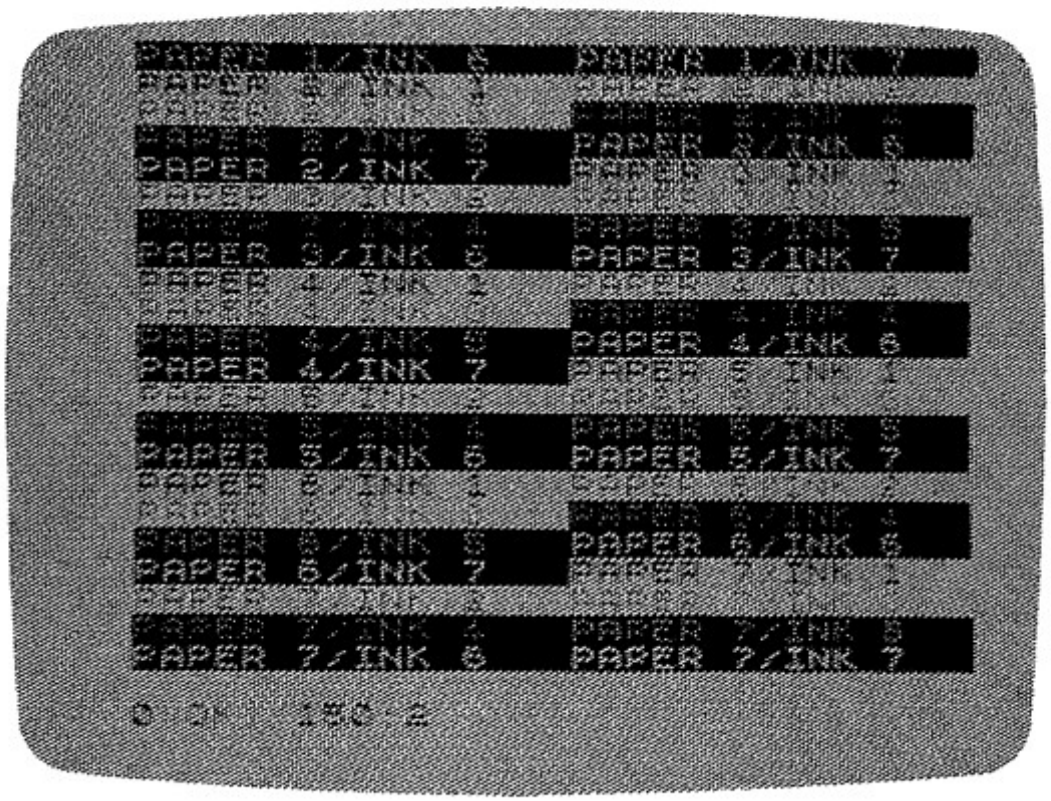


Photo 2. The result when you specify PAPER 9 in Listing 8-2.

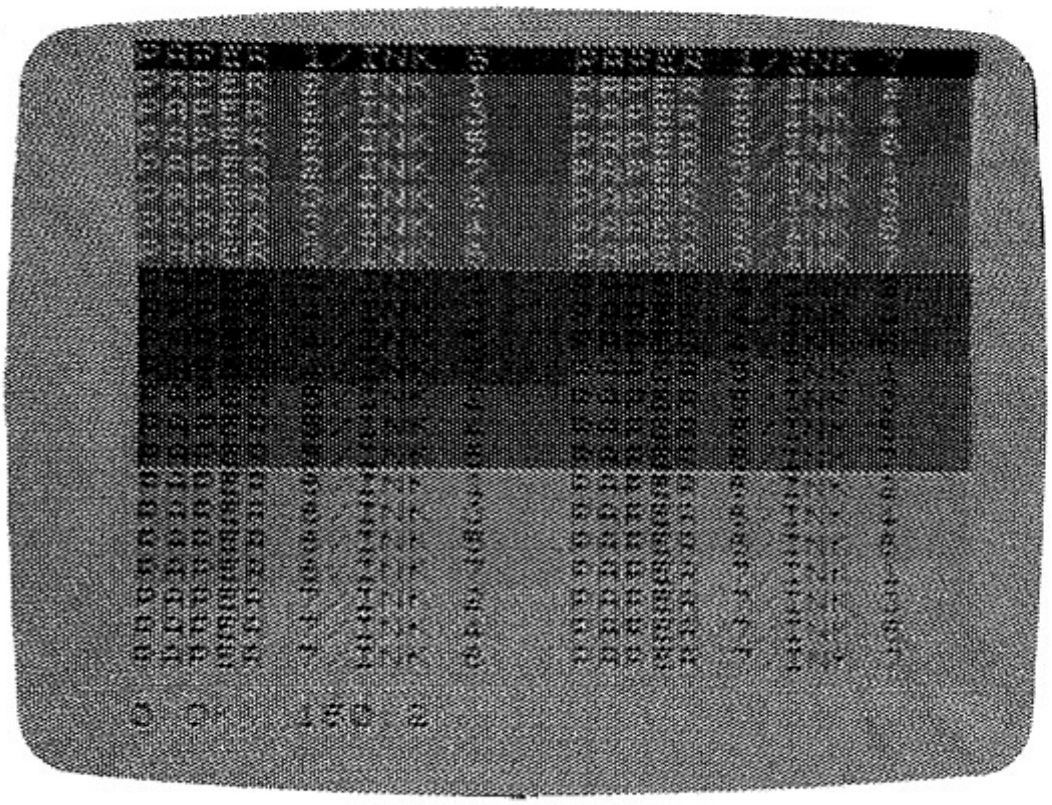


Photo 3. Conversely, the result of specifying INK 9 in Listing 8-2.

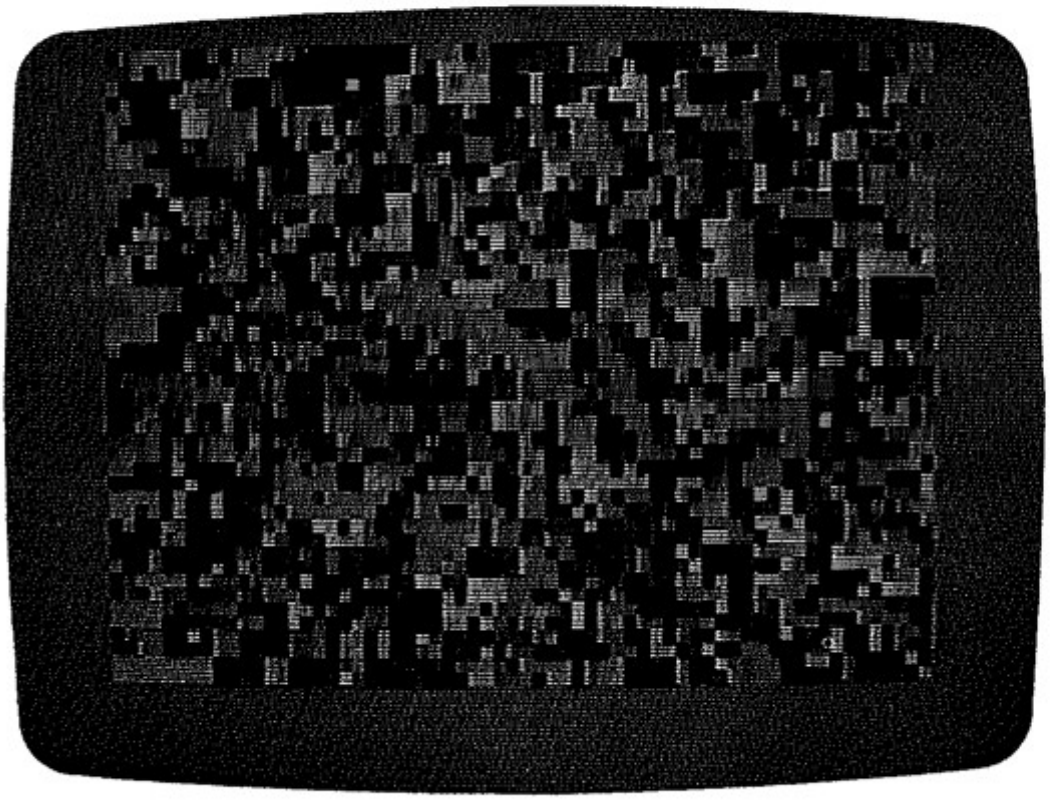


Photo 4. The color mosaic (Chapter 8) uses only low-resolution graphic block, yet achieves a multiplicity of color variations.

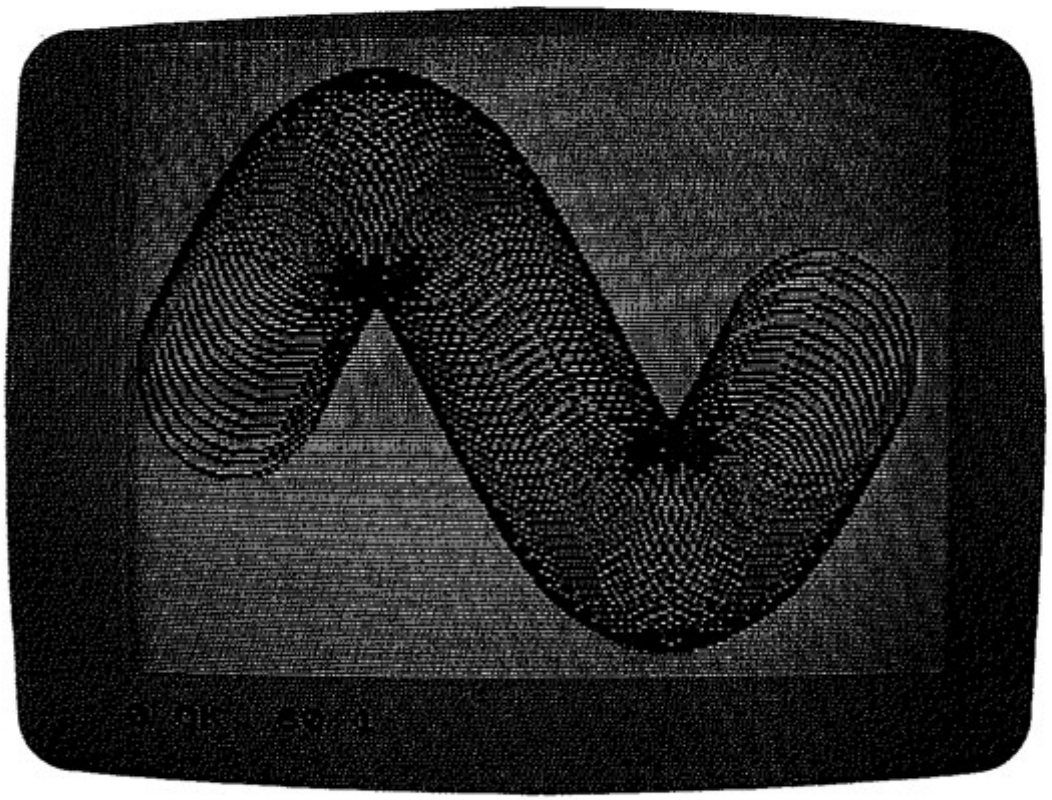


Photo 5. The program from Chapter 9, but with BORDER 5, PAPER 6, and INK 2. (You can add color from the keyboard before RUNning the program, or you can add a line to the program.)

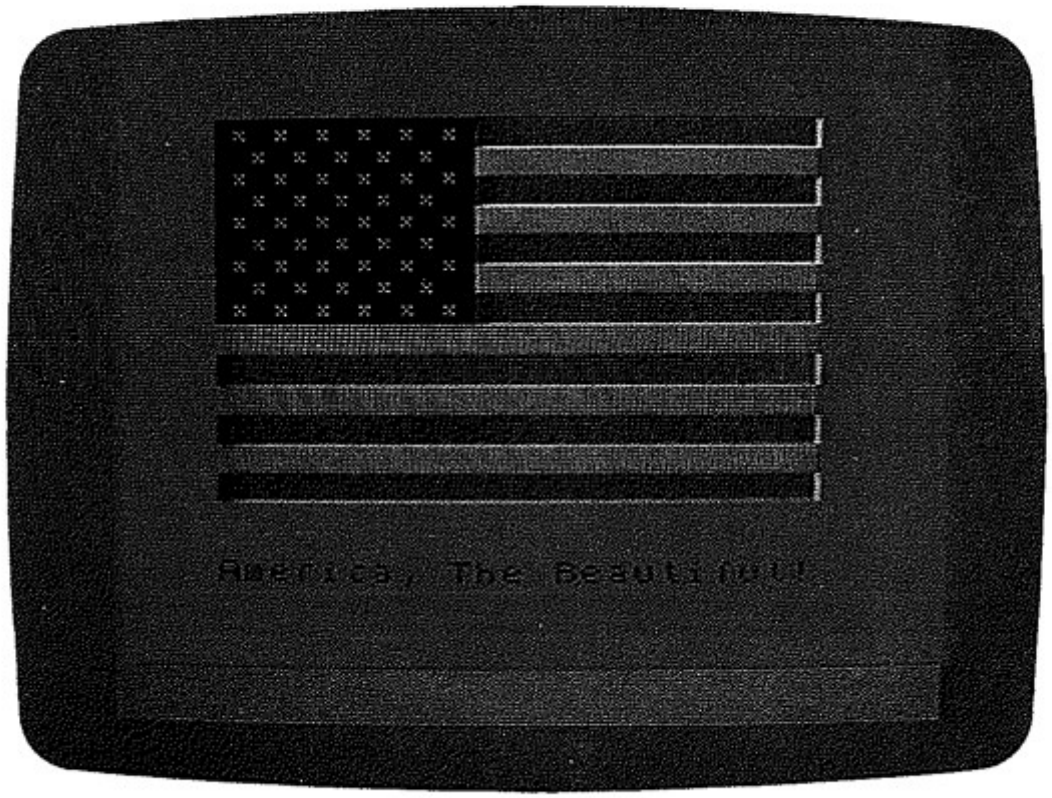


Photo 6. Old Glory (Chapter 8) uses block graphics for the stripes, pixel graphics for the stars.

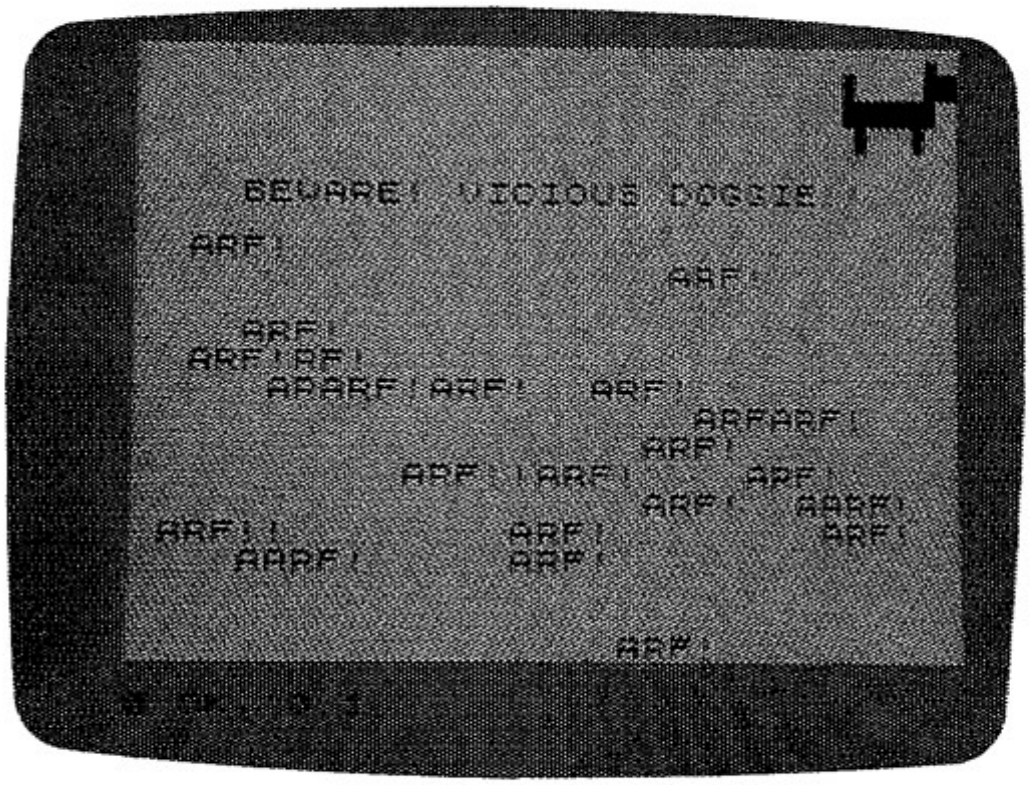


Photo 7. "Beware! Vicious Doggie!" (Chapter 9) illustrates simple animation.

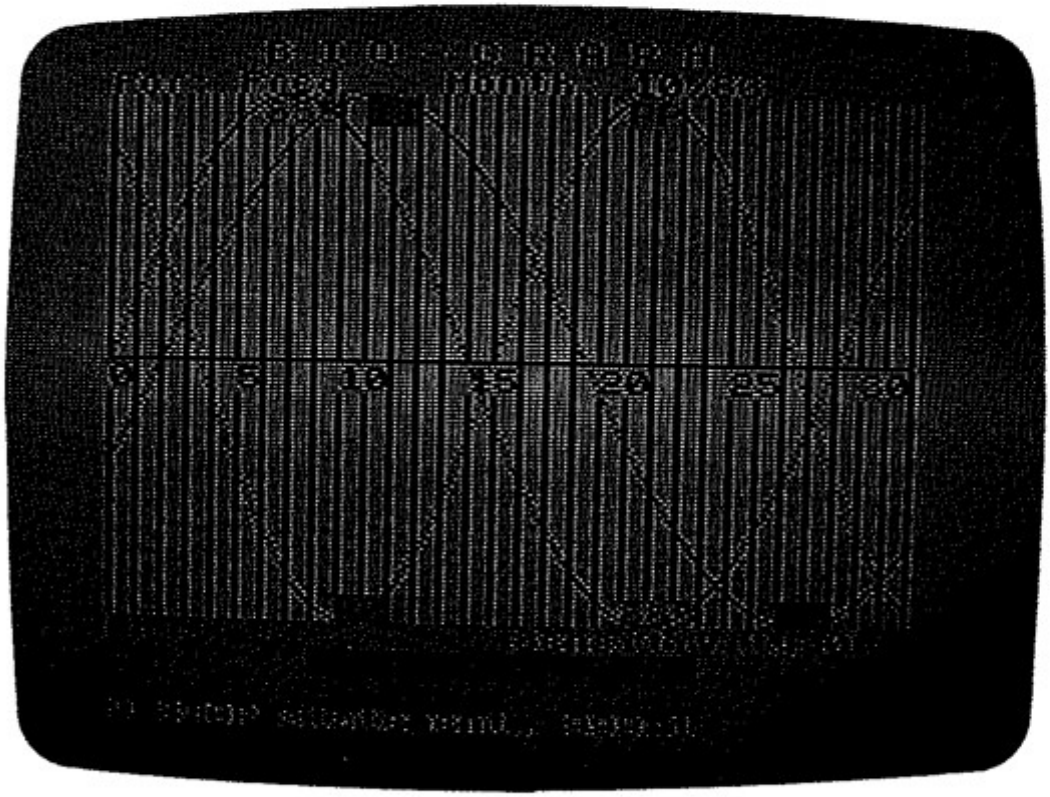


Photo 8. The Bio-Graph (Chapter 13) shows the high-resolution, curve-plotting capability of the T/S 2068, and how color can be used for highlighting. (BORDER 1 was added to the program for contrast with the graph.)

tions—2 to the 2nd power. If you have a box divided into three parts, there are eight possible light/dark combinations—2 to the 3rd power. These are shown in Fig. 9-3, which also shows the number of combinations for a box divided into four parts—16 combinations, and this happens to be 2 to the 4th power. See what's happening? The number of possible "two-color" combinations for a box with n sections is merely 2 raised to the n power. This follows a binary progression so that, for example, a box with six sections can have 2 to the 6th power, or 64 possible light/dark combinations. It follows, then, that a box with 64 sections (pixels) can have 2 to the 64th power number of

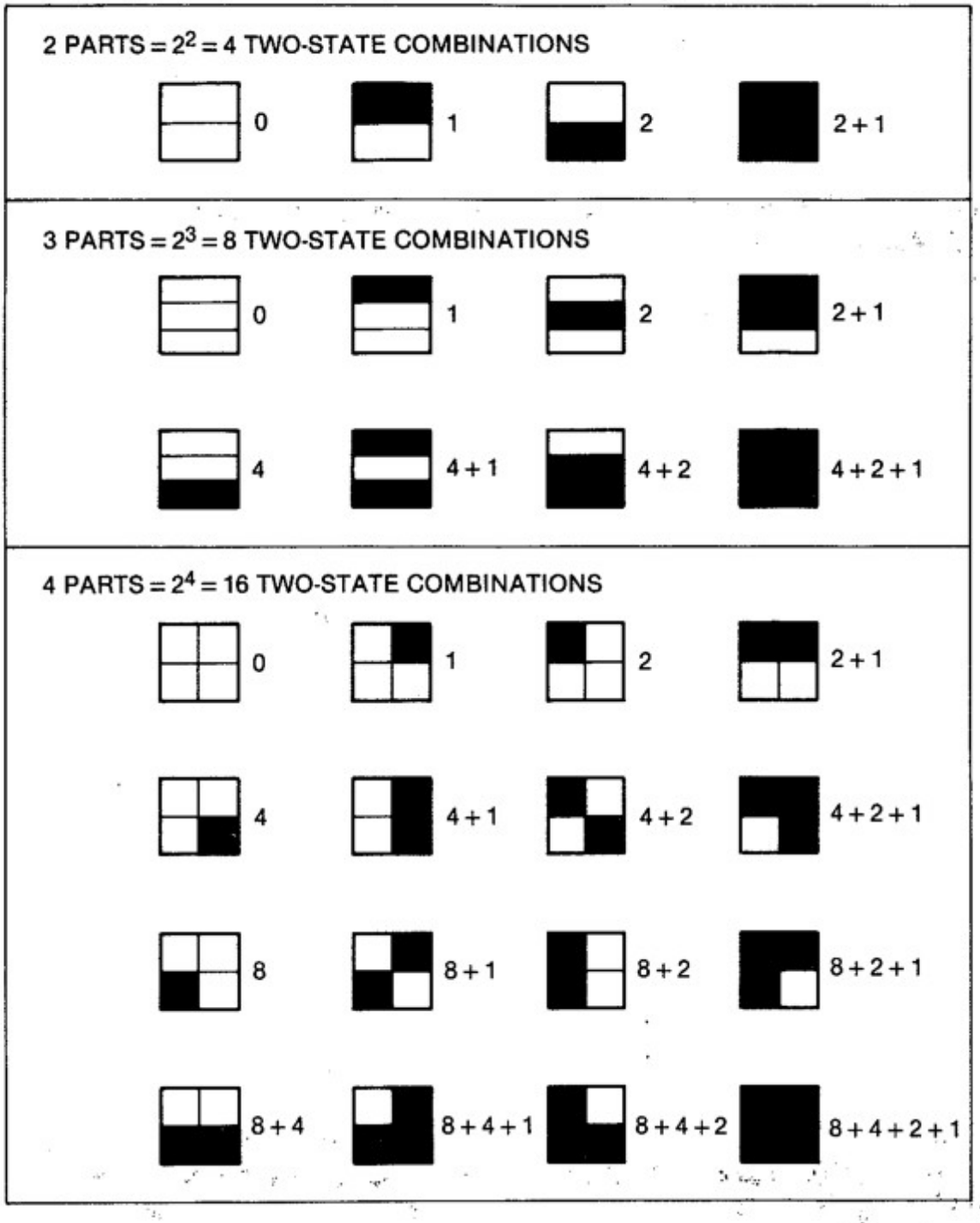


Fig. 9-3. Two-state combinations.

light/dark (PAPER, INK) combinations—and that's 1.8446744 × 10 to the 19th power, or 18,446,744,000,000,000,000!

Now, I know this is hard to swallow, so let me prove it to you. Or, let the T/S 2068 do it for you, with this simple program:

```
10 LET x=0
20 LET z=2(up-arrow)x      (up-arrow on H key)
30 PRINT x,z
40 LET x=x+1
50 GO TO 20
```

Fig. 9-4 shows the results up to the 64th power. Now do you believe me?

So What?

Well, out of over 18 billion-billion possibilities, you only can use 21 special characters at one time. Too bad! But what can you do with them? For one thing, practically any game can benefit from special characters. Why have a graphic block for a "Martian Invader" when you can design your own single-block or multiple-block image. That's right, you can even combine blocks. Let's see, if we joined four character spaces together to form a single special character, that would be 256 pixels. Hmmmm. 2 to the 256th power is . . . . . Forget it!

But there are more practical uses, and the whole process is really not complicated. In Chapter 11, "On Your Mark," we'll use two special characters to represent a waiting and running man (or woman, of course). In Chapter 16, "The Music Maker," special characters are used to display musical "notes" on the screen. In this chapter we'll cover the process of designing your own character and assigning that character to a particular key.

THE BIN STATEMENT

Just above the B key you'll find BIN. This stands for BINARY, and it tells the computer that the next eight digits to be input will be in "binary" form. Don't be alarmed. We'll get into binary numbers in Chapter 14, "BINARY Banner." For now, just accept that this will be an eight-digit number of just ones and zeros, such as 10001101.

Fig. 9-5A shows a "Running Man" plotted in a character space, with 64 pixel locations of eight "rows" and eight "columns." Each one

```
10 LET x=0
20 LET z=2↑x
30 PRINT x,z
40 LET x=x+1
50 GO TO 20
```

```
 0                    1
 1                    2
 2                    4
 3                    8
 4                    16
 5                    32
 6                    64
 7                    128
 8                    256
 9                    512
 10                   1024
 11                   2048
 12                   4096
 13                   8192
 14                   16384
 15                   32768
 16                   65536
 17                   131072
 18                   262144
 19                   524288
 20                   1048576
 21                   2097152
 22                   4194304
 23                   8388608
 24                   16777216
 25                   33554432
 26                   67108864
 27                   1.34217773E+8
 28                   2.68435546E+8
 29                   5.36870912E+8
 30                   1.07374182E+9
 31                   2.14748365E+9
 32                   4.29496730E+9
 33                   8.58993459E+9
 34                   1.71798692E+10
 35                   3.43597384E+10
 36                   6.87194767E+10
 37                   1.37438953E+11
 38                   2.74877907E+11
 39                   5.49755814E+11
 40                   1.09951163E+12
 41                   2.19902326E+12
 42                   4.39804651E+12
 43                   8.79609302E+12
 44                   1.75921860E+13
 45                   3.51843721E+13
 46                   7.03687441E+13
 47                   1.40737488E+14
```

Fig. 9-4. Powers of two.

```
48  2.8147498E+14
49  5.6294995E+14
50  1.1258999E+15
51  2.2517998E+15
52  4.5035997E+15
53  9.0071992E+15
54  1.8014398E+16
55  3.6028797E+16
56  7.2057594E+16
57  1.4411519E+17
58  2.8823038E+17
59  5.7646075E+17
60  1.1529215E+18
61  2.305843E+18
62  4.611686E+18
63  9.223372E+18
64  1.8446744E+19
```

Fig. 9-4—cont.

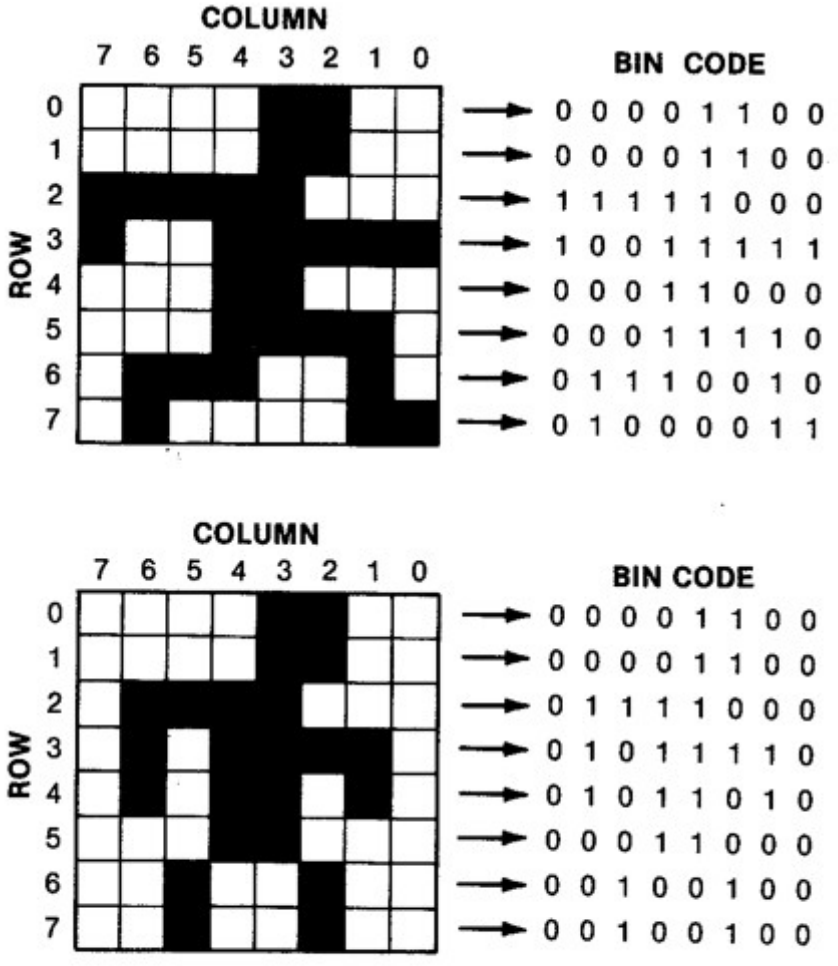


Fig. 9-5. Special characters.

of these can be "on" (INK) or "off" (PAPER). A darkened square means that pixel is "on." If you want to design a special character, you start by darkening in any of the 64 pixel locations with a pencil

Each "on" pixel will be considered a "1" in the BIN statement, and an "off" pixel will be a "0." Then, reading from left to right, you generate a BIN code for each row.

Fig. 9-5B shows a "Standing Man" special character and the BIN code. These are the special characters used in Chapter 11. To get these characters into your computer, you need to first decide to which of the keys from A to U you want to assign them. Say you want to assign Fig. 9-5A to the letter M, for "man." That choice having been made, you simply type and ENTER LISTing 9-3, very carefully.

This is a good place to remind you that you can duplicate line numbers rather easily on a T/S 2068. Wherever there are several lines that are similar, as in this case, just use the EDIT and DELETE functions to change the line numbers and any characters on the line. Remember that you'll need to hold down the CAPS SHIFT and 0 keys until the auto-repeat function defeats the desire of the computer

LISTing 9-3

```
100 POKE USR "m"+0,BIN 00001100
110 POKE USR "m"+1,BIN 00001100
120 POKE USR "m"+2,BIN 11111000
130 POKE USR "m"+3,BIN 10011111
140 POKE USR "m"+4,BIN 00011000
150 POKE USR "m"+5,BIN 00011110
160 POKE USR "m"+6,BIN 01110010
170 POKE USR "m"+7,BIN 01000011
200 PRINT "m","M","M"
```

to put the keyword DELETE on the screen as you try to "erase" the existing line number! When you change the line number and press ENTER, the line will be entered as a new line without affecting the original line! Neat, huh?

Type, Do Not RUN!

Anyway, back to the matter at hand. When LISTING 9-3 has been entered, do not RUN it yet. First type this in directly from the keyboard (no line number):

PRINT "m","M","M"

The second capital M should be a GRAPHICS M. You get it by holding down the CAPS SHIFT key while pressing the 9 key. This will give you a flashing G-cursor. Now press the M key and you'll get what appears to be an uppercase M. You have actually accessed the

place in memory where special characters are kept. Although they are programmable, when the T/S 2068 is powered-up these character locations are initialized by using the standard uppercase characters.

After typing the closing quote that follows the GRAPHICS M, press ENTER and you'll see a lowercase m, an uppercase M, and (below the m) another uppercase M. Now RUN the program, which puts your special character into the GRAPHICS M location. Notice that even though the program used the lowercase "m," it makes no difference.

Now when you type the above line again (remembering that the last M is a GRAPHICS M) and press ENTER, you'll see the little running man in place of the last M. Notice that the regular lowercase m and the uppercase M have been unaffected.

From now on, until you either reprogram this special character or turn off your 2068, this little running man is in the GRAPHICS M spot in memory. Even NEW or CLEAR will not bother him. Anytime you want him on the screen or the printer, just use PRINT (GRAPHICS)M or LPRINT (GRAPHICS) M. Are you beginning to feel the power of this?

LISTing 9-4A

```
100 POKE USR "a"+0,BIN 00001100
110 POKE USR "a"+1,BIN 00001100
120 POKE USR "a"+2,BIN 01111000
130 POKE USR "a"+3,BIN 01011110
140 POKE USR "a"+4,BIN 01011010
150 POKE USR "a"+5,BIN 00011000
160 POKE USR "a"+6,BIN 00100100
170 POKE USR "a"+7,BIN 00100100
200 PRINT "a","A","A"
```

LISTing 9-4B

```
100 POKE USR "a"+0,BIN 00001100
110 POKE USR "a"+1,BIN 00001100
120 POKE USR "a"+2,BIN 01111000
130 POKE USR "a"+3,BIN 01011110
140 POKE USR "a"+4,BIN 01011010
150 POKE USR "a"+5,BIN 00011000
160 POKE USR "a"+6,BIN 00100100
170 POKE USR "a"+7,BIN 00100100
200 PRINT "a","A","🏃"
```

A Standing Man

LISTing 9-4A is the program for Fig. 9-5B—a standing man rather than a running one. Notice the addition of line 200, with the last "A" being a graphic A. Also, notice the difference of the LISTing after (LISTing 9-4B) and before a RUN. Line 200 now shows the special graphic figure even in the LISTing!

It doesn't take a lot of imagination to see the possibilities that special graphic characters can create, and remember they can be combined in PRINT statements to form larger special characters. As a matter of fact, that's what's coming next.

MUSIC, ANYONE?

Music is not my strong suit, but I'm giving it pretty extensive coverage in Chapters 10 and 16 of this book. As a challenge to myself, I decided to see if I could design some special graphic characters to use in displaying music on the T/S 2068 screen as the melody is played. You'll see that done in Chapter 16. In some cases as many as three special characters have to be combined to get a single "note" on the screen.

The results are shown in Appendix C. The LISTings and the graphics on which they are based are shown for keyboard special characters b, c, d, e, f, g, h, j, k, l, m, n, and o—13 special characters. Some compromises with accuracy were made, but the advantage is that, once the special characters are entered into your computer, they are relatively easy to use.

PIXEL GRAPHICS

Chapter 3 introduced you to pixels, PLOT, DRAW and INVERSE. Fig. 3-1 describes the coordinate system for locating pixels with the PLOT instruction. Bear in mind that a pixel is only one "dot" out of a possible sixty-four in each character space, and it can only be an INK color (which we'll call "on") or a PAPER color (which we'll call "off"), as discussed in Chapter 8. It's important to stress that all pixels in a given character space have the same "attributes"—PAPER, INK, FLASH, and BRIGHT—so you cannot have more than two different colored pixels in the same character space (except when using Display Mode 4, ultra-high color resolution, not discussed here). Every

"on" pixel in a character space will be the last specified INK color, and every "off" pixel in a character space will be the last specified PAPER color.

The PLOT Thickens

Let's use PLOT with color to illustrate this important restriction to high resolution graphics. Use NEW to clear any resident program and then type and ENTER these lines:

```
10 INK 5
20 FOR x=0 TO 21
30 PRINT "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"  (32)
40 NEXT x
50 INK 2: PLOT 127,87
```

When you RUN this program, the screen fills with light blue (cyan) As, and a small red dot appears above and to the right of an A near the center of the screen—and that entire A is now red also! The colors chosen should allow you to see the effect even on a black and white TV. The "INK 2" statement in line 50 has changed all subsequent INK to red, and when the dot is punched into that character location, all the "on" dots (in this case, the letter A) also turn red.

At first this seems like a real bummer, but let's try to turn this "lemon" into "lemonade." While the PLOT instruction is nice for drawing thin lines, sometimes the lines are too thin. The effect just described can be used to "thicken" PLOT and DRAW lines. Add this to the existing program:

```
60 DRAW 32,0: DRAW 0,32: DRAW -32,0: DRAW 0,-32
```

Now when you run the program you get a square of red As among all the cyan As. But this doesn't quite get the point across, so now change line 30:

```
30 PRINT "(32 SHIFTed graphic-8 solid blocks)"
```

The 32 blocks in line 30 now paint the entire program portion of the display cyan. Lines 50 and 60, although they actually command only individual pixels, change entire character spaces to the new red ink color, so that you have a neat, "thick" red square on a cyan background. While this may not be the most exciting graphic "stunt" you've seen, it can be useful in such applications as drawing bar

Inverted Logic

The INVERSE 1 instruction simply swaps the PAPER and INK colors "until further notice" if used separately, or just for the next instruction if used within a PRINT statement. (The same "rule" also applies to PAPER, INK, INVERSE, OVER, BRIGHT, and FLASH). Chapter 3 described this briefly, by using PLOT, DRAW and INVERSE 1 to create what appeared to be a thin FLASHing line. All this was done on a plain white screen. Now let's do the same with a colored screen to see the different effect. Add these lines to the existing program:

```
70 PLOT 0,0: DRAW 175,175
80 PLOT INVERSE 1;0,0: DRAW INVERSE 1;175,175
90 GO TO 70
```

Now when you RUN the program, in addition to the red square, you get a ragged red line going diagonally up the screen, with a thin white line "pulsing" through it. Hmmmm. What's going on? Have I had too much of this native concoction they claim to make from berries (but I suspect may be brewed from fermented mangos)? Well, if you understand how these commands work, it all falls into place. The PLOT and DRAW in line 70 merely draw a thin line from the lower left corner to a spot 175 pixels over and 175 pixels up. However, the PLOT instruction is actually an INK command, so it changes the INK color of each pixel along the way. Since there can only be one INK color in each character space, then it follows that all the pixels in each character space "intercepted" by this thin line become red. Remember, we have colored all the pixels with cyan INK in line 10.

Line 80, which is the same as line 70 but with the INVERSE 1 added after PLOT and DRAW, is intended to "unDRAW" this line by

LISTing 9-5

```
  10 INK 5
  20 FOR x=0 TO 21
  30 PRINT "
"
  40 NEXT x
  50 INK 2: PLOT 127,87
  60 DRAW 32,0: DRAW 0,32: DRAW
-32,0: DRAW 0,-32
  70 PLOT 0,0: DRAW 175,175
  75 PAUSE 30
  80 PLOT INVERSE 1;0,0: DRAW IN
VERSE 1;175,175
```

reversing the PAPER and INK colors. Indeed it does—but only for the pixels on the thin line. The white PAPER color (which we never changed) replaces the INK for this thin line, but does not affect the rest of the INK pixels in those character spaces. Confusing, isn't it? To slow down the action, add PAUSE 30 as lines 75 and 85. LISTing 9-5 shows this whole program, in case you got lost along the way. Also, to illustrate how "colorblind" the printer is, Fig. 9-6 shows how the printer "sees" the screen. Despite the various contrasting colors, only the thin white PAPER line shows against a background of INK.

Beware! Curves Ahead!

There's more to the DRAW instruction than meets the eye. By adding another comma and parameter after the x and y displace-

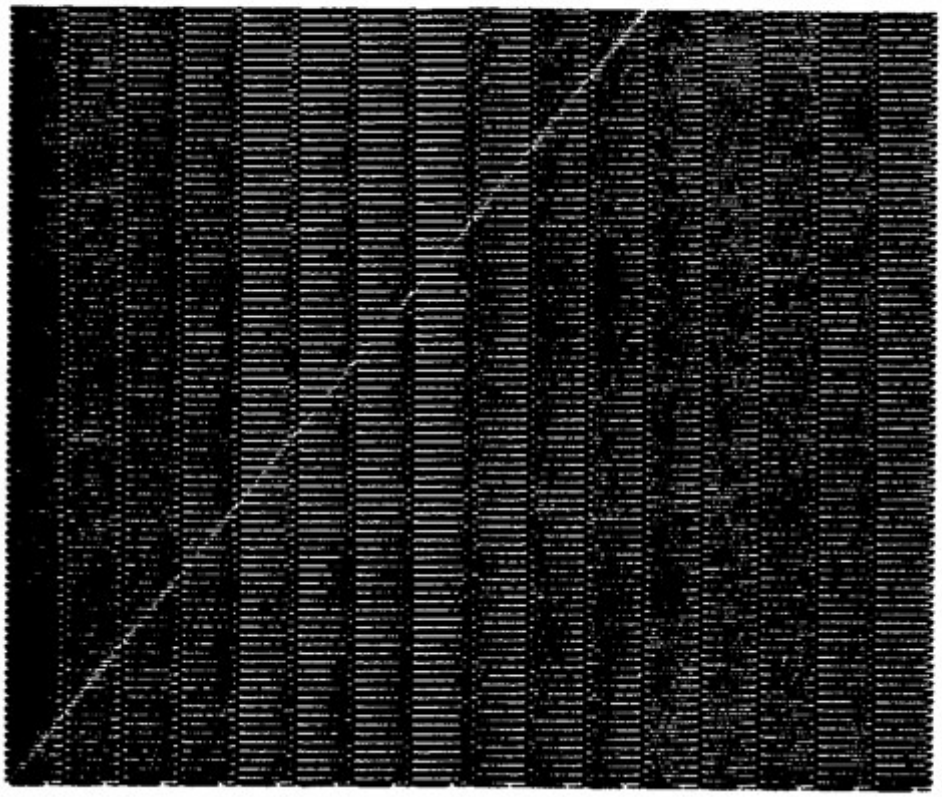


Fig. 9-6. The colorblind printer.

LISTing 9-6

```
10 PLOT 0,0: DRAW 25,25,1
20 DRAW 25,25,-2
30 DRAW 25,25,PI
40 DRAW 25,25,-1.5*PI
50 DRAW 100,-100,-2.5
60 DRAW -200,0
```

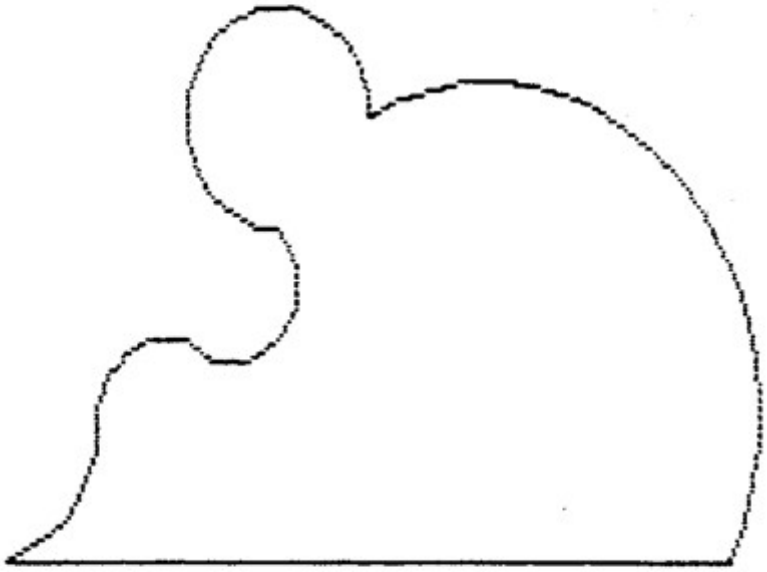
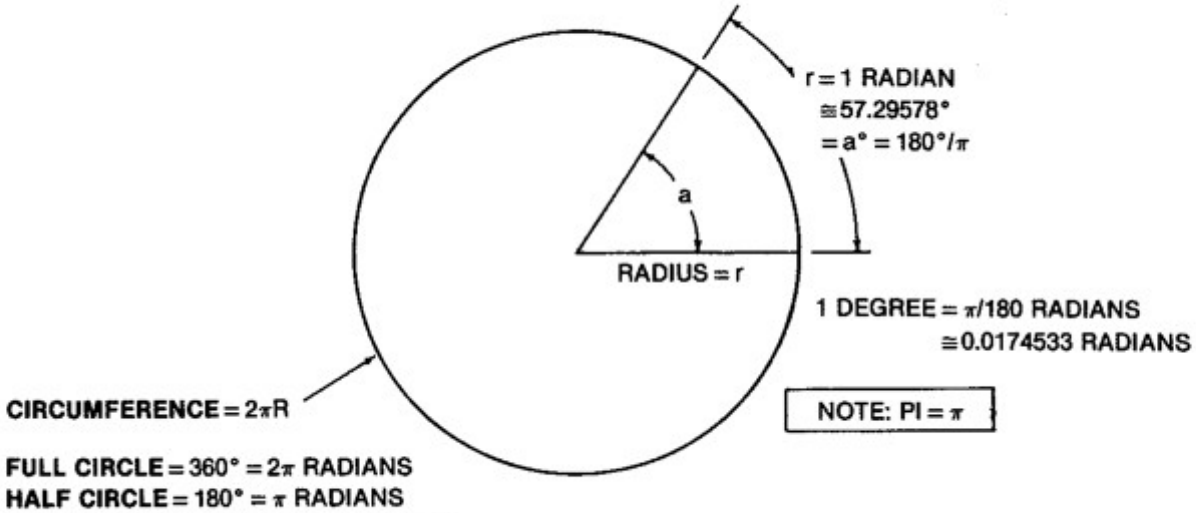


Fig. 9-7. It's a DRAW!

ment, you can DRAW a curve. Just to prevent the present program from confusing you, and to avoid all the color "surprises," use NEW to wipe out the existing program and then type and ENTER LISTing 9-6, which DRAWs Fig. 9-7.

Line 10 begins with a PLOT location, required to tell DRAW where to start. The DRAW statement says, in effect, draw a line from the PLOT starting point (lower left corner of the screen) and move 25 pixels to the right, and 25 up. However, the comma and the 1 that follow the regular "straight line" DRAW command make the line go through a left turn (counterclockwise) of one radian. See Fig. 9-8.



***TO CONVERT DEGREES TO RADIANS:** MULTIPLY DEGREES BY 0.0174533
***TO CONVERT RADIANS TO DEGREES:** MULTIPLY RADIANS BY 57.29578

Fig. 9-8. Radian measurement.

A radian is merely an arc equal in length to the radius of a circle. Since a complete circle is 360 degrees, and is equal to a length of 2 times PI times the radius of the circle (2*PI*r), then it follows that a "radian" must be 360/2*PI, or 180/PI, which is about 57.3 degrees of arc. Why is all this necessary? Because the T/S 2068 (and most microcomputers) do their trigonometric calculations in radians. To convert a number from degrees to radians, multiply by PI/180 (which equals .0174533). To convert from radians to degrees, multiply by 180/PI (which equals 57.29578). When making these conversions, just remember that radians are much larger than degrees—almost 60 times larger—so check to see if your answer makes sense.

What all this boils down to is that the end of line 10 tells the computer to swing an arc of almost 60 degrees while going from the PLOT point to the end of the DRAW point. The computer obligingly draws a left-turning arc of about one-sixth of a circle.

Line 20 tells the computer to DRAW a line 25 pixel spaces to the right and 25 spaces up, this time going through a right-turning arc of two radians—about 120 degrees. Why is it turning right? Well, it seems the T/S 2068 is "left-handed" in this command, and the negative sign in front of the two makes it go the opposite way. Incidentally, you can also DRAW the line in the opposite direction and get the same arc without the negative sign. Think about that one!

Line 30 is about the same, except there is no negative sign (so the curve will swing left) and instead of a number we are using PI (the M key with the E-cursor). PI causes DRAW to make a semicircle (since 2*PI radians equal a complete circle).

Line 40 draws a 270 degree clockwise arc. Notice that this function, like most, allows calculations within the commands. Lines 50 and 60 merely complete this meaningless figure.

Fig. 9-9. Familiar game character.

Obviously, those who are artistic (or who spent much time with join-the-dots pictures as kids) could design any kind of drawings with PLOT and DRAW. However, if you're not careful, and the arc tries to go beyond the screen limits, the program stops with an error.

Fig. 9-9 and its LISTing 9-7 draw a familiar computer game character. How did I come up with the 5.3 in line 30? Trial and error!

GOING IN CIRCLES

The CIRCLE instruction is very convenient and easy to use. You simply program the starting pixel location and then the radius. For example, CIRCLE 75,100,50 will draw a circle with a center 75 pixels to the right of the lower left corner, 100 pixels up, and with a radius of 50 pixel spaces. You can use the various attributes (PAPER, INK, FLASH, INVERSE, OVER, and BRIGHT) in the statement, such as CIRCLE PAPER 5;75,100,50 to draw the same circle on cyan

LISTing 9-7

```
10 PLOT 127,87
20 DRAW 25,18
30 DRAW 0,-36,5.3
40 DRAW -25,18
50 PLOT 120,105
60 DRAW 10,0,-PI
70 DRAW -10,0
```

PAPER—but remember that it will change the PAPER color in every character space it passes through.

Actually, you can PLOT a circle by using trigonometry. If you let x1 and y1 equal the center of the circle, with a the angle, and r the radius, then the location of any point on the circle is:

```
x=r*SINa+x1
y=r*COSa+y1
```

In other words, if we use different values of a, we can PLOT a circle. Let's do that, for angles one degree apart, and with a radius of 50. The starting point will be x=127 and y=87 (center screen). Here's the program:

```
30 FOR a=0 TO 360
40 LET x=50*SIN (a*PI/180)
50 LET y=50*COS (a*PI/180)
60 PLOT x+127,y+87
70 NEXT a
```

Line 30 sets up a FOR-NEXT loop with line 70 to LET the variable a equal angles from 0 to 360 degrees. Line 40 establishes the variable x as 50 times the SINe of angle a (converted to radians by multiplying by PI/180). Line 50 does the same thing in establishing variable y as 50 times the COSine of a. Line 60 does the actual PLOTting, very slowly and deliberately, one point at a time, with the 127 and 87 establishing the center of the circle. It takes about 45 seconds for you to draw a circle this way, plus the programming time.

To see the power of the CIRCLE command, just type the following from the keyboard (no line number):

```
CIRCLE 127,87,50
```

This draws the same CIRCLE in just over a second. Of course, you can use this line in a program to substitute for all the lines above.

Squishing a Circle

To "flatten" this circle—in other words, to make an ellipse—just change the "50" multipliers in lines 40 and 50. Make the multiplier 70 in line 40 and 35 in line 50. This "stretches" the figure along the x-axis (width), and shortens it along the y-axis (height).

The Big Doughnut

Now we'll use CIRCLE to draw a CIRCLE of CIRCLEs! See Fig. 9-10 and make these changes to the above program:

Add to line 30: STEP 4
Change the number 70 in line 40 to number 50
Change the number 35 in line 50 to number 50
Change line 60 to: 60 CIRCLE x+127,y+87,30

STEP 4 was added to line 30 so that the drawing time is faster—and the resulting figure actually looks better. Lines 40 and 50 are changed to get us back into circle coordinates instead of an ellipse. Also, the CIRCLE statement is used in line 60 in place of PLOT. This statement says, in effect, "Draw a CIRCLE with its center at x+127, y+87, and with a radius of 30." Since x and y are transcribing a circle as the angle increases four degrees at each STEP, the result is an annulus or "doughnut" shape. Fig. 9-10 is somewhat elongated by the manner in which the T/S 2040 Printer proportions its printing.

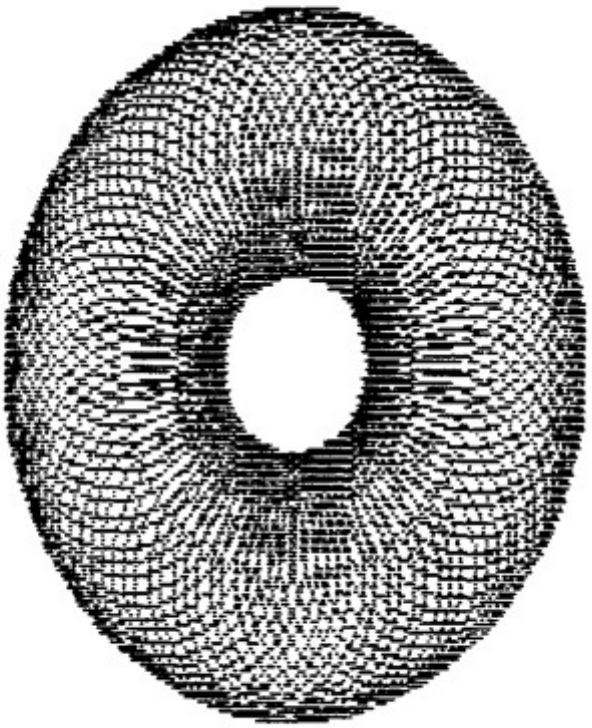


Fig. 9-10. "Big doughnut."

A Squished Doughnut

To make an elliptical doughnut, just change the "50" in lines 40 and 50 to numbers 70 and 35, respectively, as done previously to make an ellipse. RUN and you'll see the difference. Now hit BREAK and go get a cup of coffee and a real doughnut!

A SINE OF SPRING

Having the computer draw any mathematical curve is just a matter of placing the formula in a program. The formula for a SINe curve is:

$$y = \text{SIN } a$$

The variable y is the vertical displacement as the curve moves through the angle a, with the angle a plotted along the horizontal axis. If we adjust the horizontal length to fit on the display, we can use the following short program to draw a SINe curve. Use NEW to clear memory first.

```
10 PLOT 0,87: DRAW 255,0
20 FOR a=0 TO 360
30 LET y=50*SIN (a*PI/180)
40 LET x=a/2
50 PLOT x+35,y+87
60 NEXT a
```

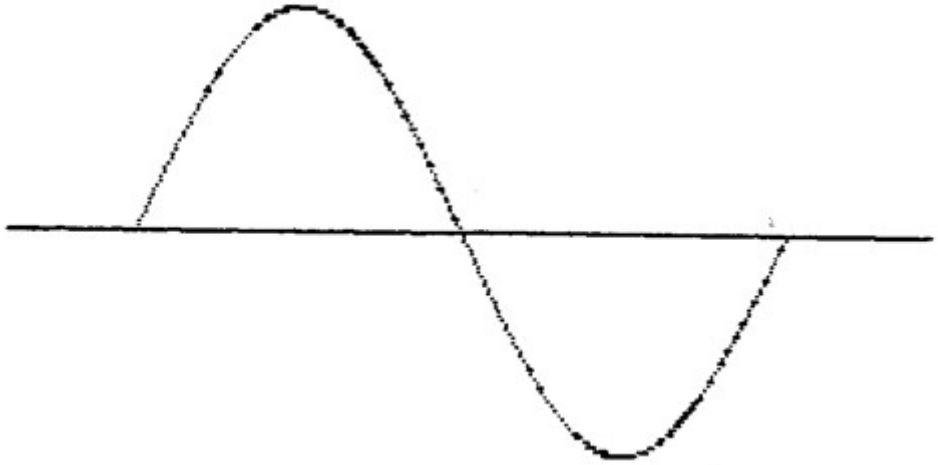


Fig. 9-11. SINe Wave.

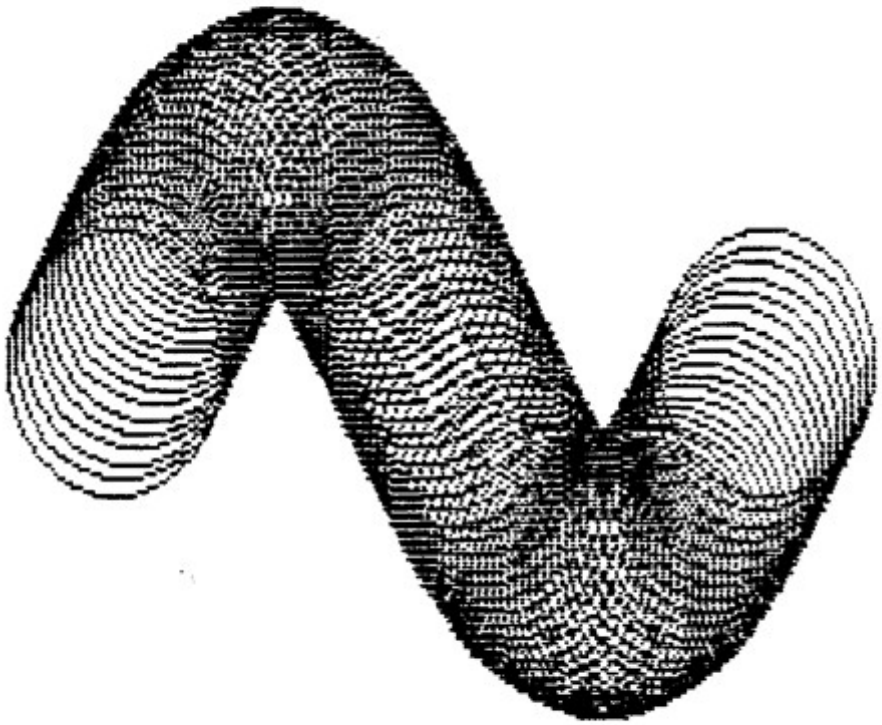


Fig. 9-12. "A SINe of spring."

This is really very similar to the circle/ellipse program. Line 10 DRAWS a full-width horizontal axis line halfway up the screen. Line 20 sets up a loop to advance degrees by one each time. Line 30 establishes the maximum height of the SINe wave as 50. Line 40 sets the horizontal maximum at 360/2, or 180, well within the 256 horizontal pixel spaces available. Line 50 PLOTs the graph coordinates, with the 35 and 87 to provide approximate centering on the screen. Line 60 completes the loop. When you RUN this program you'll get Fig. 9-11. But how would you like to get Fig. 9-12?

With a few minor changes, the SINe Wave program will produce

LISTing 9-8

```
  10 REM * Flag Program *
  20 REM * (C) Copyright Fred Bl
echman 1983 *
 100 BORDER 4: PAPER 5: FOR x=1
TO 22: PRINT "
                ": NEXT x
 110 FOR x=3 TO 13 STEP 2
 120 PAPER 7: INK 2: PRINT AT x,
4;"                              "
 130 PRINT AT x+1,4;"
                "
 140 NEXT x
 150 PRINT AT 15,4;"              
                "
 200 FOR x=3 TO 9
 210 PAPER 1: PRINT AT x,4;"
      "
 230 NEXT x
 300 INK 7: FOR z=1 TO 200
 310 READ x,y
 320 IF x=999 THEN GO TO 400
 330 PLOT x,y
 340 PLOT x-1,y+1: PLOT x+1,y+1:
 PLOT x-1,y-1: PLOT x+1,y-1
 350 NEXT z
 400 PAPER 5: INK 1: PRINT AT 18
,4;"America, The Beautiful!"
 900 DATA 39,147,52,147,65,147,7
8,147,91,147,104,147
 910 DATA 45,141,58,141,71,141,8
4,141,97,141
 920 DATA 39,135,52,135,65,135,7
8,135,91,135,104,135
 930 DATA 45,129,58,129,71,129,8
4,129,97,129
 940 DATA 39,123,52,123,65,123,7
8,123,91,123,104,123
 950 DATA 45,117,58,117,71,117,8
4,117,97,117
 960 DATA 39,111,52,111,65,111,7
8,111,91,111,104,111
 970 DATA 45,105,58,105,71,105,8
4,105,97,105
 980 DATA 39,99,52,99,65,99,78,9
9,91,99,104,99
 999 DATA 999,999
1000 PAPER 7: INK 0
```

Delete line 10
Add STEP 4 to line 20

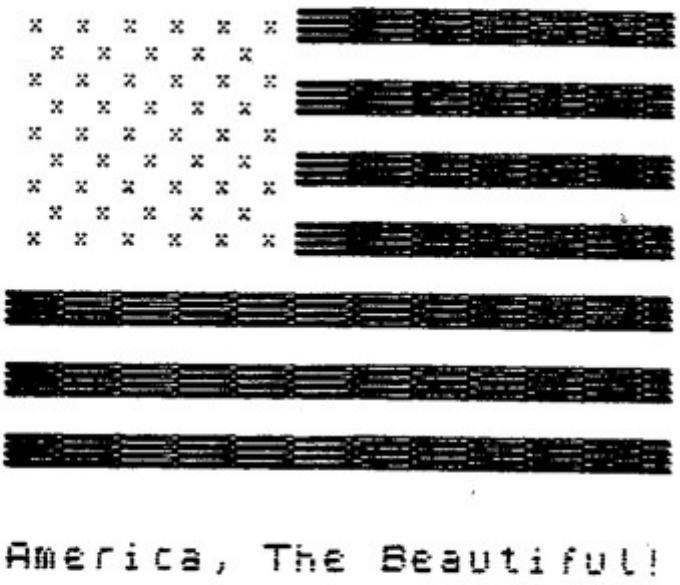


Fig. 9-13. The stars and stripes.

That's enough math for now. Later on, in Chapter 13, "Bio-Graphs and You," we'll run into SINe waves again. Then, in Chapter 17, "Long-Distance Navigator," we'll get into spherical trigonometry and actually make a screen display of the globe with a typical spherical triangle using geographic coordinates. (I'm expecting to use this program to plan my flight out of this infernal jungle. First I need an airplane.)

OLD GLORY

There isn't too much enthusiasm for the U.S. flag out here in this miserable, hot, dry jungle (when it's not an intolerable, muddy, humid swamp!) so I designed my own to display in glorious color on my TV set. LISTing 9-8 is the program, and Fig. 9-13 is the noncolor printout.

This display is really a knockout in color, with the red and white stripes and the stars on a blue field, all against a sky-blue (cyan) background surrounded by a green border. Makes me want to stand up and salute! Not only that, but in Chapter 16, "The Music Maker," there is a music program you can add to it which plays "America, The Beautiful."

The Stripes

Line 100 sets up the green border, then draws a cyan field of PAPER with a 22-line loop that prints blanks across the whole screen. Line 110 sets up a loop to print the stripes on alternating

lines. Line 120 resets the PAPER to white, and the INK red, for the statements that follow, then PRINTs a red stripe. Line 130 PRINTs a white stripe (blanks, or PAPER) on the next line. Line 140, with line 110, continues the loop, counting by two, for the rest of the stripes. Simple and fast.

Lines 200–230 PRINT in the blue star field with a short FOR-NEXT loop and blue PAPER, using the same technique as that used for the white stripes. So now we have the flag with no stars.

Getting the "stars" properly placed on this field took some doing. Lines 300–350, together with DATA statements 900–999, do the job. Line 300 sets the INK color to white, and starts a counting loop. Each "star" is actually five pixels. The center pixel in each star is placed in DATA, where it is READ by line 310. For example, the first star location is at location 39 and 147. That means 39 pixels over from the left and 147 up from the bottom. These values are assigned to x and y by line 310. Now line 330 PLOTs that point, and line 340 PLOTs four points around it, in the form of a small x. Line 350 goes back for another location. Line 320 checks to see if the end of DATA has been reached, and line 400 PRINTs "America, The Beautiful" under the flag.

How did I figure out where to PLOT the stars? I used graph paper and simply proportioned their locations to fit the space. It took time and patience—and trial and error—but that's programming, folks. Was it worth my trouble? (Say "YES"!) Now, if I could only figure out a way to have the flag wave.

QUESTIONS

1. How many block symbols are on the keyboard keys? Is the black part of the symbol PAPER or INK? How do you get the "reverse" of these symbols?
2. Change the "Vicious Doggie" graphic to a "Vicious Kitty."
3. Design your own special graphic character (say, a spider) to be used with the keyboard letter s.
4. How many pixels across in Display Mode 1? How many usable pixels on the entire screen (not including reserved lines)?
5. How many shapes can you program into a 64-pixel space?
6. What happens when a red PLOT point enters a space with green INK? Yellow INK? Red INK? White PAPER? Red PAPER?
7. How many colors does the T/S 2040 Printer produce? How does it distinguish between different INK and PAPER colors?
8. How many radians in a circle? How many degrees in a circle? How many degrees in a radian?
9. Does the T/S 2068 do trigonometric calculations in degrees or radians?

10
The Sounds of Music?

```
Program: keyboard
  10 REM * Keyboard *
  20 REM * (C) Copyright Fred Bl
echman 1983 *
 100 LET a$=INKEY$
 110 FOR f=1 TO 8
 120 READ f$,t
 130 IF f$=a$ THEN BEEP .2,t
 140 NEXT f: RESTORE : GO TO 100
 200 DATA "q",0,"w",2,"e",4,"r",
5,"t",7,"y",9,"u",11,"i",12

0 OK, 0:1
```

Play music on the keyboard.

INTRODUCTION

Computer-generated sounds vary from simple tones to complex synthesized music which simulates various instruments or takes on its own unique character. The T/S 2068 Series computers have not been designed as synthesizers but have a broad capability in the area of sound generation using the BEEP and SOUND commands.

Actually, it's "sound" that has brought me to this steaming, sweaty jungle. My mission, which I chose to accept, is to determine if, in truth, the signals transmitted by jungle drums are really in binary coded decimal format. So far, I've discovered that one of the lesser

known tribes here is communicating in modified bisynchronous ASCII . . . but that's for another book.

This book will not cover the SOUND command. Since it is complex to explain and use, it will be covered in the *Timex 2068 Intermediate/ Advanced Guide,* the second book in this series. However, I will cover the BEEP command, with its 130 semi-tones, and together we'll use simple notation and programming to produce the musical scale. Chapter 16 of this book provides details on programming simple tunes. Just don't expect the T/S 2068 to substitute for your stereo music system!

MUSIC AND THE T/S 2068

I am familiar with many subjects and can do many things—but writing or playing music is not one of them. (I have trouble playing the radio!) However, lest you feel that this "limitation" will prevent me from describing how to play melodies on the T/S 2068, let me point out that I am forced to keep my explanations simple and understandable. If you're a musician you may come away from this chapter screaming at the liberties I take in my description and notation of music, but the information necessary to make "music" on your 2068 will be here in rudimentary form, nevertheless. On the other hand, if you are a musical novice, you should find this information easy to follow and use—and it may lead you into becoming a Timex Tschaikowsky.

The BEEP Command/Statement

Very simply stated, the BEEP makes a sound for a specific duration at a designated frequency. (You may prefer the term "pitch" to "frequency." For us nonmusical types, frequency or "tone" seems more meaningful. We'll leave the "pitch" to a salesman or baseball player!) The syntax is:

```
BEEP duration, frequency
```

BEEP is obtained by a sequence of commands. Firstly, press the SYMBL SHIFT key and either CAPS SHIFT key at the same time. This will give you an E-cursor on your display. Now, holding the SYMBL SHIFT key down, press the Z key (which has BEEP printed below it) and BEEP should appear on your screen.

BEEP alone does nothing. You need to specify a frequency and duration, with the duration first. This seems backwards (and is, indeed, the reverse of some other computers), but you can't argue with Mother Timex. Just for illustration, next press 1, then a comma (by pressing the SYMBL SHIFT and N key at the same time). Now press 0. On the screen you should have:

```
BEEP 1,0
```

When you press ENTER the speaker will produce a "mellow" note for about one second. "Mellow" in this case happens to be "Middle-C," or about 262 cycles per second (or 262 Hz for you technical types). The frequency designator (we used 0 in this example) can be any number from −69 to +69. The duration (we used one second here) can be any number (including decimals) up to (but not including) 10.5.

A Piano Keyboard

Fig. 10-1 shows a typical piano keyboard, with notations based on the "American Equal Temperament Musical Scale." In this musical scale (there are others used in different countries or for special types of music) the standard frequency of A is 440 Hz, and the frequency of each key, moving from left to right, is one "semi-tone" higher than the preceding key (including the black keys). What's a "semi-tone"? Well, mathematically, that's exactly 2 raised to the $\frac{1}{12}$ power for this system of frequencies. For you and me, it's the next key on the keyboard.

For your convenience, Fig. 10-1 shows 24 keys (that's two "octaves," since there are twelve keys per octave) with the common music designations, the frequency, and the T/S 2068 BEEP frequency designator.

You'll notice that some of the keys are marked with a "#." This is the musical symbol for "sharp," and it simply means the next key to the right (one semi-tone upscale). For example, C-sharp is the first key to the right of the C key. Now, just to confuse us all, musicians also refer to "flat," and use a symbol that resembles a "b." This merely means the next key to the left (one semi-tone downscale). So a "D-flat" is the same as a "C-sharp." This should put it all together for you musical types. For the rest of you, I hope by now you are not key-bored (ouch!).

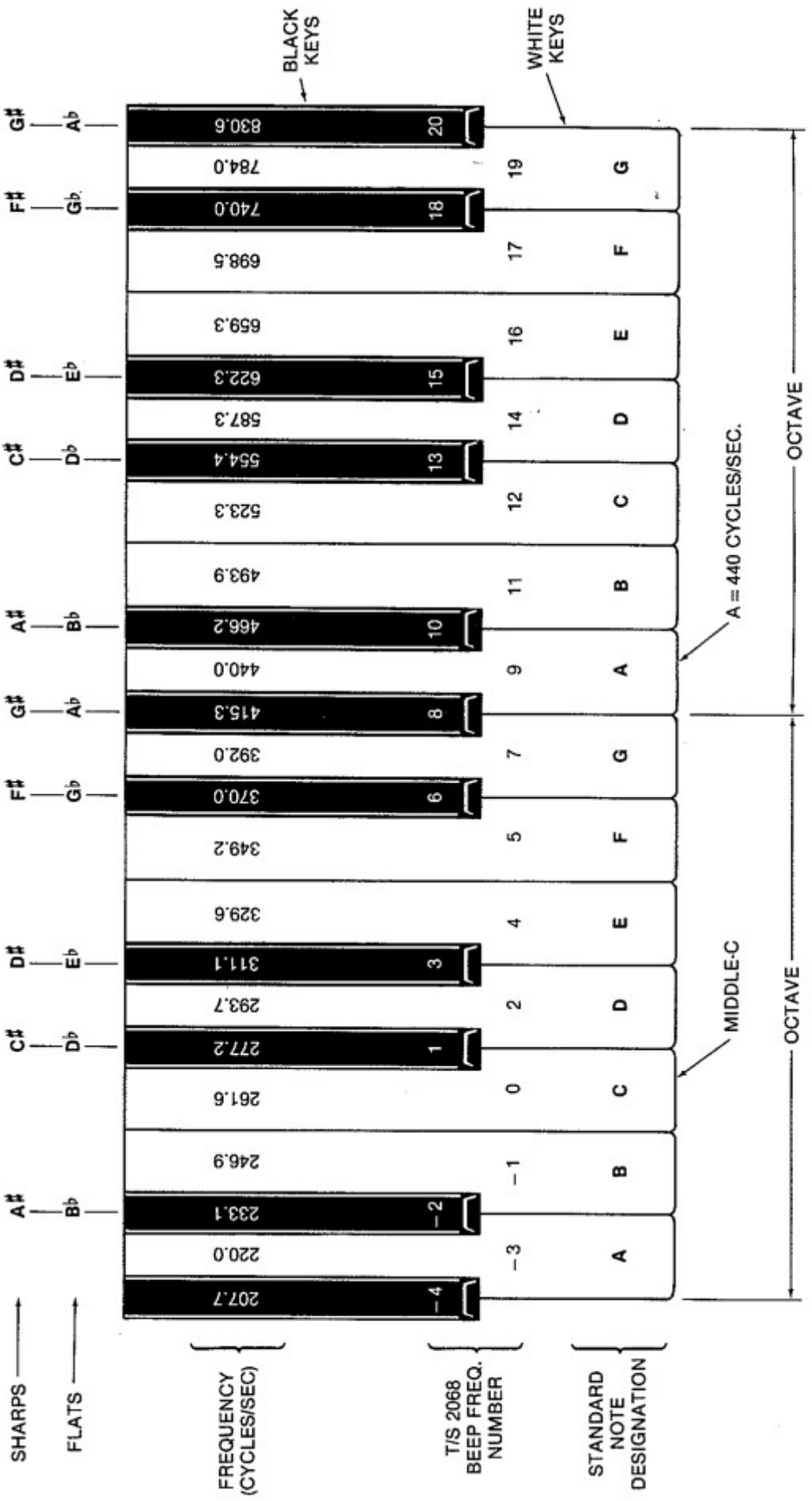


Fig. 10-1. Part of piano keyboard. (American Equal Temperament Musical Scale)

Musical Notation

Well, that's all well and good. Now we know what a keyboard looks like, and how to make individual BEEPS. But what about making music? Music is really just a specified sequence of tones, each played for defined durations. (I'll bet you never heard music defined *that* way before! Neither have I. But that's what it seems to me to be.) In order to describe music on paper, an elaborate scheme of special symbols and rules of notation has evolved. These are no doubt necessary for meaningful music, but since the T/S 2068 was never meant for Carnegie Hall (and because regular musical notation mystifies me) I have devised my own "hybrid" system that you should be able to follow easily. This notation is rudimentary—and I leave it for you music buffs to embellish it with your own features—but it is satisfactory for our purposes here.

Fig. 10-2 shows the system of musical notation I'm using in this volume. Different note symbols are used to designate duration. Their placement on the musical "staff" (the five horizontal lines) is conventional, and the "tails" all extend upward to keep things simple. No notes are connected together. This is simplistic, an equivalent to a kindergarten music primer, but it's easy to follow for nonmusicians.

TUNE-ING UP

OK, let's program the computer to play some music. All we have to do is let each key represent a note on the piano, and then find a way to instruct the computer to play the specified note when a key is pressed. Actually, if we are willing to accept some inherent limitations, a relatively short program can put the musical scale on part of one row of the keyboard.

Musical Keyboard

The program to do this is shown in LISTing 10-1. Line 100 sets up an INKEY$ loop. That is, it sets a$ equal to the computer internal value of whatever key is pressed—or nothing if no key is pressed. Then lines 110–140 set up a FOR-NEXT loop to READ the DATA (line 200) two items at a time. The first of each two DATA items establishes "f$" and the second sets the value of variable t.

Meanwhile, each time through the FOR-NEXT loop, line 130 checks to see if the new value of f$ is the same as the key being

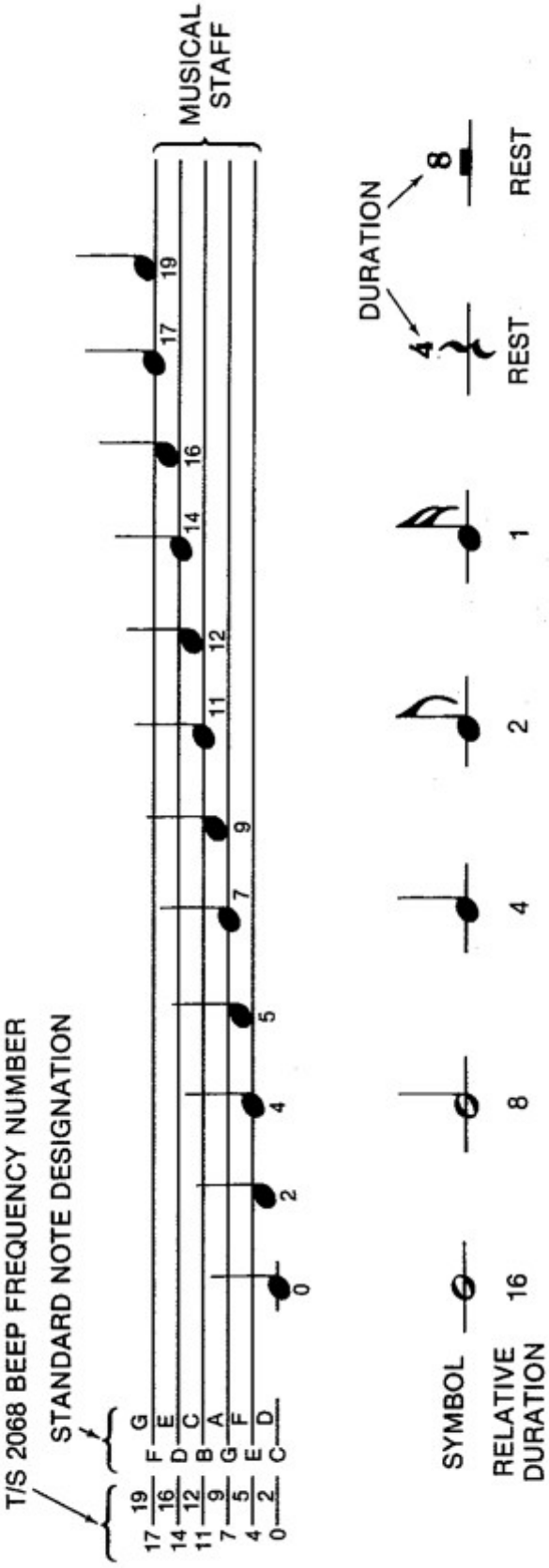


Fig. 10-2. Musical notation (as used in this book).

LISTing 10-1

```
  10 REM * Keyboard *
  20 REM * (C) Copyright Fred Bl
echman 1983 *
 100 LET a$=INKEY$
 110 FOR f=1 TO 8
 120 READ f$,t
 130 IF f$=a$ THEN BEEP .2,t
 140 NEXT f: RESTORE : GO TO 100
 200 DATA "q",0,"w",2,"e",4,"r",
5,"t",7,"y",9,"u",11,"i",12
```

pressed. If it is, it BEEPs for a duration of .2 second, using the value just set for t (0, 2, 4, 5, etc.) to establish the T/S 2068 BEEP frequency (from middle-C to the next C upscale). If, however, no key is pressed, or some key other than q,w,e,r,t,y,u, or i is pressed, the program continues to line 140. Here another "look" is taken until a total of eight "tries" is completed by NEXT f.

The RESTORE statement sets the DATA pointer back to the beginning of line 200, and then the program goes back to line 100 to look some more. To exit this program, press BREAK.

The simplicity of this program (much easier to use than to explain) results in some shortcomings. Only keys q–i, along the second row of your keyboard, create sounds. Also, each sound is of a specified duration. This means when you play a tune with notes that have different durations, you have to use your imagination and hesitate between keystrokes. Also, this program won't "play" faster than the duration specified in line 130 (.2 second per note). You can change that number, of course, to .1 or .3 to hear the difference. Furthermore, it takes just a smidgen longer for the program to detect when the i key is pressed as compared to the q key, since the DATA list must be read further to find "i"—more loop time is required. Therefore, as you can see, this program is not designed for playing "The Sabre Dance."

Just to give you some keyboard practice (and to chase away your dog, cat and probably the rest of your family) Fig. 10-3 gives you the "music" for "Home, Sweet Home." Just press the keys in the sequence shown (after RUN to start the program) and do the best you can on the timing. Have heart, however, because in Chapter 16 we'll program the T/S 2068 to play the tune, timing and all.

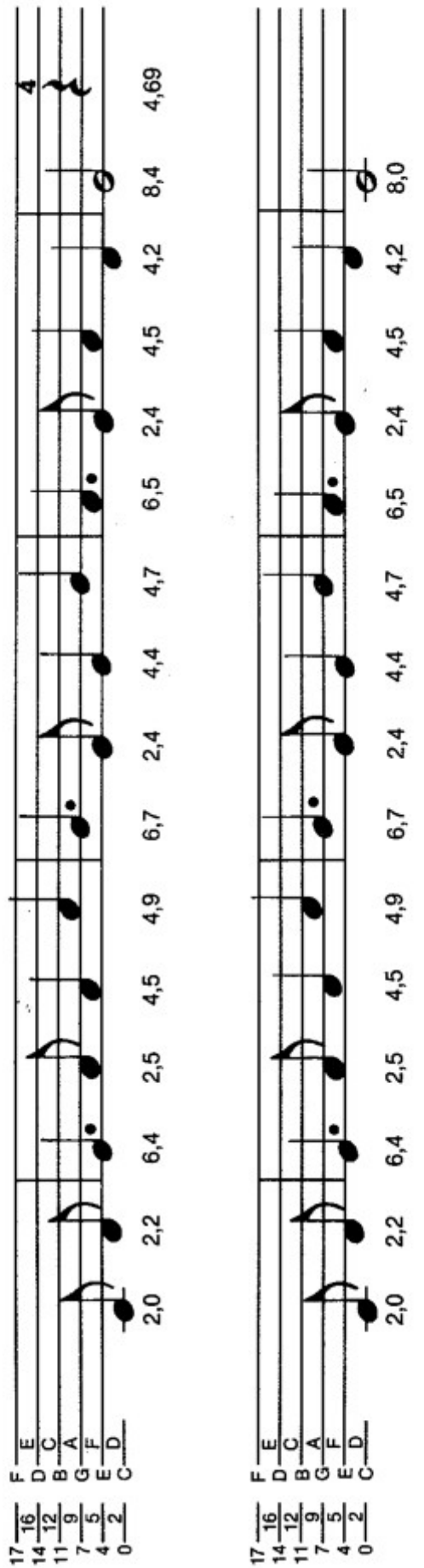


Fig. 10-3. "Home, Sweet, Home."

QUESTIONS

1. What is the "syntax" of the BEEP command?
2. Approximately how many seconds is a duration of .5?
3. What musical note is produced by BEEP 2,1? For how long?
4. What is the maximum duration number that can be used with the BEEP command?
5. What is the maximum frequency number that can be used with the BEEP command? Minimum number?
6. How many semi-tones in an octave?
7. Why are D-sharp and E-flat the same frequency?
8. How many piano keys in an octave?
9. What is the standard mid-scale frequency for A?

PART 2
SOME PRACTICAL PROGRAMS EXPLAINED

SECTION A
HOME AND PERSONAL USE PROGRAMMING

Learning to write effective programs can take literally hundreds of hours—but learning to type and ENTER programs takes very little time, and you learn as you go. The programs in the rest of this book are written, for the most part, to do something practical—or, at least, to illustrate some useful programming techniques.

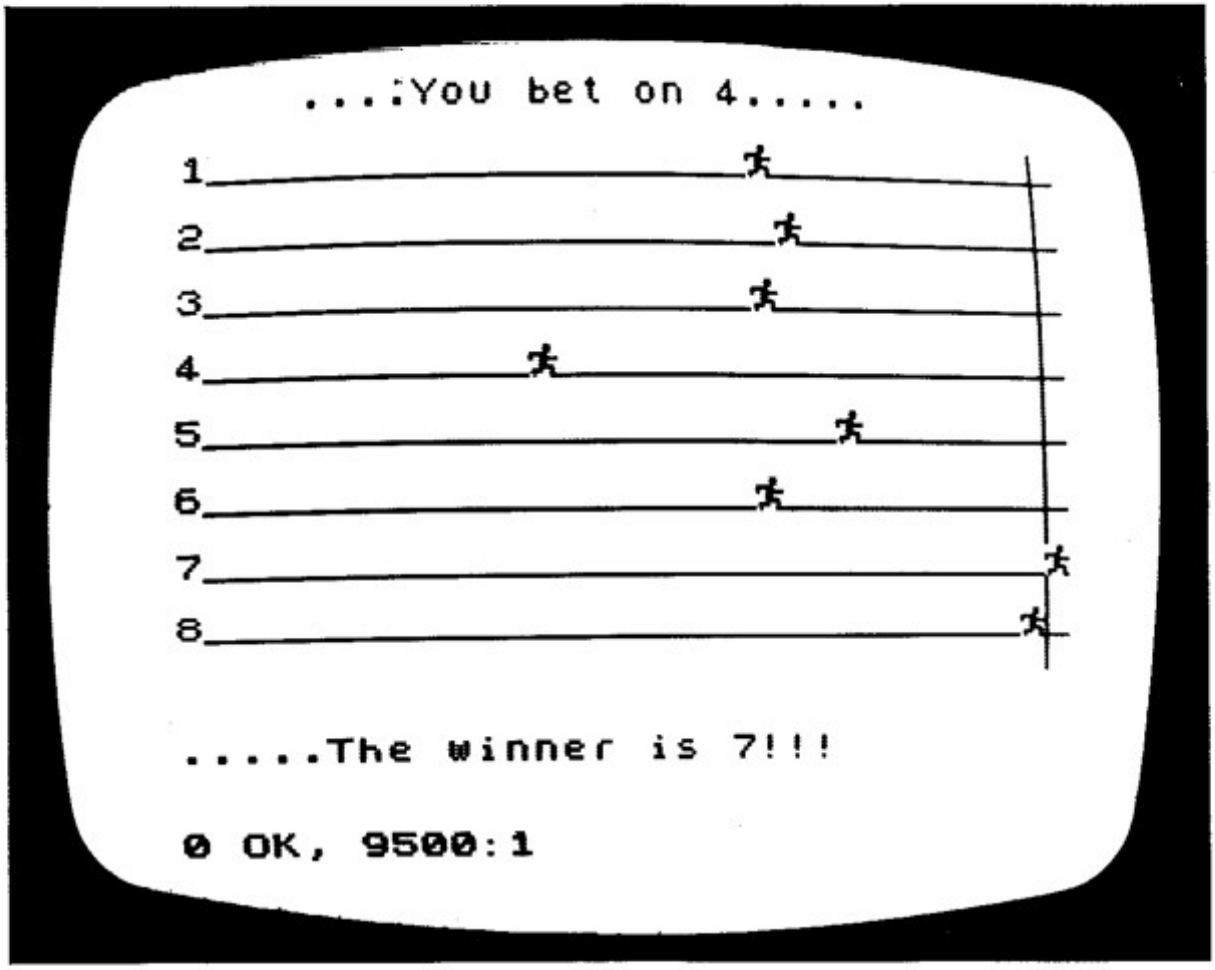
Chapter 11 lets you have a little fun with an animated footrace, using specially designed figures. You can root for a winner, but no betting allowed!

Chapter 12 can save you a bundle on long-distance telephone calls, if you just remember to use it. It requires no modification or connection to your telephone.

Chapter 13 illustrates some of the graphic capabilities of the T/S 2068 and will tell you how you feel, in case you don't already know!

Chapter 14 helps Timex sell rolls and rolls of printer paper. . . . Have fun!

11
On Your Mark!



You lose!

INTRODUCTION

It's time to have some fun with your newfound knowledge by putting some graphics together with BASIC programming and having an eight-man footrace. Just for good measure, some color and sound are added. You even get to pick your runner!

PROGRAM DESCRIPTION

When you RUN this program, an eight-lane track is displayed with each lane numbered, and a runner stands at the beginning of each

lane. At the bottom of the screen you're asked to choose your winner. When you press ENTER, the race begins. The screen border color changes and the computer "beeps" a different tone with each movement. Finally, one of the runners crosses the finish line and the winner is announced on the screen.

DISCUSSION

This program illustrates several programming techniques, including the use of special GRAPHICS characters. It provides a little harmless fun and can be expanded to include wagering—for entertainment only!

SPECIAL GRAPHICS CHARACTERS

In Chapter 9, "Grappling With Graphics," we developed two special GRAPHICS characters which we'll use here. A "standing man" and a "running man." (Of course, these could be women, but we'll refer to them as men for convenience.)

The "standing man" is assigned to the "A" key in the GRAPHICS mode, and the "running man" is assigned the "M" key in the GRAPHICS mode. Line 10 of LISTing 11-1 READs and POKEs the DATA from the BINary codes in line 20 to program the proper INK dots in the GRAPHICS M memory locations to form the running character. If you don't follow how this is done, go back to Chapter 9 to refresh your memory.

Using FOR-NEXT With BIN

There is a difference here in programming technique. In Chapter 9 we used eight program lines to enter a single character. Here we

LISTing 11-1

```
  5 REM * On Your Mark! * © Fred
 Blechman 1983
  10 FOR x=0 TO 7: READ n: POKE
USR "M"+x,n: NEXT x
  20 DATA BIN 00001100,BIN 00001
100,BIN 11111000,BIN 10011111,BI
N 00011000,BIN 00011110,BIN 0111
0010,BIN 01000011
```

LISTing 11-1—cont.

```
USR "A"+x,n: NEXT x
  40 DATA BIN 00001100,BIN 00001
100,BIN 01111000,BIN 01011110,BI
N 01011010,BIN 00011000,BIN 0010
0100,BIN 00100100
 100 FOR x=2 TO 16 STEP 2
 110 PRINT AT x,0;x/2;"_A";"____
_____"
 120 NEXT x
 125 FOR y=31 TO 159: PLOT 248,y
: NEXT y
 130 PRINT AT 0,0;"     ";: INPUT
 "Who do you bet on??";b: PRINT
"....You bet on ";b;"....."
 135 GO TO 200
 200 RANDOMIZE : LET a=3: LET b=
3: LET c=3: LET d=3: LET e=3: LE
T f=3: LET g=3: LET h=3: LET i=3
 210 LET r=INT (RND*8+1)
 215 BEEP .05,r: BORDER r-1
 220 GO TO r*1000
1000 PRINT AT 2,a-1;"_M";
1020 LET a=a+1: IF a=32 THEN GO
TO 9500
1030 GO TO 210
2000 PRINT AT 4,b-1;"_M";
2020 LET b=b+1: IF b=32 THEN GO
TO 9050
2030 GO TO 210
3000 PRINT AT 6,c-1;"_M";
3020 LET c=c+1: IF c=32 THEN GO
TO 9050
3030 GO TO 210
4000 PRINT AT 8,d-1;"_M";
4020 LET d=d+1: IF d=32 THEN GO
TO 9050
4030 GO TO 210
5000 PRINT AT 10,e-1;"_M";
5020 LET e=e+1: IF e=32 THEN GO
TO 9050
5030 GO TO 210
6000 PRINT AT 12,f-1;"_M";
6020 LET f=f+1: IF f=32 THEN GO
TO 9050
6030 GO TO 210
7000 PRINT AT 14,g-1;"_M";
7020 LET g=g+1: IF g=32 THEN GO
TO 9500
7030 GO TO 210
8000 PRINT AT 16,h-1;"_M";
8020 LET h=h+1: IF h=32 THEN GO
TO 9500
8030 GO TO 210
9500 PRINT AT 20,0;".....The win
ner is ";r;"!!!"
```

use only two lines in a FOR-NEXT loop (see Chapter 5), with READ and DATA statements (see Chapter 7), to accomplish the same thing.

Lines 30 and 40 program the GRAPHICS A memory locations to form the standing character, using the same programming technique.

TYPING THE LISTING

LISTing 11-1 must be typed in carefully. Not only does it require the normal care in typing the correct characters, but this program uses special GRAPHICS characters that are not apparent in a LISTing until the program has been RUN. Once the program has RUN, these characters are POKEd into memory, and the GRAPHICS character is displayed in its own shape. Compare LISTing 11-2 with LISTing 11-1 and you'll see where the GRAPHICS characters are. Once the program has been RUN and these characters have been entered into memory, they remain until they are intentionally changed or the computer is turned off. Even NEW or CLEAR does not remove them.

The only requirement when typing in the GRAPHICS character is that you are in the GRAPHICS mode (CAPS SHIFT and 9 key give you a G-cursor) when you press the letter A or M key. You exit the GRAPHICS mode the same way as you enter it—CAPS SHIFT and 9 key.

LISTing 11-2

```
  5 REM * On Your Mark! * © Fred
 Blechman 1983
  10 FOR x=0 TO 7: READ n: POKE
USR "A"+x,n: NEXT x
  20 DATA BIN 00001100,BIN 00001
100,BIN 11111000,BIN 10011111,BI
N 00011000,BIN 00011110,BIN 0111
0010,BIN 01000011
  30 FOR x=0 TO 7: READ n: POKE
USR "M"+x,n: NEXT x
  40 DATA BIN 00001100,BIN 00001
100,BIN 01111000,BIN 01011110,BI
N 01011010,BIN 00011000,BIN 0010
0100,BIN 00100100
 100 FOR x=2 TO 16 STEP 2
 110 PRINT AT x,0;x/2;"_M";"____
                          "
 120 NEXT x
 125 FOR y=31 TO 159: PLOT 248,y
```

LISTing 11-2—cont.

```
: NEXT y
 130 PRINT AT 0,0;"     ";: INPUT
 "Who do you bet on??";b: PRINT
"....You bet on ";b;"....."
 135 GO TO 200
 200 RANDOMIZE : LET a=3: LET b=
3: LET c=3: LET d=3: LET e=3: LE
T f=3: LET g=3: LET h=3: LET i=3
 210 LET r=INT (RND*8+1)
 215 BEEP .05,r: BORDER r-1
 220 GO TO r*1000
1000 PRINT AT 2,a-1;"_";
1020 LET a=a+1: IF a=32 THEN GO
TO 9500
1030 GO TO 210
2000 PRINT AT 4,b-1;"_";
2020 LET b=b+1: IF b=32 THEN GO
TO 9050
2030 GO TO 210
3000 PRINT AT 6,c-1;"_";
3020 LET c=c+1: IF c=32 THEN GO
TO 9050
3030 GO TO 210
4000 PRINT AT 8,d-1;"_";
4020 LET d=d+1: IF d=32 THEN GO
TO 9050
4030 GO TO 210
5000 PRINT AT 10,e-1;"_";
5020 LET e=e+1: IF e=32 THEN GO
TO 9050
5030 GO TO 210
6000 PRINT AT 12,f-1;"_";
6020 LET f=f+1: IF f=32 THEN GO
TO 9050
6030 GO TO 210
7000 PRINT AT 14,g-1;"_";
7020 LET g=g+1: IF g=32 THEN GO
TO 9500
7030 GO TO 210
8000 PRINT AT 16,h-1;"_";
8020 LET h=h+1: IF h=32 THEN GO
TO 9500
8030 GO TO 210
9500 PRINT AT 20,0;".....The win
ner is ";r;"!!!"
```

PROGRAM ANATOMY

Figs. 11-1, 11-2, and 11-3 show what you'll see on your screen at the beginning of the race, during the race, and at the end of the race.

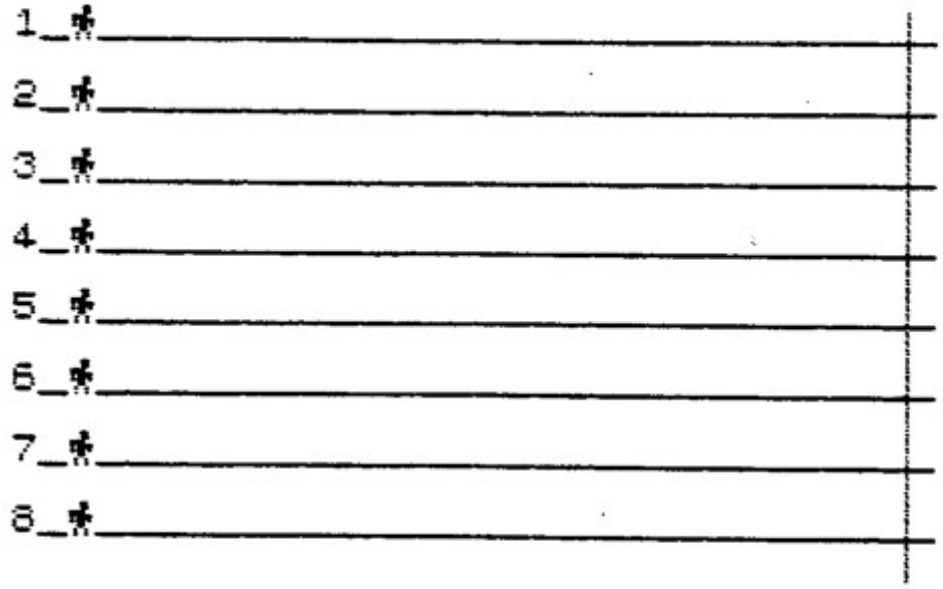


Fig. 11-1. "On your mark . . ."

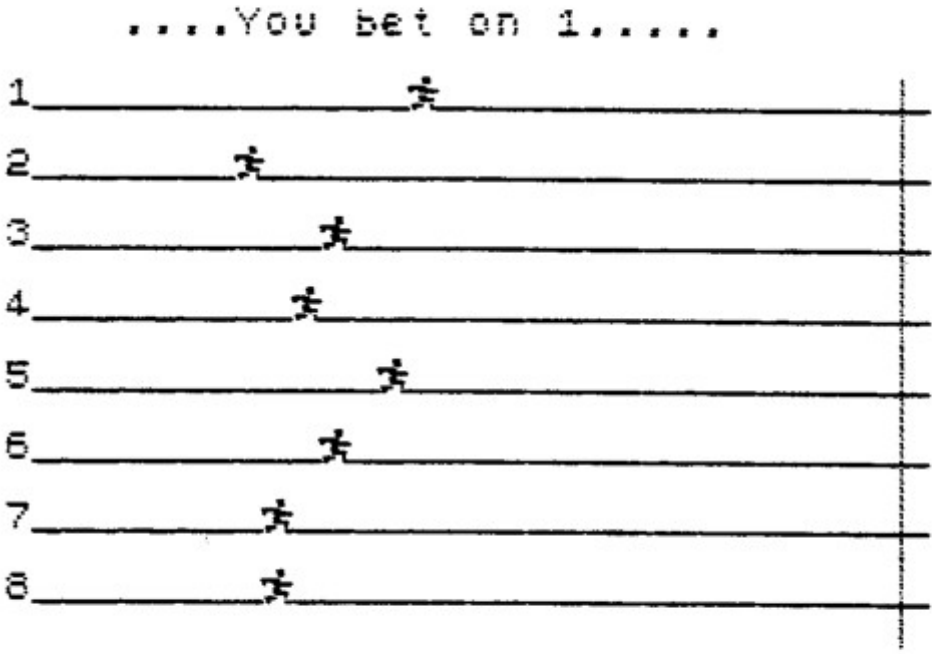


Fig. 11-2. "C'mon #1!"

Setting Up the Track

Lines 100–125 draw the track and finish line, number the lanes, and put a standing character at the start of each lane. That seems like a lot for only four lines, but it also illustrates the power of FOR-NEXT loops. Let's cover this in detail.

Line 100 sets up a loop to set the value of x from 2 TO 16 in STEPs of 2. That means x will have the value of 2 the first time through the loop, then 4 the next time, then 6, and so on—until it exceeds 16.

Line 110 tells the computer to PRINT the value of x divided by 2 AT screen row 2 (since x is equal to 2) in the first character space

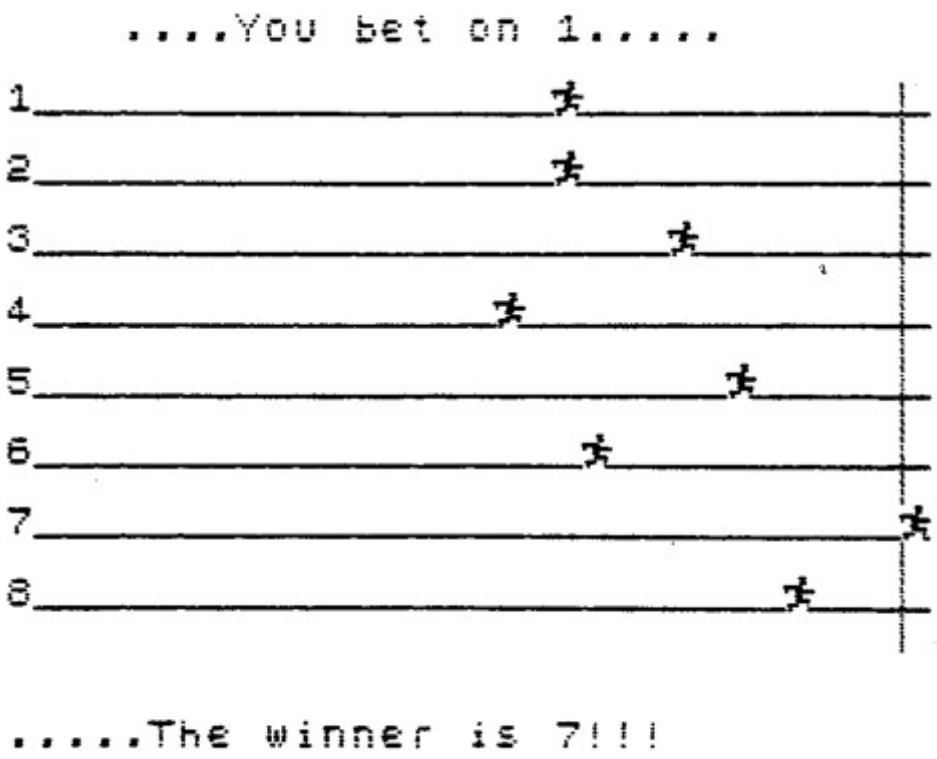


Fig. 11-3. "... and the winner is ..."

(0) of that row. What is the value of x divided by 2? Well, since x is equal to 2 this first time through the loop, then x = 2/2 or 1.

Next, because of the semicolon following "x/2" on line 110, the computer is told to continue PRINTing at the next character space, and to PRINT an underline character (SYMBL SHIFT 0) followed by what looks like a regular character A. Well, there's no way to tell from the LISTing (until the program has been RUN), but that A is a GRAPHICS A, which we've programmed to be a standing man.

The next semicolon on that line keeps the next PRINT instruction, 30 more underline characters, on the same line. The total number of characters now on that line is 32—a full line.

The Other Seven Lanes

Line 120 sends the computer back to line 100, and x becomes 4. Now line 110 prints lane 2 on screen row 4. As the program keeps shuttling back and forth between lines 100 and 120, with the value of x increasing by 2 each time, the other seven lanes are drawn in like fashion. When the value of x exceeds 16, the program moves on to line 125.

The Finish Line

Line 125 has three statements on one line, quite common with simple FOR-NEXT loops. It merely PLOTs a series of points on the

screen to draw a finish line at location 248 across. (Remember, the horizontal PLOT locations go from 0 to 255.) Actually, the DRAW command could have been used here for a faster finish line. Line 125 would then be:

```
125 PLOT 248,31:DRAW 0,128
```

Line 130 puts a few blank spaces at the top of the screen (you won't see them, but they're there—that's why they are called "blank spaces"). The INPUT question appears on the bottom of the screen and waits for your response. The program is written so that you can put in any number, then press ENTER. If you want to add "error trapping" to reject any number outside the range 1–8, be my guest. I've tried to keep this program simple, and since the program doesn't keep score of your wins or wagers, it doesn't really matter what number you enter.

Next the program prints your choice at the top of the screen (right after the blank spaces) and line 135 sends the program to line 200.

"Whoa," you say, "why do you need line 135? Won't the program continue to the next line, 200, anyhow?" The answer (blush, blush) is "yes." You caught my oversight. It's not unusual for a program to contain some harmless lines of programming that were used during development and were simply left there after their functions were no longer needed. That's what happened here—and probably in some other programs in this book. (And you thought I was perfect! Actually, I blame it on the hairy tarantula spiders that seem to love to cuddle up to the computer power supply. They are very distracting. Talk about computer bugs!)

Selecting a Runner

Line 200 starts out with RANDOMIZE, since we'll be using RaNDom numbers in the race, and we want all races to be different. RANDOMIZE simply causes the computer to move up the built-in pseudo-RaNDom number table to a point based on how long the computer has been turned on. This was discussed in Chapter 5.

Line 200 now sets variables a–i each to equal 3. Why 3? That's the next character space on each line past the standing character. The first space is the lane number, then a single-character underline, then the GRAPHICS character, occupying spaces 0–2. So 3 is the next space.

Notice that a, b, c, d, e, f, g, h, and i are nine variables. Why 9? It seems like we should only need 8, one for each lane. Once again (blush, blush, blush), you got me! Forget the variable i. It was left over from the nine lanes I used in the game. I later changed to eight lanes because there are only eight colors, as you'll see in a moment.

Line 210 picks a RaNDom number from 1 to 8 and assigns it to variable r. The RND statement by itself only chooses numbers greater than 0 and less than 1, so the multiplication by 8 provides numbers up to (but not including) 8. By adding 1, we assure that the minimum number is 1 and the maximum 8. The INT statement gives us a whole number, required by the statements that follow.

Line 215 BEEPs a tone based on the value of r, and the BORDER color becomes r-1. (The maximum BORDER color is 7.)

Now we come to line 220, which directs the computer to one of the subroutines that follow. Each lane has its own subroutine. The trick here is to use line numbers which are multiples of a given number for the subroutines so you can have "selected branching." In this case, the number 1000 is used as the multiplier.

For example, suppose the RaNDom number selected by line 210 is "5" (after the INT statement makes it a whole number). Line 220 then sends the program to line 5000, since that's 1000 times 5. Lines 5000–5030 move the running man one space to the right. Let's see how.

Moving and Checking

Line 5000 PRINTs AT line 10, which is the Number 5 lane. It PRINTs an underline character and a running man character, and starts this at a position one space to the left of the standing man, so that the previous figure is "erased" (actually PRINTed over) and a lane line put in his place. If this isn't done, you'll have a trail of special character figures on each line.

Line 5020 now adds one to the counter for that line, which is variable e in this case. It then checks to see IF variable e equals 32, the position just crossing the finish line. IF it has, then the program jumps to line 9500 and ends with the announcement of the winning runner.

IF e has not yet reached 32, the program jumps back to line 210 for the selection of another runner to advance one space. This keeps on going until one runner crosses the finish line.

MODIFICATIONS

There are all kinds of changes you can make to this program. You can add wagering, where the program tallies your winnings or losses before and after each race. You can make the runners move faster by advancing them two or three spaces at a time. You can make each lane and runner a different color, or make only the runners different colors. You can have the program play a short tune at the end of each race as it FLASHes the winner on the screen. If you really want to get creative, how about making several hurdles on each lane, with the runner jumping over each one as it is reached?

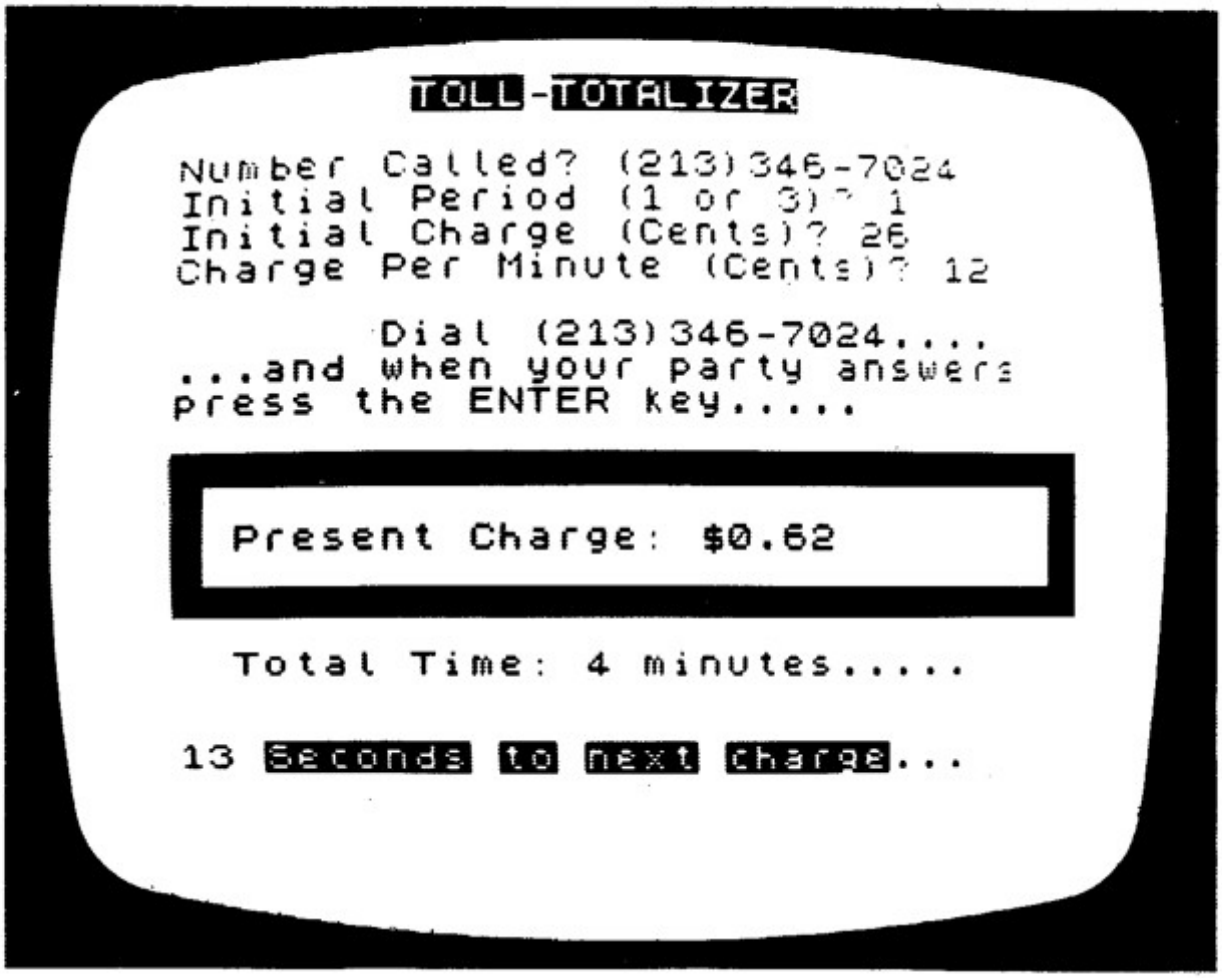
QUESTIONS

1. Explain how lines 10 and 20 create a special character.
2. How long will that special character remain in memory?
3. What do lines 100–120 accomplish?
4. Why is RANDOMIZE used in line 200?
5. Why is the INTeger statement needed in line 210?
6. Explain how line 220 functions.
7. In line 3000, why is "c-1" used instead of "c"?
8. What happens, in line 130, if you use 15 as the value for variable b?
9. What is the purpose of line 135? Is it necessary? Does it hurt anything?

VARIABLES

a Lane 1 runner position
b Lane 2 runner position; your "bet" selection
c Lane 3 runner position
d Lane 4 runner position
e Lane 5 runner position
f Lane 6 runner position
g Lane 7 runner position
h Lane 8 runner position
i not used (remnant of program development)
n BIN number from DATA
r value of RaNDom whole number
x loop counter; screen row number
y PLOT vertical location

12
Toll Totalizer



Keep track of toll charges while you talk.

INTRODUCTION

Telephone bills keep going up, up, and up! This program allows you to have the actual charges displayed on your computer as you're making a call. It tells you, by the second, how long till the next charge and how much more it will be. It even reminds you of the number you're calling! Next time you will be making a long-distance phone call, use the "Toll Totalizer" to keep those charges down.

PROGRAM DESCRIPTION

The program introduces itself and its purpose on the screen and asks if you want further information. If you respond with a "yes,"

you'll see two more screens of information showing how charges are calculated by the phone company, and the regular discount periods.

Next you're asked for the number you're calling, the initial period, the initial charge, and the charge for additional minutes. You are then instructed to press ENTER when the connection is completed. The program draws a box on the screen, puts the present charge in the box, and starts counting down by the second to the next charge. To catch your attention at the beginning of each minute, the BORDER color changes and the BEEP sounds.

DISCUSSION

You'll have to find the rates and discount periods that apply to your own situation from your own phone book and phone company, since these vary considerably—especially with the new phone company deregulation. This information is usually at the pages near the front of your phone directory. You might want to tear out those pages that apply to long-distance calling and keep them near your computer, so you have easy access to the various calling rates and discount periods.

While this program won't itself lower your phone bill, it greatly increases your awareness of the expense of a call as it's actually in progress. Also, since the phone company only bills by the minute, the second-counting feature allows you to perhaps cover an additional subject during the minute for which you're already being charged.

Two precautions should be mentioned. Some phone companies and services (such as MCI or SPRINT) may charge by the half-minute; you can check with them to find out. Also, the timing accuracy of this program will be affected by computer temperature, computer "clocking" speed, and ambient temperature. Line 850 of the program controls the counting speed. See Program Anatomy to adjust this speed.

PROGRAM ANATOMY

LISTing 12-1 shows the program and Fig. 12-1 through Fig. 12-4 show the four screen displays. Type the program in carefully, especially lines 700–740, which use SYMBL SHIFTed GRAPHICS 8 solid blocks and should LIST on your screen exactly as shown. Lines 700

LISTing 12-1

```
  10 REM * Toll Totalizer *
  20 REM * © Fred Blechman 1983
*
  30 REM * Line 850 counts secon
ds *
 100 PRINT AT 5,8;"[illegible]
"
 110 PRINT : PRINT "    This pro
gram is useful when"
 120 PRINT "making telephone tol
l calls. It"
 130 PRINT "will display your pr
esent"
 140 PRINT "charge, total minute
s charged,"
 150 PRINT "and seconds to next
charge."
 160 PRINT : PRINT "Additional i
nformation (y/n)?": INPUT r$
 170 IF r$(1)="y" THEN GO TO 200
 180 CLS : GO TO 500
 200 CLS : PRINT "     Your charg
es are based on"
 210 PRINT "several things:"
 220 PRINT "     (1)Initial perio
d"
 230 PRINT "     (2)Initial Charg
e"
 240 PRINT "     (3)Charge per mi
nute"
 250 PRINT "     (4)Time of day"
 260 PRINT : PRINT "     If you u
se an operator to"
 270 PRINT "assist you, the init
ial period"
 280 PRINT "is 3 minutes. Direct
 dialing"
 290 PRINT "is 1 minute...."
 300 PRINT : PRINT "     The char
ge per minute is"
 310 PRINT "based on the destina
tion called"
 320 PRINT "and the caller time-
of-day."
 330 PRINT "These rates are near
 the front"
 340 PRINT "of your local phone
directory."
 350 GO SUB 1000
 400 PRINT "  Most common discou
nt rates:"
 410 PRINT : PRINT "30% Discount
:"
 420 PRINT "    5PM-11PM Monday-F
riday"
```

LISTing 12-1—cont.

```
 425 PRINT "   5PM-11PM Sun. (Ou
t of State)"
 430 PRINT : PRINT "60% Discount
:"
 440 PRINT "   11PM-8AM Monday-F
riday"
 450 PRINT "   All Day Saturday
& Sunday"
 460 PRINT "   All Day certain h
olidays"
 470 PRINT : PRINT "(Verify with
 local directory,"
 480 PRINT " since this may chan
ge....)": PRINT
 490 GO SUB 1000
 500>PRINT "          TOLL-TOTALL
ER"
 510 PRINT : PRINT "Number Calle
d? ";: INPUT n$: PRINT n$
 520 PRINT "Initial Period (1 or
 3)? ";: INPUT a: PRINT a
 525 IF a<>1 AND a<>3 THEN GO TO
 520
 530 PRINT "Initial Charge (Cent
s)? ";: INPUT c: PRINT c
 540 PRINT "Charge Per Minute (C
ents)? ";: INPUT m: PRINT m
 600 PRINT : PRINT "       Dial
";n$;"...."
 610 PRINT "...and when your par
ty answers"
 620 PRINT "press the ENTER key.
....": INPUT z$
 700 PRINT : PRINT "
                    "
 710 PRINT "
           "
 720 PRINT "
           "
 730 PRINT "
           "
 740 PRINT "
                    "
 750 LET c=c/100: LET m=m/100: L
ET b=1
 800 REM * Counting seconds & mi
nutes *
 810 LET t=60: IF a=3 THEN LET t
=179
 820 PRINT AT 13,2;"Present Char
ge: $";c;"      "
 830 PRINT AT 17,2;"Total Time:
";a;" minutes....."
 840 PRINT AT 20,0;t;" Seconds t
o next call..."
```

LISTing 12-1—cont.

```
 845 BORDER b
 850 FOR x=1 TO 97: NEXT x
 860 LET t=t-1: IF t=-1 THEN GO
TO 900
 870 GO TO 820
 900 BEEP .1,0: LET t=59: LET a=
a+1: LET c=c+m
 910 LET b=b+1: IF b>7 THEN LET
b=0
 920 GO TO 820
1000 FLASH 1: PRINT : PRINT "...
.PRESS ENTER TO CONTINUE...": IN
PUT z$: FLASH 0: CLS : RETURN
```

```
    This program is useful when
making telephone toll calls. It
will display your present
charge, total minutes charged,
and seconds to next charge.

Additional information (y/n)?
```

Fig. 12-1. First screen.

```
    Your charges are based on
several things:
     (1)Initial period
     (2)Initial Charge
     (3)Charge per minute
     (4)Time of day

    If you use an operator to
assist you, the initial period
is 3 minutes. Direct dialing
is 1 minute....

    The charge per minute is
based on the destination called
and the caller time-of-day.
These rates are near the front
of your local phone directory.

....PRESS ENTER TO CONTINUE...
```

Fig. 12-2. Second screen.

and 740 each have 32 solid blocks. Lines 710, 720, and 730 each have one block, 30 spaces, and then another block. The INV.VIDEO (CAPS SHIFT 4) characters in lines 100, 500, and 840 are just for effect; they may be standard characters.

```
  Most common discount rates:

30% Discount:
    5PM-11PM Monday-Friday
    5PM-11PM Sun. (Out of State)

60% Discount:
    11PM-8AM Monday-Friday
    All Day Saturday & Sunday
    All Day certain holidays

(Verify with local directory,
 since this may change....)

....PRESS ENTER TO CONTINUE...
```

Fig. 12-3. Third screen.

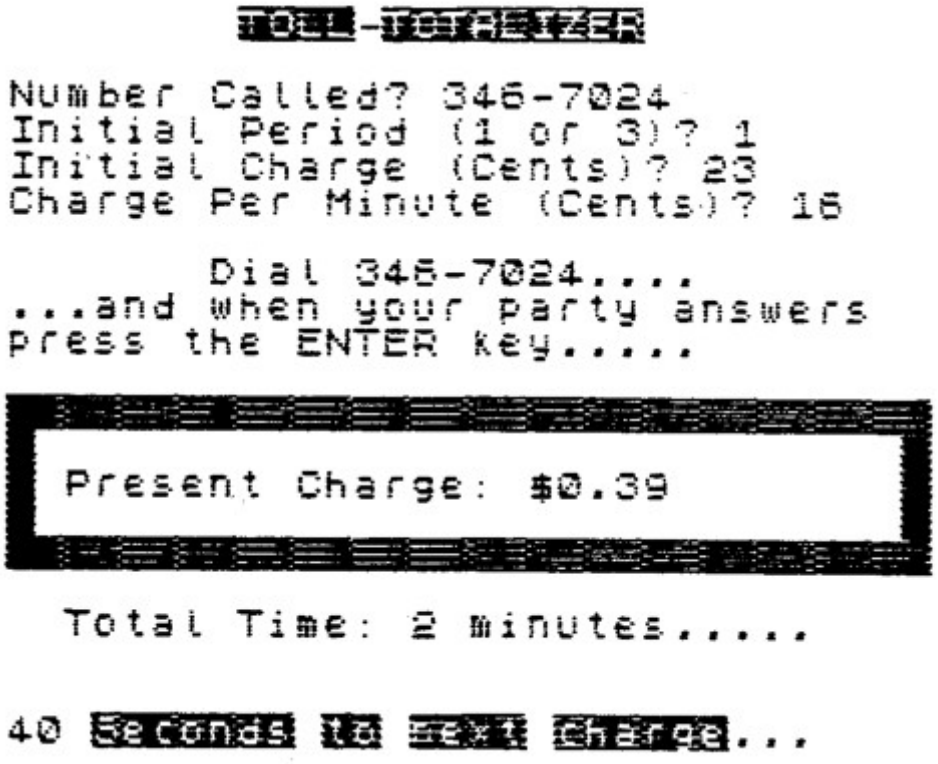


Fig. 12-4. RUNning program.

Program Introduction

Lines 10–30 identify the program and remind you that line 850 is the seconds counter. Lines 100–150 introduce the program on your screen, and line 160 allows you to bypass further instructions. If you type and ENTER any response that does not start with a lowercase letter y, the program proceeds to line 180, where the screen is cleared and the program goes to line 500.

Additional Instruction Screens

IF, however, line 170 sees that you ENTERed any word whose first character is a lowercase "y" (such as "yes," "yep," "yessir," "yeah," "you betcha'," etc.), THEN the program is sent to line 200, where the screen is cleared, more information is PRINTed (lines 200–340) and finally line 350 sends the program to a SUBroutine at line 1000.

Line 1000 has seven statements on one line. FLASH 1 tells the computer you want the following characters to FLASH (switch PAPER and INK colors about twice a second until FLASH 0 is encountered). The first PRINT simply skips a screen line. The next PRINT instruction FLASHes the prompt to press ENTER. The INPUT statement waits for you to press ENTER. When you do, the dummy string, z$, is made a "null" (nothing, not even a blank space), but this satisfies the INPUT prompt, and the program line continues. Now the FLASH 0 turns off the FLASH command, CLS clears the screen, and the program RETURNs to the line it came from, line 350 in this case.

Since there's nothing more on line 350, the program continues on to the next line, 400, and lines 400–480 put more information on the screen. Line 490 GOes to the SUBroutine at line 1000 for another "... PRESS ENTER TO CONTINUE ..." interrupt. When you press ENTER the program RETURNs to line 490. There's nothing more there, so it goes on to line 500.

Notice that whether or not you request the "Additional Information (y/n)?" queried in line 160, you still arrive at line 500.

User INPUTs

Line 510 asks you to type and ENTER the number you are calling. This is not used by the program except to PRINT the number on the screen for your reference. Notice the technique here of putting the question on the screen, then PRINTing the response after it. Normally, the INPUT response appears at the lower left of your screen and disappears when you press ENTER.

Line 520 asks you to ENTER the number of minutes for the initial time period—either 1 or 3 minutes, depending on whether it is an operator-assisted call. Line 525 checks to see that your INPUT was either a 1 or a 3. IF it was not, the program goes back to line 520 for another INPUT.

Lines 530 and 540 ask for the initial and per-minute charges,

which are INPUT in cents. In other words, 20 cents would be entered as 20, not .20.

Ready To Go!

Lines 600–620 repeat the number for you to dial and instruct you to press ENTER when the person at the other end of the line picks up the phone. When you press ENTER, lines 700–740 draw a large box across the full width of the screen, using simple PRINT statements.

Line 750 converts your initial and minute charges to decimal cents and sets variable b equal to 1.

Tempus Fugit

The actual time-counting and display is performed by lines 800–920. Line 810 sets variable t, time to next charge, to 60. Actually (I just realized, as I'm writing this!) this should be 59, since it counts down to zero—a total of 60 counts.

IF you indicated in line 520 that the initial time would be 3 minutes, the value of t is THEN changed to 179 for 180 counts to zero. (The number of seconds in three minutes is, of course, 180.)

Line 820 PRINTs the present charge inside the box, using a dollar sign and decimal cents. Then line 830 prints the total minutes on the second screen line below the box. On the third line below that, screen line 20, program line 840 PRINTs the seconds to the next charge.

Line 845 changes the BORDER to the color represented by the variable b (which was initialized at a value of 1 in line 750).

Line 850 is a short delay FOR-NEXT loop that counts to itself 97 times before moving on to the next program line. Why is the number 97 used? Simply from trial and error on my computer.

Actually, every T/S 2068 runs at a slightly different speed as a result of component variation. Furthermore, this speed is affected by the internal computer temperature—which is dependent on how long the computer has been operating and the surrounding room temperature.

Therefore, you may find your program running too fast or too slow. You can adjust this speed by changing 97 to a higher number for slower counting, or a lower number for faster counting. Even changing by one count (say, from 97 to 96) will have an observable effect over a 30-second period. "Patience is a virtue," it has been said, and

you might have to be very virtuous to get this count accurate. Fortunately, you won't be timing Space Shuttle flights and reentry with this program!

Minutes and Seconds

Lines 860–920 keep the minutes and seconds in order, and tally the total charge. Line 860 subtracts 1 from the seconds count and looks to see if it has gone below zero. IF it has, THEN the program jumps to line 900. If not, back to line 820 to update the screen and count down another second.

When t becomes −1, line 900 is active. The computer puts out a short BEEP, the value of t is set to 59, the value of variable a (minutes) is increased by one, and the value of the present charge, variable c, is increased by the incremental minute charge, variable m.

Line 910 increments the BORDER color variable, b, by one, and then checks to see that it has not exceeded 7. IF it has, THEN it is reset to equal 0.

Finally, the program GOes back TO line 820 for screen updating and countdown of the next minute and repeats lines 820–920 until you press CAPS SHIFT and BREAK to end the program. This leaves all the information on the screen for you to log manually if you keep track of toll calls.

MODIFICATIONS

Try to devise a more accurate counting system that can be "tuned" more accurately. This could involve nested loops in line 850, counting within each other. Bear in mind that no matter what you do to initially establish an accurate count, it will "drift" with temperature change.

You might also want to add some significant three-minute signal for operator-assisted calls, like FLASHing the entire screen for 30 seconds before the three minutes are up.

QUESTIONS

1. Where do you find local telephone rate information?
2. What is the minimum time charge for an operator-assisted call?

3. In what increments of time are phone charges usually made? How about special long-distance services?
4. How do you obtain the reversed characters shown in line 100?
5. What happens if you respond with a capital (uppercase) Y to the question in line 160?
6. Why is a FLASH 0 statement necessary in line 1000? What would happen if it were left out?
7. If the 97 in line 850 were changed to 100, would the program count seconds slower or faster?
8. How does the program determine if you want additional information in lines 160 and 170? Describe the exact process.
9. How many times can a subroutine be used within a program?

VARIABLES

a	initial minutes; total minutes
b	BORDER color
c	initial charge; total charge
m	charge per minute
n$	number called
r$	response to y/n question
t	seconds to next charge
x	delay loop counter
z$	dummy input to continue

13
Bio-Graphs and You

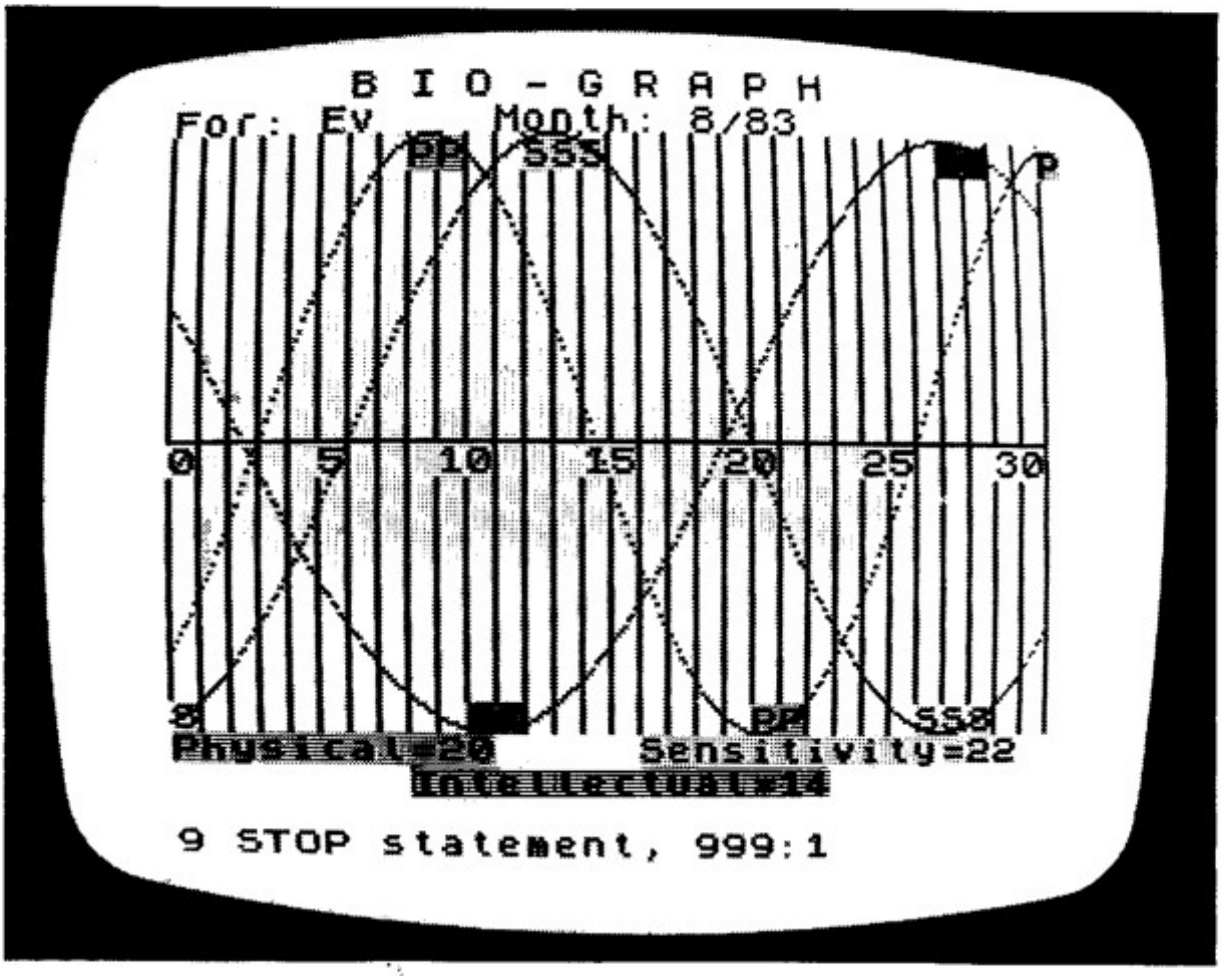


Photo 13-1. Typical bio-graph.

INTRODUCTION

Biorhythms have been a topic of discussion for years. If you know what biorhythms are, you don't need to hear more about them from me—and you'll love this program! If you don't know about biorhythms, then the discussion of the program will go into just enough detail to perhaps get you curious. In either case, "Bio-Graphs" are merely my name (I love to be different) for biorhythm charts—curves that supposedly show your daily Physical, Sensitivity, and Intellectual "highs and lows" based on how long you've lived. You don't have to "buy" it to try it, so put down your knapsack, pick up a canteen, and settle back for a bio-break.

PROGRAM DESCRIPTION

LISTing 13-1 shows the program, and Fig. 13-1 shows a typical "Bio-Graph." The program starts by asking for your birthday, the month for which you want a Bio-Graph, and your name. It then calculates both the number of days you've been alive and the number of days in the target month. The graph grid is DRAWn very quickly on the screen, with a horizontal line labeled every five days.

LISTing 13-1

```
 100 REM * "BIO-GRAPH" © Fred Bl
echman 1983 *
 105 GO SUB 2000
 110 FOR l=0 TO 8*days STEP 8
 120 PLOT l,15: DRAW 0,144: NEXT
 l
 200 PLOT 0,88
 210 DRAW days*8,0
 400 PRINT AT 11,0;"0": PRINT AT
 11,5;"5"
 410 FOR n=10 TO days STEP 5
 420 PRINT AT 11,n-1;n
 430 NEXT n
 500 REM * Physical Curve *
 510 PAPER 5: GO SUB 2200
 520 LET a=360/(8*23): LET s=a*d
*8: LET a$="P"
 530 GO SUB 1000
 600 REM * Sensitivity Curve *
 610 PAPER 6: GO SUB 2300
 620 LET a=360/(8*28): LET s=a*d
*8: LET a$="S"
 630 GO SUB 1000
 700 REM * Intellectual Curve *
 710 PAPER 4: GO SUB 2400
 720 LET a=360/(8*33): LET s=a*d
*8: LET a$="I"
 730 GO SUB 1000
 740 PAPER 7
 999 STOP
1000 FOR x=0 TO 8*days
1005 LET y=88+72*SIN (s*.0174533
)
1006 IF y>158 AND x/8=INT (x/8)
THEN PRINT AT 2,x/8;a$
1007 IF y<18 AND x/8=INT (x/8) T
HEN PRINT AT 19,x/8;a$
1010 PLOT x,y
1020 LET s=s+a
1030 NEXT x
1040 RETURN
2000 INPUT "Your BIRTHDAY (MM/DD
```

LISTing 13-1—cont.

```
/YY)?";m,d,y: LET y=y-1
2005 GO SUB 3000
2010 LET ds=dt
2020 INPUT "Bio-Graph month (MM/
YY)?";m,y: LET d=1: LET days=m:
LET month=m: LET years=y: LET y=
y-1
2025 GO SUB 3000
2030 LET de=dt
2040 LET r=de-ds: IF r<0 THEN PR
INT "OOPS!! Dates entered wrong!
": FOR t=1 TO 250: NEXT t: CLS :
 GO TO 2000
2082 IF days=2 THEN LET days=28
2084 IF days=28 AND years/4=INT
(years/4) THEN LET days=29
2087 IF days=4 OR days=6 OR days
=9 OR days=11 THEN LET days=30:
GO TO 2090
2088 IF days<>28 AND days<>29 TH
EN LET days=31
2090 INPUT "What is your FIRST N
AME?";n$
2092 PRINT "       B I O - G R A
P H"
2093 PRINT "For: ";n$;"     Month
: ";month;"/";years
2095 RETURN
2100 REM * Calculate days for ea
ch cycle *
2200 REM * Physical Cycle *
2210 LET q=INT (r/23): LET d=r-(
q*23): PRINT AT 20,0;"Physical="
;d
2220 RETURN
2300 REM * Sensitivity Cycle *
2310 LET q=INT (r/28): LET d=r-q
*28: PRINT AT 20,16;"Sensitivity
=";d
2320 RETURN
2400 REM * Intellectual Cycle *
2410 LET q=INT (r/33): LET d=r-q
*33: PRINT AT 21,8;"Intellectual
=";d
2420 RETURN
3000 LET yt=y-1901
3010 LET lt=INT (yt/4)
3020 IF y/4=INT (y/4) AND m<3 TH
EN LET lt=lt+1
3100 IF m=1 THEN LET mt=0: GO TO
 3400
3110 IF m=2 THEN LET mt=31: GO T
O 3400
3120 IF m=3 THEN LET mt=59: GO T
```

LISTing 13-1—cont.

```
0 3300
3130 IF m=4 THEN LET mt=90: GO T
O 3300
3140 IF m=5 THEN LET mt=120: GO
TO 3300
3150 IF m=6 THEN LET mt=151: GO
TO 3300
3160 IF m=7 THEN LET mt=181: GO
TO 3300
3170 IF m=8 THEN LET mt=212: GO
TO 3300
3180 IF m=9 THEN LET mt=243: GO
TO 3300
3190 IF m=10 THEN LET mt=273: GO
 TO 3300
3200 IF m=11 THEN LET mt=304: GO
 TO 3300
3210 IF m=12 THEN LET mt=334
3300 LET dt=(yt*365+306)+lt+mt+d
: LET y=y+1
3310 IF y/4=INT (y/4) THEN LET d
t=dt+1
3330 RETURN
3400 LET dt=(yt*365+306)+lt+mt+d
: RETURN
```

Next the program PLOTs the Physical, Sensitivity, and Emotional curves one at a time, with notations giving the position of each curve on the first day of that month. You can then COPY the graph on your printer and compare it with your life that month. You can also do this for your friends, or examine the Bio-Graphs of famous people in history. (What were Lindbergh's bio-curves when he flew the Atlantic solo?)

DISCUSSION

I've played around with biorhythms ever since I first heard about them over 15 years ago. I was fascinated by the mathematics involved, regardless of whether the theory of repetitive biological cycles since birth is really factual.

I've found it great fun to PLOT my biorhythm cycles occasionally and then compare the curves with how I really feel on given days. (And when things go wrong you can cop-out by blaming it on your biorhythms!) Whatever use you make of the curves, this program will PLOT them for you. Just don't take them too seriously.

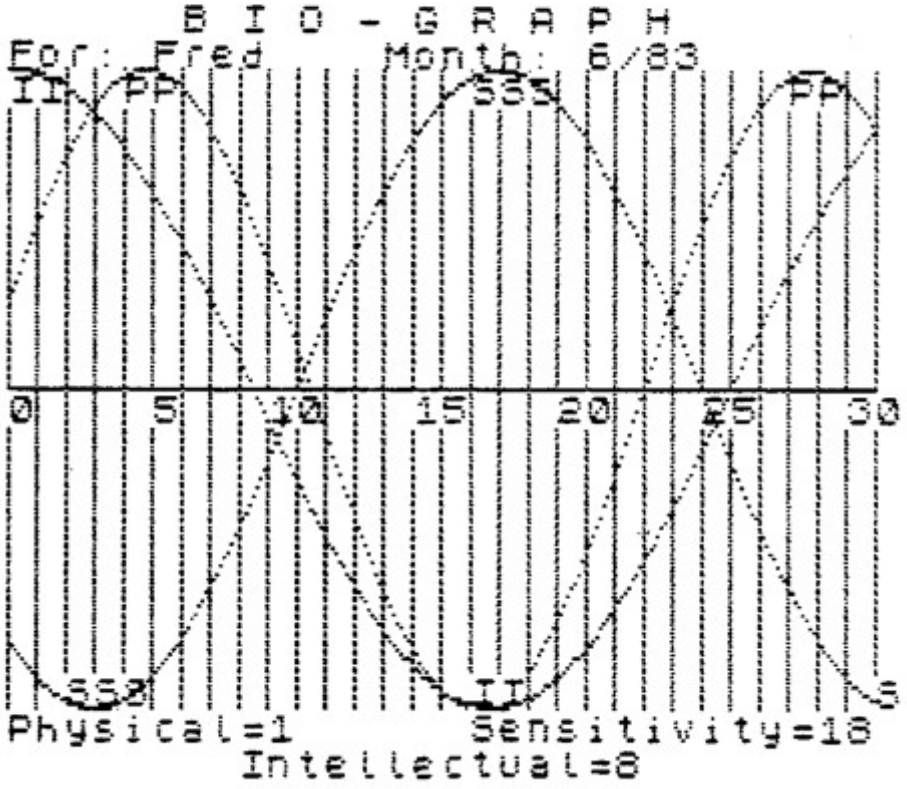


Fig. 13-1. Typical bio-graph.

The Theory

There are lots of books that cover the details of the theory of biorhythms, but suffice it to say here that when you are born, you supposedly begin following three biological cycles. The Physical cycle runs 23 days, the Sensitivity (sometimes called "Emotional") cycle lasts 28 days, and the Intellectual cycle repeats every 33 days. Well, if we assume the theory is correct, then the PLOTting of the curves is a snap if you start the day you're born—three sine waves all starting at zero on Day 0 (birthday) and moving on from there. However, since the curves are different lengths (23, 28 or 33 days), soon after you're born they all become intermingled.

If you think about it, it's not necessary to go back to your birthday and PLOT all the curves from that date. All you really need to do is find out how many complete cycles each curve has completed, and the number of days each is into its present cycle. You then PLOT the curves starting at that point in time.

An Example

For example, if you have been alive 21 years, that's either 7670 or 7671 days, depending on the leap years in between. (Take 21, multiply by 365 and add 2¼.) Let's say it's been 7670 days. OK, divide

by 23 (Physical cycle length) and you get 333.47826. Huh? Well, that's obviously 333 complete physical cycles plus some number of days. But how many days? Just multiply 333 by 23 and you get the number of full cycle days, 7659. Obviously, then, 7670 minus 7659 is 11, the number of days you have presently moved into your Physical cycle. Confused? Go over this paragraph again (unless you're at an Intellectual "low").

Similarly, you can calculate the number of days into your Sensitivity and Intellectual cycles, and then plot the curves "offset" by that number of days. You'll see exactly how this is done when you follow the Program Anatomy. When the curves are PLOTted they are identified at positive and negative peaks in color (on a color TV), and the same color is used just below the graph to indicate the number of days into the present cycle for each curve.

PROGRAM ANATOMY

Line 100 titles the program and line 105 immediately branches to the SUBroutine starting at 2000. This line asks you for your birthday and then subtracts one from the year to determine the full years. Line 2005 then jumps into the "number of days" SUBroutine starting at line 3000.

Number of Days SUBroutine

I've seen several of these "number of days between two dates" programs, but couldn't figure out how they worked—so I wrote my own. This program module can be used in other programs—especially business and financial programs—so pay attention!

The approach I decided upon was to relate the starting date (the subject's birthday) and the ending date (the first day of the selected month) to the first day of March, 1900. All years from that date until February 28, 2100 have a leap-year day if the year is divisible by four with no remainder.

While it is not generally known, century years (such as 1700, 1800, 1900) are *not* leap years *unless* they are divisible by 400 with *no* remainder (like the year 2000). So, to keep things relatively simple, this program module (and hence the accuracy of the entire program) is limited to dates between March 1, 1900 and February 28, 2100. (Note: Anyone using this program AFTER February 28, 2100 may request a FREE update from my great, great, great grandson!)

By simply calculating the number of days between each of the target dates and March 1, 1900, and then subtracting the larger result from the smaller, you have the number of days between the two target dates. See Fig. 13-2.

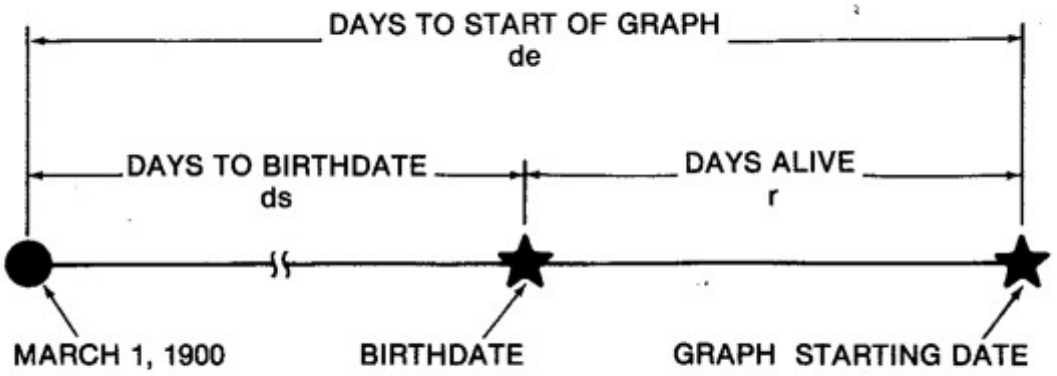


Fig. 13-2. Calculating the days between two dates.

This process starts in line 3000 where a new variable, yt (standing for total years) is set equal to the just-entered birth year (minus one) minus 1901. This gives you the number of full years. To this you'll later add the months and days to get the total number of days from March 28, 1900 to your birthdate.

Leap Days

Line 3010 determines how many leap years fall into this range of years by simply dividing the number of years by four, and calling this lt (total leap days). However, if the target date falls before March 1 on a leap year, another leap day may need to be added, and this is done by line 3020. This line, in plain English, says, "IF the year number divided by four has no remainder, THEN add one more leap day." The target year divided by four is compared with the INTeger of the target year divided by four. IF they are equal, THEN one is added to lt. Mull this over and it will make sense.

Monthly Days

Lines 3100 to 3210 are really very simple to follow. They calculate the number of days for the full months of the *partial* last year. Each statement checks the value of m (month) that you enter in response to the question in line 2000. The numbers for mt (monthly totals) simply reflect how many days there have been in the *full* months up to that point.

For example, suppose your birthday was in April. You would enter a 4 in response to the "Birthday?" question. Line 3130 recognizes this and assigns mt a value of 90, which equals 31 days in January, plus 28 days in February, plus 31 days in March. $31+28+31=90$. What about April days? Since April is not a full month, we count those days separately, in the next program line. What about if February has 29 days? Read on, McDuff!

Line 3300 puts all the numbers together. It starts by taking the years since 1901 (yt) and multiplies this by 365, to get the total number of days in those full years. Then it adds 306—the number of days from March 1, 1900 to January 1, 1901. Next the total leap days (lt) are added, then the month days (mt) are added, and finally the days of the final month (d) are added. Whew!

Next, in the same program line, the birthday year, which was reduced in line 2000, is restored to its original value. Why? To see if it's a leap year, as in line 3310. If it is, another day is added to the total days (dt). Line 3300 RETURNs to line 2010 (the line following the GO SUB from whence it came) and assigns the variable ds (for starting date) equal to dt.

Notice a little "bypass" at the end of lines 3100 and 3110. If your birthday is in either January or February, and it also happens to be a leap year, line 3310 would add a day too many, so the program jumps to line 3400. This line makes the same calculation for dt as line 3300, but without checking for a leap year adjustment. Then RETURN sends it to line 2010, where ds is set equal to dt. So, either way, after all is said and done, the variable ds now represents the number of days from your March 1, 1900 to your birthday.

Bio-Graph Month Input

Line 2020 is the line that asks you to specify what month and year you want for the Bio-Graph. Since the graph always starts on the first of the month, the variable d is set to 1. Setting "days" and "months" equal to m and "years" equal to y is done here for use later in the program.

Line 2025 goes to the number-of-days SUBroutine starting at line 3000, which operates exactly as just described for the birthday calculation. The result is again dt in line 3300 or line 3400, but the SUBroutine RETURNs to line 2030 this time, and de (end date) is set equal to dt.

Now, if you think about it, we have a number of days for the ending date (de) and another number of days for the starting date (ds), and

all we need to do to get the difference between them—the number of days between the two dates—is to subtract ds from de. This is done in line 2040, with the result assigned to variable r. This line also checks to see if the result is less than 0. If it is, you're trying to get a Bio-Graph for before you were born, or you made some grandiose entry error. If you did, the program goes back to line 2000, after a short FOR-NEXT delay loop, and clears the screen for another try.

Days in a Month

The next program line is 2082. Using the value of "m" for "days" established in line 2020, lines 2082–2088 determine how many days there are in the graph month. Obviously, if the month is "2," for February, there would normally be 28 days. However, line 2084 checks to see if this is a leap year, and if it is, then "days" equals 29.

Line 2087 looks for April, June, September or November, which are months with 30 days. It assigns "days" a value of 30 and then jumps around the next line to line 2090. Line 2088 concludes that if there are not 28 or 29 days in this month, then there must be 31 (since if there were 30 days, this line would have been bypassed altogether). The whole purpose of these shenanigans is to determine how many days will be shown on the Bio-Graph.

Name and Title

Line 2090 asks you for your first name, and then lines 2092 and 2093 print the graph title, your name, and the month and year of the Bio-Graph. The RETURN on line 2095 sends the program back to line 110. (Believe it or not, after all this time we're only up to the third line in the program listing!)

DRAWing the Graph Grid

Now we'll be DRAWing the graph grid—the vertical lines that show the days of the month, and the "axis," or horizontal line that represents the zero or "critical" value for each curve. If you've forgotten the graphic commands and screen layout, refer back to Chapter 9, "Grappling With Graphics," and refer to the Fig. 3-1 Video Display Layout for PLOT and PRINT at locations.

Line 110 starts a FOR-NEXT loop with the variable I, going from zero to eight, multiplied by the number of days in the desired month. Why eight? Because there are eight "pixels" for each character

space, and each day on the graph will use one character space. The reason for the "STEP 8" is that we will DRAW vertical lines one character space apart.

Line 120 starts with a PLOT instruction at location I across and 16 up. Since I is zero at this point, that means the PLOT location is on the left border, 16 pixels up from the bottom. Actually, a small dot is placed at this location. The next instruction on that program line says "DRAW 0,144." This means, "starting where you are, DRAW a straight line that does not move horizontally (0) but goes up 144 pixels." ZAP! Faster than you can blink, a line is DRAWn.

The next statement on line 120 increments the FOR-NEXT loop and the value of I becomes 8 (because of the STEP 8 in line 110). Another vertical line is DRAWn 8 pixels to the right of the first one. And on it goes, until the value of I is equal to "8 times days." For example, if "days," as assigned in line 2020, were 30, then the last vertical line would be at horizontal pixel location 240. This all happens so fast you must watch closely to see the sequence on the screen.

Drawing the Axis

At this point there are several vertical lines on the screen, five character spaces apart, with "B I O - G R A P H" on the top line, and your name with the graph month and year on the second line. The vertical lines snug up right against the printing on the second line. But we need a horizontal axis, and we need to put reference numbers along this axis to indicate the days in that month. This is done starting at line 200. First a dot is placed at the left border 88 pixel-locations up from the bottom of the usable screen area (the bottom of the twenty-second screen line) with PLOT 0,88. Next line 210, using this location as a starting point, DRAWs a line horizontally to the right. The length is the number of days for the desired month (like line 110) times eight pixels for each day. In other words, if "days" is equal to 30, then the line will be drawn to extend 240 places to the right. This, of course, also corresponds with the right-most vertical line. Isn't that neat—and fast?

Getting the numbers on this horizontal line is a bit tricky, since some are one-digit, some are two-digit, and I wanted to start with zero, not one. This leads to some problems in placement. So I chose to put 0 and 5 exactly where I wanted them, using line 400, and then let lines 410–420 create a simple FOR-NEXT loop to put the rest of the numbers in, five spaces apart. Notice that the variable "days" is

again used, and the numbers stop at the last multiple of five before the end of the line.

So much for the mechanics of DRAWing the grid. Now let's tackle what we are really after—the biorhythm curves.

Watch Those Curves!

Lines 500–740 draw the curves, with the help of the two SUBroutines. Now, this uses a little math and some trigonometry (you know—the subjects you hated in high school!) and is the real core of the program. I'll go through the PLOTting of the Physical Cycle. The Sensitivity and Intellectual curves follow the same procedure. Take another swig from your canteen before you go on since this much detail might dehydrate you!

The Physical Curve

Line 510 starts by telling the computer to set the PAPER to 5 (cyan) as a background color. The background does not change, however, until the next printing command, and only changes in those character spaces used. This is absolutely great for highlighting certain printing, as you'll see. Even on a black-and-white display, each "color" has a different shade of gray.

Line 510 then jumps to the SUBroutine starting at line 2200. (The REMark statement on this and many lines in the program are nonfunctional as far as the computer is concerned. They simply are your signposts in the programming jungle.) Line 2210 takes the value of r determined in line 2040 (the number of days from your birth to the first day of the graph), divides it by the cycle length (23 for Physical), and then extracts the INTeger—the number before the decimal point. (INT rounds down to the next lowest whole number.) That takes care of the full cycles since your birth. But now the computer needs the remainder in days. By taking the number of full cycles (q) and multiplying it by 23, and then subtracting this from r, the number of days already completed in the present cycle, d, is obtained. This point on the curve, then, will be the starting point of the graph for that curve.

Advancing the Cycle

Line 2210 now prints (with the cyan background) on screen line 20 the number of days already passed in your current Physical cycle,

just as a reference. Line 2220 shoots the program back to line 520, the first line following the GO SUB 2200 in line 510.

Time for a bit of simple trigonometry. (Is there such a thing?) A circle has 360 degrees. If you take the value of the SINe of each degree of a circle and plot it out on a straight horizontal line, you get a SINe wave. (See Fig. 13-3.) This runs from zero to a vertical height of one at 90 degrees, zero at 180 degrees, minus-one at 270 degrees, then back to zero at 360 degrees.

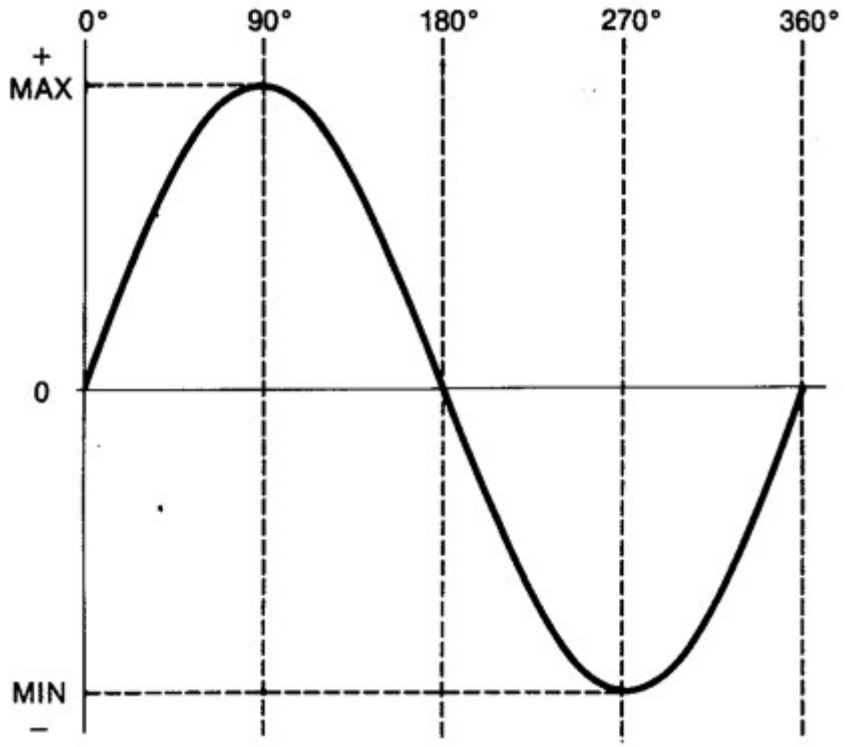


Fig. 13-3. The SINe curve.

If we want a SINe wave with a specific number of points, all we have to do, to start, is divide that number of points into 360 to find out how many degrees between each point. We know that going along the axis of the graph we are using eight pixels for each day, so multiplying "days" by eight gives us the number of points required. Line 520 starts by doing this calculation and calling the answer "a." That means a is the number of degrees that each point will increase from the previous point.

The next part of line 520 simply says that a new variable, s, will be the sum of the incremental angles up to the point where the graph will begin. In other words, the number of days already into the cycle (d from line 2210) multiplied by eight pixels per day, and then multiplied by the incremental angle, a. This, in effect, advances the starting point of the graph before any PLOTting.

PLOTting the Curve

The last part of line 520 sets string variable a$ to "P" for Physical. This letter will be used to identify the curve. Line 530 now sends the program to the SUBroutine at line 1000. We're approaching the end of the trail, and the camp is dead-ahead, so don't give up now!

Line 1000 sets up a FOR-NEXT loop to count the number of PLOTting locations—eight times the number of "days" in the graph month. The next line, 1005, is the tiger that has been lying in wait for you in the jungle. Conquer this "beast" and you're ready to tackle elephants!

To understand this line, look at Fig. 3-4. The vertical grid lines extend 72 pixel spaces above and below the axis, which is 88 pixel spaces above the bottom of usable space, line 21. The SINe wave we want to PLOT must be designed to fall within the upper and lower limits, and must have its beginning and end on the axis. Since the maximum sine of an angle is one, we must multiply this by 72 to reach the top of the graph. Also, we must PLOT each position of the curve with 88 as the zero point.

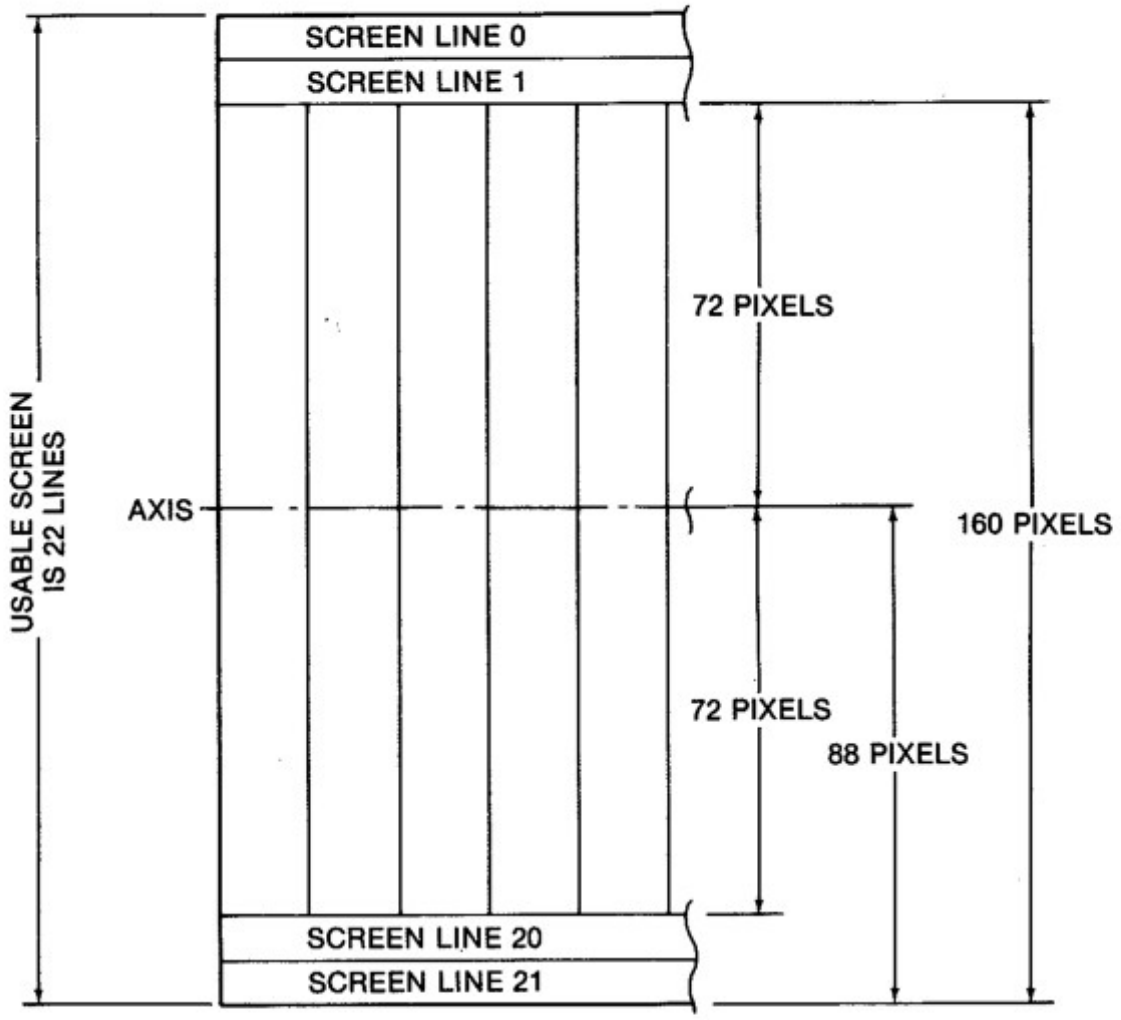


Fig. 13-4. Positioning the graph vertically.

Radians

Furthermore (enough, already!) the computer has the gall to calculate the SINe of an angle in *radians* instead of degrees! What are radians? Look back to Chapter 9, "Grappling With Graphics," to Fig. 9-8.

To summarize, a radian is the angle included in the length along a circle equal to the radius. The circumference of a full circle is 2 times PI times the radius. So, if you divide the number of degrees in a circle (360) by 2 times PI, or 6.2831852, you get 57.29578—the number of degrees in a radian. Now since there are this many degrees in a radian, then each degree is equal to 1 divided by 57.29578, or .0174533 radians. So, to convert from degrees to radians just multiply the number of degrees by .0174533. (And to go from radians to degrees, should you choose to accept this mission, just multiply by 57.29578.)

Line 1005 should now make sense. The value of y, the vertical PLOT coordinate, is set equal to 88 (so that it starts on the axis) plus 72 (maximum height) times the SINe of the beginning angle converted from degrees to radians. This value will range from plus 72 to minus 72.

Labeling the Physical Curve

If there were only one curve on the graph it would probably not be necessary to label it. The graph title could tell you what the curve is. However, with three curves to be drawn on the same grid, it becomes necessary to label each curve. One approach is shown in lines 1006 and 1007. The intent here is to label just the peaks and valleys of each curve. Line 1006 looks for a peak value (158) just below the maximum possible of 160 (88+72). It also checks that this is a character space by comparing x/8 with the INTeger of x/8. IF the curve is near the peak, AND the curve is in a character space, THEN the letter "P" (as assigned to a$ in line 520) is printed on a cyan background on screen line 2—the top line of the graph, at the designated character location. Similarly, line 1007 looks for valleys to print the "P" on a cyan background on screen line 19.

Finally, line 1010 PLOTs the calculated point, line 1020 increments the angle by a, and line 1030 continues the loop until the maximum value of x in line 1000 is reached. Then line 1040 RETURNs to line 600.

The Other Curves

The Sensitivity and Intellectual Curves are drawn in the same manner (lines 600–730), with "S" and "I" used to label the curves, and with yellow (PAPER 6) and green (PAPER 4) used respectively as the background colors for the printing. Why aren't the curves each drawn in color? Well, each character space, containing 64 pixels (8 across and 8 up and down) can only have one PAPER and INK color at one time—unless you're in Display Mode 4, but that's a "zebra of another color" as they say here in the jungle. If you PLOT into a character space with a new INK or PAPER color, it changes *all* the pixels in that character space to that color of INK or PAPER, making a real mess in this case.

ENDING IT ALL

Since the last PAPER color used was green (line 710) it is necessary to return the PAPER to white for the next RUN. This is done in line 740. Although nothing appears to happen at this point, the next RUN will convert the entire active area of the screen to white. Line 999 STOPs the program from crashing through into the SUBroutine at line 1000. End of program.

WHAT NOW?

Use the COPY command if you have a printer and you'll get a beautiful one-color printout of the screen display. The printer does not recognize color and PRINTs only INK dots, regardless of color. Use the program any way you like. It's great at parties—plot everyone's Bio-Graph. Start your own Bio-Graph business (after arranging with me for a royalty, of course!).

There are plenty of books which explain how to interpret the curves and the "critical days" when they cross the axis. I won't go into that here. You might note, in checking closely with various references, that some yield curves offset one day from another. That's because some start with the first graph day as "zero" and some start with "one"—so don't be alarmed if the results of this graph appear to be "off" one day.

By all means, use Bio-Graphs only for entertainment. It's not a good idea to take this whole theory seriously enough to start planning your life around your cycles.

MODIFICATIONS

You might want to make each of the curves a different height, to help visually separate them. As designed, all 22 screen lines are used; you might prefer not to make the graph so high by using additional screen lines to print peak, valley and critical days for each cycle. You also might want to add plus and minus signs at the top and bottom of the graph.

QUESTIONS

1. How many days are in each of the three cycles?
2. How can you tell if a specific year is a leap year?
3. The year 1900 was not a leap year. Will the year 2000 be a leap year? Why?
4. The graph grid uses DRAW for all the lines. Could the graph be drawn entirely with PLOT? Why is DRAW better?
5. What value is used to convert from degrees to radians? Do you multiply or divide?
6. Line 3300 calculates dt as the total days from birthday to the beginning of the graph month. Explain each item in the formula. Why is 306 in there?
7. Exactly what would happen on the display if line 999 were removed from the program?
8. Why is this program not accurate for dates before March 1, 1900?
9. What colors does the printer recognize?

VARIABLES

a	angle increment
a$	curve identifier
d	days; days into present cycle
days	days in graph month
de	days from 3/1/1900 to first day of graph month
ds	days from 3/1/1900 to birthday
dt	total days
l	vertical graph line
lt	total leap year days
m	months
month	graph month number
mt	total monthly days in graph year
n	day number
n$	name
q	full cycles since birthday
r	days from birth to start of graph
s	starting angle; sum of angle increments
x	horizontal curve coordinate
y	years; vertical graph coordinate
years	graph year number

SECTION B
HOBBY PROGRAMMING

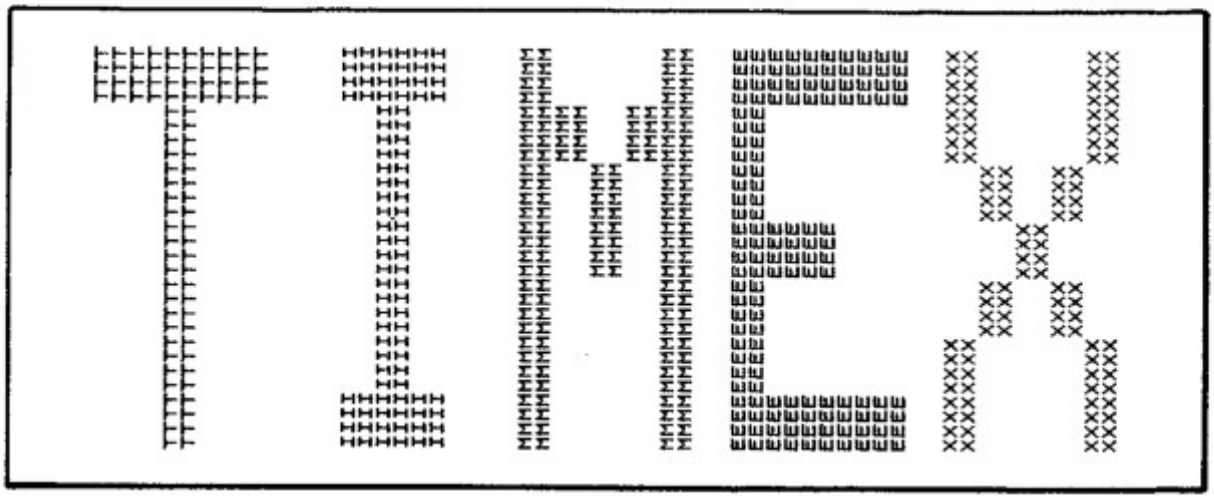
The programs in Section B are just three examples of how you might be able to "tame" the T/S 2068 to do something useful for your hobby. With the cost of computer and printer combined now under $300, you can't use that excuse anymore! (You probably spend more than that on a weekend with your hobby.)

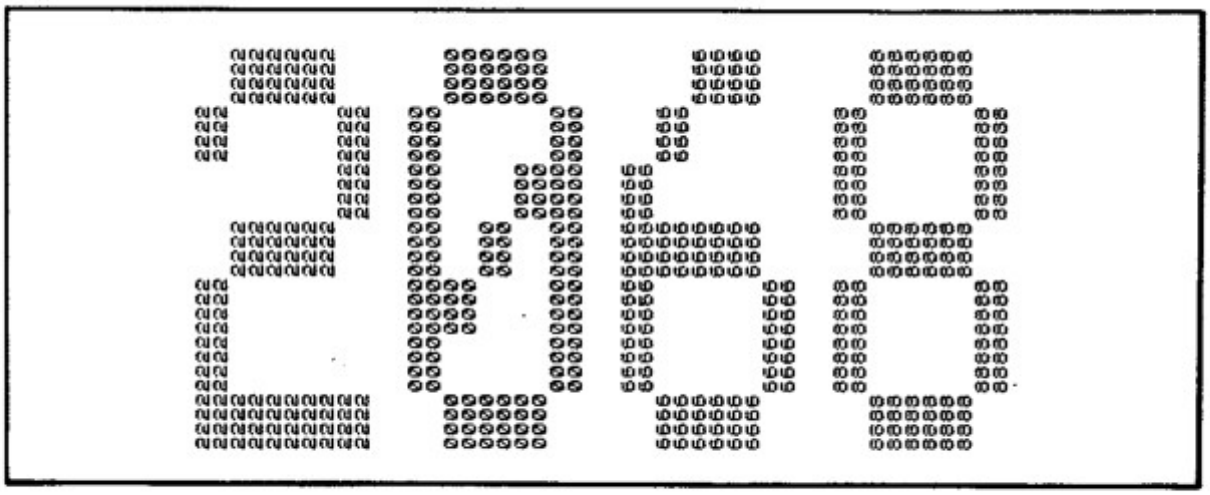
Chapter 15 is for collectors—from rock hounds to those of you running around with butterfly nets! If whatever you collect has a value, this program can be useful to you.

Chapter 16 is for the Nonmusical types who have a strong constitution and can stand the kind of "music" a computer produces. You can actually compose music and play it with the information in this chapter—and display the notes on the screen as the music plays!

Chapter 17 is for those of you who own boats or airplanes, or are involved with long-distance radio communication. But you'll still need to know where North is, so don't forget your compass!

14
BINARY Banner





Banners can be pasted on larger sheets.

INTRODUCTION

How would you like to make banners on your Timex printer? You know—banners like "MISS AMERICA" or "WELCOME HOME DAD"? Well, with the simple program given here, a T/S 2068 computer and a Timex Sinclair 2040 Personal Printer, you can produce banners as long as you like—and you can even design your own character fonts.

DISCUSSION

Why the peculiar word BINARY in the title? That stands for "Boy It's Neat And Really Yuseful." (Everyone knows computer programmers can't spell.) Besides, the binary numbering system is used to put any characters you want on these banners. You can use the banners for birthdays, for parties (like New Year's Eve—giant confetti with your name on it!) or, if the depression hits, wrap yourself in a banner instead of a barrel to cover your bare bones. Out here in the jungle we use them as identification bracelets for the elephants.

Actually, they really are fun at a party as an ice-breaker. Make one up for each guest and have him get everyone else to autograph it. You can also use them above (or on the front of) your desk, as a "nameplate." You'll certainly get attention!

PROGRAM ANATOMY

The program is shown in LISTing 14-1, with DATA for numbers and uppercase letters only. Later on I'll describe how you can add whatever character shapes you want—using a form of binary code. But first let's see what the program does, line by line.

LISTing 14-1

```
  10 REM * BASIC Banner * © Fred
 Blechman 1983
  20 CLS : LET p$="": INPUT "Mes
sage?";m$
  30 FOR l=1 TO LEN m$: LET a$=m
$(l TO l): PRINT a$;
  40 READ x$,l1,l2,l3,l4,l5: IF
a$=x$ THEN GO TO 100
  50 GO TO 40
 100 LET x=l1: GO SUB 200
 110 LET x=l2: GO SUB 200
 120 LET x=l3: GO SUB 200
 130 LET x=l4: GO SUB 200
 140 LET x=l5: GO SUB 200
 150 LPRINT : LPRINT : RESTORE :
 NEXT l: STOP
 200 IF x-64>=0 THEN GO SUB 500:
 LET x=x-64: GO TO 220
 210 IF x-64<0 THEN GO SUB 510
 220 IF x-32>=0 THEN GO SUB 500:
 LET x=x-32: GO TO 240
 230 IF x-32<0 THEN GO SUB 510
 240 IF x-16>=0 THEN GO SUB 500:
```

LISTing 14-1—cont.

```
 LET x=x-16: GO TO 260
 250 IF x-16<0 THEN GO SUB 510
 260 IF x-8>=0 THEN GO SUB 500:
LET x=x-8: GO TO 280
 270 IF x-8<0 THEN GO SUB 510
 280 IF x-4>=0 THEN GO SUB 500:
LET x=x-4: GO TO 300
 290 IF x-4<0 THEN GO SUB 510
 300 IF x-2>=0 THEN GO SUB 500:
LET x=x-2: GO TO 320
 310 IF x-2<0 THEN GO SUB 510
 320 IF x-1>=0 THEN GO SUB 500
 330 IF x-1<0 THEN GO SUB 510
 340 LPRINT p$: LPRINT p$: LET p
$="": RETURN
 500 LET p$=p$+a$+a$+a$+a$: RETU
RN
 510 LET p$=p$+"    ": RETURN
1000 DATA " ",0,0,0,0,0
1010 DATA "A",124,18,17,18,124
1020 DATA "B",65,127,73,73,54
1030 DATA "C",62,65,65,65,34
1040 DATA "D",65,127,65,65,62
1050 DATA "E",127,73,73,65,65
1060 DATA "F",127,9,9,1,1
1070 DATA "G",62,65,65,73,121
1080 DATA "H",127,8,8,8,127
1090 DATA "I",0,65,127,65,0
1100 DATA "J",32,64,64,64,63
1110 DATA "K",127,8,20,34,65
1120 DATA "L",127,64,64,64,64
1130 DATA "M",127,2,12,2,127
1140 DATA "N",127,2,4,8,127
1150 DATA "O",62,65,65,65,62
1160 DATA "P",127,9,9,9,6
1170 DATA "Q",62,65,81,33,94
1180 DATA "R",127,9,25,41,70
1190 DATA "S",38,73,73,73,50
1200 DATA "T",1,1,127,1,1
1210 DATA "U",63,64,64,64,63
1220 DATA "V",7,24,96,24,7
1230 DATA "W",127,32,16,32,127
1240 DATA "X",99,20,8,20,99
1250 DATA "Y",3,4,120,4,3
1260 DATA "Z",97,81,73,69,67
1300 DATA "1",0,66,127,64,0
1310 DATA "2",114,73,73,73,70
1320 DATA "3",34,65,73,73,54
1330 DATA "4",24,20,18,127,16
1340 DATA "5",39,69,69,69,57
1350 DATA "6",60,74,73,73,48
1360 DATA "7",1,113,9,5,3
1370 DATA "8",54,73,73,73,54
1380 DATA "9",6,73,73,41,30
1390 DATA "0",62,81,73,69,62
```

Initialization

Line 10 identifies the program and author, and establishes program ownership. Notice the special copyright symbol? It's hidden under the word RESET below the P key, and you get it by pressing P when you go into the E-cursor mode from an L-cursor mode. However, be careful! If you happen to press the P key in the E-cursor mode, but you were in the K-cursor mode just before that, you'll get RESET.

Line 20 clears the screen and sets the string variable p$ to a null. Next the INPUT statement stops the computer and waits for you to enter the message you wish to have made into a banner. Notice that there are three statements on line 20, separated by colons. Although you can't put multistatement lines on the TS1000/1500/ZX81, this can be done on the T/S 2068.

Your Own Message

When you type in the message you want, the program stores this message as string variable m$. (Incidentally, although lowercase variables are used throughout this program, the T/S 2068 treats them as uppercase variables, making no distinction.)

Line 30 sets up a FOR-NEXT loop based on the length of your message, and then takes one character at a time, calls it a$, and prints this character on the screen. Line 40 READs a complete line of DATA, starting with Line 1000, the first DATA line. The first DATA item in each line is read as x$. This is then compared with a$, your message character. If they are identical, then the program proceeds to Line 100. If they are *not* identical (and here uppercase *is* distinguished from lowercase), the program moves to Line 50, where it goes back to Line 40 and READs the next line of DATA.

Invalid Letter

If your message letter is not found, the program merely runs out of DATA and ends with an error message—so be sure your message only contains characters that are programmed in the DATA. More on that later.

Meanwhile, assuming your message character (and we're still on the first message character at this point) is "found" (matched) with a DATA entry, the program proceeds through Lines 100–140 and the SUBroutines in Lines 200–510 to actually print the character. These

SUBroutines are the real heart of the program, and we'll cover their operation in detail later on. For now, just try to follow the overview. Operating together with the numbers in the DATA statements, the SUBroutines allow you to program any shape that will fit in a 7 by 5 matrix.

Line 150 prints two blank lines so that the banner characters are separated. The RESTORE puts the "data pointer" back to the first DATA statement (Line 1000). The "NEXT I" loops the program back to "LET a$=m$(I TO I)" in Line 30 to select the next character of your message. This continues to the end of the banner, when the STOP at the end of Line 150 terminates the program.

The SUBroutines—7-Bit Binary Code

The SUBroutines do the actual forming and printing of the characters you've put in DATA statements. But how does that work, and how can you program your own characters? Well, it may take some head-scratching to follow this, but give it a try, and you may be able to penetrate the mumbo-jumbo world of binary code. The real key is understanding Fig. 14-1, a typical character and its coding.

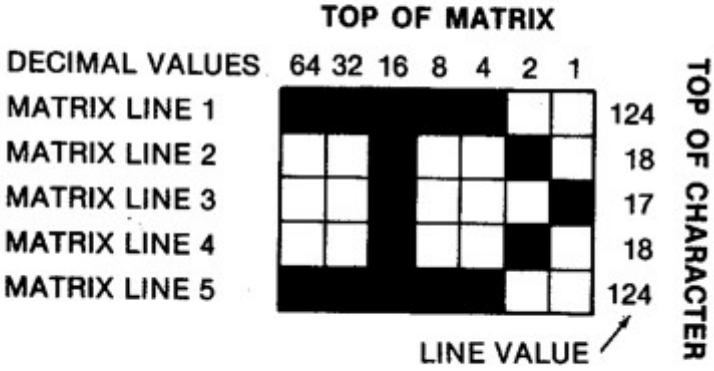


Fig. 14-1. Capital A matrix.

Fig. 14-1, a "matrix" 7-across by 5-high, is an example of a single character to show how the system works. The boxes filled in black form the letter A lying on its right side. (Turn the figure 90-degrees counterclockwise and you'll see it really is an uppercase A.) Along the top, from left to right, are the numbers 64, 32, 16, 8, 4, 2, and 1. These are powers of 2, in descending order—binary code. At the left side of the figure you see, from top to bottom, that each line of the matrix is numbered from 1 to 5. When this character is output to the printer, it will be printed sideways, as shown, first line 1, then 2, 3, 4 and 5.

Breaking the Code

Let's examine the first line, marked 1. The first five squares are blacked in, then the last two are blank. Adding the numbers only above the black squares, you get 64 + 32 + 16 + 8 + 4 = 124. That's the number you'll find on the right side of line 1 in Fig. 14-1. In conventional 7-bit binary code, this would be 1111100. But here we are using the total decimal values, from the "most significant bit" (MSB) of 64 to the "least significant bit" (LSB) of 4 in this case. Beginning to get the idea?

Line 2 would be 0010010 in 7-bit binary code. Adding the MSB of 16 to the LSB of 2 gives 18. Similarly, line 3 of this character would be 16 + 1 = 17. Lines 4 and 5 are the same as lines 2 and 1, respectively.

Another Example

Fig. 14-2 shows an uppercase B plotted in the 7 × 5 matrix, and the binary-derived decimal numbers. Add the decimal values of each black space on a line and you'll see that they total the number on the right side of that line.

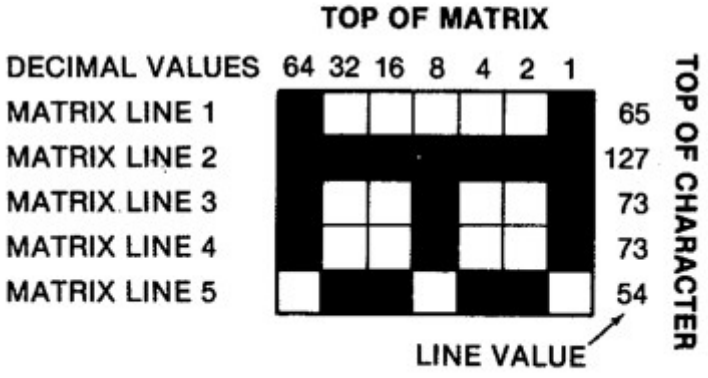


Fig. 14-2. Capital B matrix.

It takes very little imagination to see that you can plot any shape into the matrix and then derive the decimal numbers to define each line. But then what? How are these decimal numbers used?

Using the Code

The numbers you get by "adding" the black squares in the 7 × 5 character block are used in program DATA statements, as shown in

program Lines 1010 and 1020 for the letters A and B (Fig. 14-1 and Fig. 14-2). It's that simple!

To plot your own character just fill in the appropriate squares in a 7 × 5 drawing (like Fig. 14-3), then total the decimal values of the black blocks on each line. Assign any keyboard character (except a quotation mark) to call that special character for the banner message. Then use the five decimal line values in a DATA statement beginning with the keyboard character you've selected. Look at the DATA in the program for more examples.

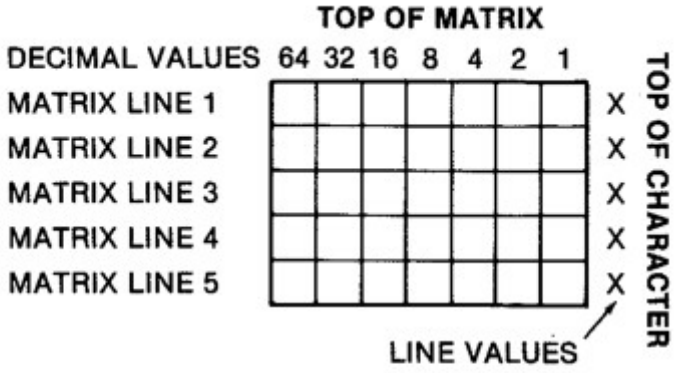


Fig. 14-3. Blank matrix.

SUBroutine ANATOMY

Now, if you're faint-hearted, stop at this point, since the rest of the explanation of this program is confusing at best. However, follow it through, and you'll be able to change the height and width of each character.

One Line at a Time

For the sake of illustration, let's assume you select the letter A as the first letter in your banner message. Notice, that's an uppercase A—use the CAPS SHIFT and A key. Looking at the DATA lines, you'll see program line 1010 is the DATA line for "A," and the numbers on that line are 124, 18, 17, 18, and 124.

Line 40 READs the A as x$, compares it with the first letter of your message (a$ in line 30), sees the "match," and goes to line 100. Line 100 sets the value of x to the number 124, the first number in the DATA statement for A, and then branches to the SUBroutine at line 200.

Line 200 subtracts 64 from x. In this example, since x is 124, then 124 − 64 = 60. Since this is greater than zero, the program says to GOTO the SUBroutine at line 500. Another SUBroutine? We're already in a SUBroutine! That's right, SUBroutines can be "nested," just like loops.

Line 500 looks very peculiar. It is an example of string concatenation, which was discussed in Chapter 6. In this case, the letter A (which is a$) is "added" to p$ four times. Since p$ was nulled in line 20, the result is simply AAAA—four As.

The RETURN at the end of line 500 sends the program back to line 200, where the value of x is now reduced by 64. This means x will be 124 minus 64, or 60. That's the new value of x as the program is sent to line 220.

Line 220 subtracts 32 and finds the remainder is greater than zero, so line 500 adds four more As to p$. This means p$ at this point is AAAAAAAA (eight As). The value of x is reduced to 28 by line 220 as the program moves on to line 240.

This process continues through lines 240, 260, 280, and the SUBroutines as the value of x is reduced to 12, then 4, then 0. Meanwhile, p$ has grown to 20 As, since it has added four As for each "iteration" (repeat) of the sequence.

Something different finally happens in line 300, however. Since the value of x has reached zero, when line 300 subtracts 2 the value of x is no longer greater than or equal to zero, so line 300 is ignored and the program moves to 310, which routes the program to line 510 (since x minus 2 is less than zero).

Shooting Blanks

Line 510, instead of adding four more characters to p$, adds four blank spaces. Aha! Now you see what's going on, don't you? When a block of the character matrix is to be filled, four of those characters are added to p$. When the block is to be "empty," then four blank spaces are added.

Line 320 is ignored (since x minus 1 is not greater than or equal to zero), but line 330 is satisfied and line 510 adds four more blanks to p$. Now p$ consists of 20 As followed by 8 blanks—a total of 28 characters.

Finally, the printer comes into the act in line 340, where p$ is LPRINTed twice, p$ is again nulled, and the program RETURNs to line 100.

The Other Character Lines

Program lines 110–140 perform exactly the same way for character matrix lines 2, 3, 4, and 5, using the DATA numbers 18, 17, 18, and 124, respectively. Fig. 14-4 shows the finished printout of a banner A.

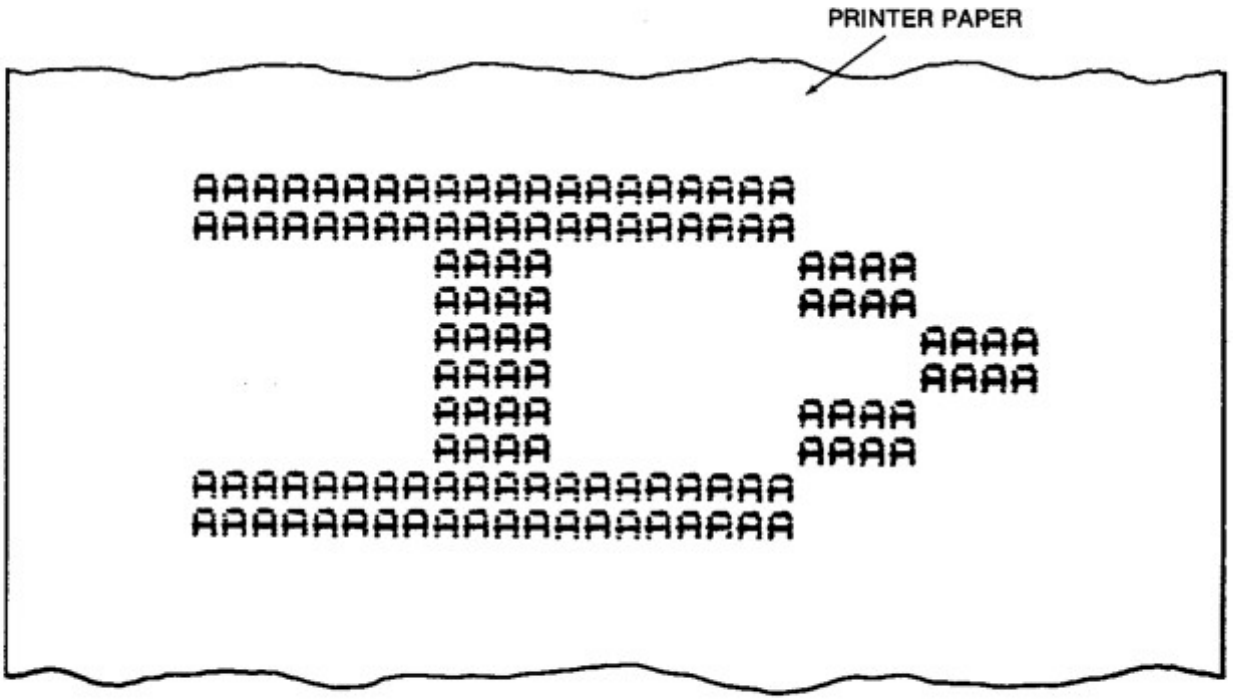


Fig. 14-4. Printed capital A.

Line 150 tells the printer to LPRINT two blank lines, which spaces between characters. The RESTORE moves the DATA pointer back to the beginning of DATA for the next character, and the NEXT I returns to line 30 to select the next character of your message. If the message is complete, the program STOPS.

Summary

In summary, lines 100–140 represent each line of the character matrix with a number from the DATA statement for that character. Lines 200–330 examine each of these five character lines in turn, breaking down the DATA number into seven blocks of four-character strings of either the keyboard character or blank spaces, thus forming a string of 28 characters for each line. This string is then printed twice, and then the next message character is processed until the banner is completed.

Fig. 14-5 is a sample banner I gave to the Chief of this village here

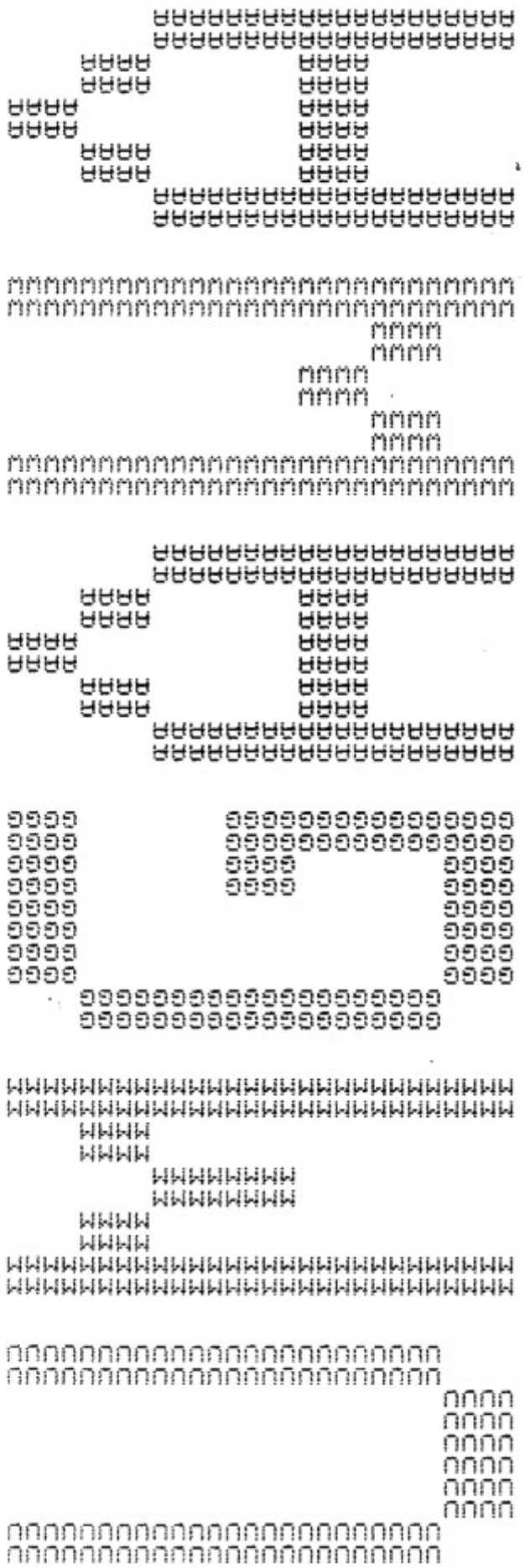


Fig. 14-5. The chief's favorite word.

in Lower Slabovia. It's his favorite expression when eating hippo hips smothered in onions.

MODIFICATIONS

If you wish to have "wider" letters, add more ":LPRINT p$" in Line 340. To add more space between characters, put additional ":LPRINT" statements on Line 150.

To make characters that are not as high, use less a$ on Line 500 and a similarly reduced number of blanks on Line 510.

Design your own lowercase characters and punctuation in a 7 × 5 matrix and then add the necessary DATA lines. Remember that each banner character, regardless of its shape, will be composed of the individual keyboard character you assign at the beginning of the DATA statement for that banner character.

QUESTIONS

1. Where is the copyright symbol on the keyboard? How do you get it into line 10?
2. When a string is "nulled," what does that mean?
3. What happens, in this program, if you put a character in your "message" that is not in DATA?
4. If your message character is E, what will the values be for each of the lines of the character matrix?
5. Why are there two LPRINTs in line 150?
6. If the value of x is 8 as line 260 is reached, does the program add As or blanks to p$?
7. How many characters are in p$ when line 340 is reached?
8. How would you make each banner character "wider"? How about "shorter" in height?
9. Can lowercase characters be added? How?

VARIABLES

a$	message character
l	loop counter
l1	character matrix line 1
l2	character matrix line 2
l3	character matrix line 3
l4	character matrix line 4
l5	character matrix line 5
m$	message string
p$	printer string
x	current value of character matrix line
x$	character from DATA

15
Collection Evaluator

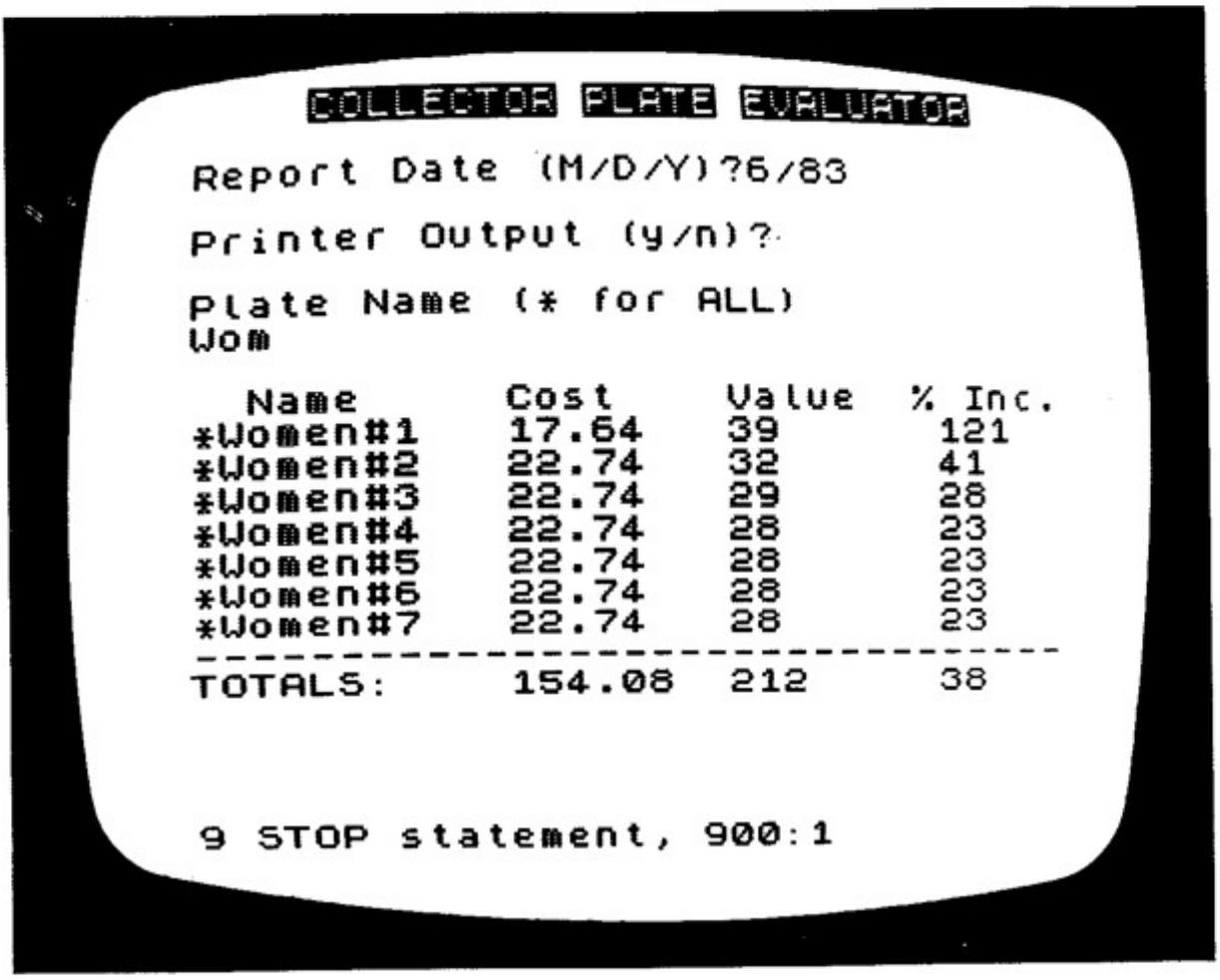


Photo 15-1. Evaluating the "Women" series of collector plates.

INTRODUCTION

If you collect coins, stamps, old bottles, plates, paintings, ceramics, or anything of value, this program will let you keep track of the original cost, the present value, and the percentage of increase or decrease in value. Here in the jungle I use it to keep track of elephant tusks, although the trading isn't too brisk. (But wait until I link all these villages together with T/S 2068s, modems and "teletusk"!)

PROGRAM DESCRIPTION

The program must be initialized with your own DATA. When the program is RUN it asks for the date and the item you wish to check. You have the option of asking for all items in DATA. Each selected item is displayed, showing its original cost, its present value, and the percentage of change in that figure. The items selected are totalled, with total cost, total value and average percentage shown. If you have a printer active, and request printer output, the screen output is summarized on the printer.

DISCUSSION

The sample program I'll describe uses collector plates offered by the Bradford Exchange in Chicago. These are plates made of various materials molded or painted with art and collected either for their artistic or investment value.

LISTing 15-1 shows the program, with Fig. 15-1 and Fig. 15-2 showing typical RUNs. I've put the plate names, original cost, and quoted value as of June, 1983 into DATA statements. Updating is just a matter of changing the last number in each DATA line. I can get the value of any series of plates, or all plates, as well as the total value and percentage change compared to the original cost.

LISTing 15-1

```
  10 REM * Collection Evaluator
Program *
  20 REM * Copyright Fred Blechm
an 1983 *
 100 PRINT "  COLLECTION EVALUATOR
R PROGRAM"
 110 PRINT : PRINT "   If you co
llect practically"
 120 PRINT "anything, this progr
am will"
 130 PRINT "determine the total
cost, the"
 140 PRINT "total present value,
 and the"
 150 PRINT "percentage increase
in value..."
 160 PRINT : PRINT "   You must
enter your own DATA"
 170 PRINT "starting at Line 200
0. Do NOT"
```

LISTing 15-1—cont.

```
 180 PRINT "delete Line 5000! Al
so, the NAME"
 190 PRINT "must not exceed 10 c
haracters..."
 200 PRINT : PRINT "   The DATA
shown in the LISTing"
 210 PRINT "represent Collector
Plates on"
 220 PRINT "the Bradford Exchang
e, and are"
 230 PRINT "to demonstrate progr
am use...."
 240 PRINT : PRINT "Press ENTER
for sample RUN...": INPUT z$
 300 DIM c(100): DIM r(100)
 400 CLS : PRINT TAB 4;"[illegible]
[illegible]"
 410 PRINT : PRINT "Report Date
(M/D/Y)?";: INPUT d$: PRINT d$
 420 PRINT : PRINT "Printer Outp
ut (y/n)?": INPUT q$
 500 PRINT : PRINT "Plate Name (
* for ALL)": INPUT n$: PRINT n$
 510 PRINT : PRINT "   Name      C
ost     Value   % Inc."
 520 IF q$="y" THEN LPRINT "Repo
rt Date:  ";d$: LPRINT : LPRINT "
  Name      Cost     Value   % Inc.
"
 600 LET x=LEN n$: LET ct=0: LET
 rt=0: LET c=1
 610 READ a$,c(c),r(c): LET y=LE
N a$
 620 IF a$="z" THEN GO TO 800
 640 FOR s=1 TO y-x+1
 650 IF n$=a$(s TO s+x-1) THEN G
O TO 700
 660 NEXT s
 670 GO TO 610
 700 BEEP .5,7: LET p=(r(c)/c(c)
)-1: LET p=INT (p*100+.5)
 710 PRINT a$;TAB 11;c(c);TAB 19
;r(c);TAB 27;p
 700>BEEP .5,7: LET p=(r(c)/c(c)
)-1: LET p=INT (p*100+.5)
 710 PRINT a$;TAB 11;c(c);TAB 19
;r(c);TAB 27;p
 720 IF q$="y" THEN LPRINT a$;TA
B 11;c(c);TAB 19;r(c);TAB 27;p
 730 LET ct=ct+c(c): LET rt=rt+r
(c): LET c=c+1
 740 GO TO 610
 800 PRINT "-------------------
-----------"
```

LISTing 15-1—cont.

```
 810 IF q$="y" THEN LPRINT "----
------------------------------"
 820 LET pt=(rt/ct)-1: LET pt=IN
T (pt*100+.5)
 830 PRINT "TOTALS:  ";TAB 11;ct;
TAB 19;rt;TAB 27;pt
 840 IF q$="y" THEN LPRINT "TOTA
LS:  ";TAB 11;ct;TAB 19;rt;TAB 27
;pt
 900 STOP
2000 DATA "*Lafay#1",14.82,48
2010 DATA "*Lafay#2",19.82,53
2020 DATA "*Lafay#3",19.82,48
2030 DATA "*Lafay#4",19.82,59
2040 DATA "*Lafay#5",19.82,67
2050 DATA "*Lafay#6",19.82,51
2100 DATA "*Noel  75",24.32,114
2110 DATA "*Noel  76",24.32,32
2120 DATA "*Noel  77",24.32,31
2130 DATA "*Noel  78",26.81,28
2140 DATA "*Noel  79",26.81,26.81
2150 DATA "*Noel  80",28.74,28.74
2200 DATA "*Women#1",17.64,39
2210 DATA "*Women#2",22.74,32
2220 DATA "*Women#3",22.74,29
2230 DATA "*Women#4",22.74,28
2240 DATA "*Women#5",22.74,28
2250 DATA "*Women#6",22.74,28
2260 DATA "*Women#7",22.74,28
2300 DATA "*Pens.Mad.",45,52
2400 DATA "*Wizard#1",19,85
2410 DATA "*Wizard#2",19,38
2418 DATA "*Wizard#2",19,38
2420 DATA "*Wizard#3",19,30
2430 DATA "*Wizard#4",19,30
2440 DATA "*Wizard#5",19,27
2450 DATA "*Wizard#6",19,20
2460 DATA "*Wizard#7",19,20
2470 DATA "*Wizard#8",24,24
2500 DATA "*Wind#1",21.5,85
2510 DATA "*Wind#2",21.5,23
2520 DATA "*Wind#3",21.5,21.5
2600 DATA "*Rockrefl",24.5,28
2610 DATA "*Rockpride",24.5,24.5
2620 DATA "*Rocktoy",14.5,185
2630 DATA "*Rockcobb",19.5,84
2640 DATA "*Rocklight",19.5,39
2650 DATA "*Rockship",19.5,19.5
5000 DATA "z",0,0
```

```
COLLECTOR EVALUATOR PROGRAM

   If you collect practically
anything, this program will
determine the total cost, the
total present value, and the
percentage increase in value...

   You must enter your own DATA
starting at Line 2000. Do NOT
delete Line 5000! Also, the NAME
must not exceed 10 characters...

   The DATA shown in the LISTing
represent Collector Plates on
the Bradford Exchange, and are
to demonstrate program use....

Press ENTER for sample RUN...

COLLECTOR PLATE EVALUATOR

Report Date (M/D/Y)?6/19/83

Printer Output (y/n)?

Plate Name (* for ALL)
Wiz

  Name       Cost      Value    % Inc.
*Wizard#1    19        85         347
*Wizard#2    19        38         100
*Wizard#2    19        38         100
*Wizard#3    19        30          58
*Wizard#4    19        30          58
*Wizard#5    19        27          42
*Wizard#6    19        20           5
*Wizard#7    19        20           5
*Wizard#8    24        24           0
-------------------------------------
TOTALS:      176       312         77
```

Fig. 15-1. Typical RUN (Wizard plates only).

I won't argue that for this small number of items the program could well be considered "overkill." However, for a large number of items in several categories, this program becomes far more practical than an adding machine, a calculator and file cards. With some re-programming, other facts about each item could certainly be added to this simple dedicated "data base" program.

```
COLLECTOR PLATE EVALUATOR
Report Date (M/D/Y)?6/19/83
Printer Output (y/n)?
Plate Name (* for ALL)
fay

  Name       Cost     Value   % Inc.
*Lafay#1    14.82     48        224
*Lafay#2    19.82     53        167
*Lafay#3    19.82     48        142
*Lafay#4    19.82     59        198
*Lafay#5    19.82     67        238
*Lafay#6    19.82     51        157
-------------------------------------
TOTALS:     113.92    326       186
```

Fig. 15-2. Lafayette plates only.

PROGRAM ANATOMY

Lines 10–240 identify the program name and author, and then fill a screen with introductory information. Generally, if there is enough memory, it's a good idea to put program instructions on the screen so the user does not need additional documentation.

DIMensioning the Arrays

When you press ENTER to continue, line 300 sets up two single-DIMension 100-position numerical arrays, array-c and array-r. In this case, "c" stands for "cost" and "r" stands for "retail value," although, of course, they could have any single-letter names. If you're rusty on arrays, refer back to Chapter 7.

Actually, we could use a two-dimensional array here as well, except we really don't need any arrays at all. They are used here merely as a convenience, instead of other variables and could just as well be called "cost" and "retail." As a matter of fact, if you want to enlarge this program beyond 100 items, you'll either have to change the DIM statement to larger numbers, or switch to plain numerical variables. Then you are only limited by computer memory.

User INPUTs

Line 400 CLears the Screen and PRINTs the title of the program, and line 410 asks for the report date. You can type and ENTER the date in any format you want, since it will be held in d$ and PRINTed now, and again later in the program, in the same form. In some programs, date entry format must be rigidly observed, since the date is actually used in some fashion for calculation within the program (such as the Bio-Graph Program of Chapter 13). In this case, the date is only used as a notation.

Line 420 skips a couple of screen lines and asks if you want output to the printer, then assigns your response to q$. There is no error-trapping here, and only a lowercase y will give you printer output later in the program. If you don't have the printer attached, or ready, and output is directed to the printer, the program just proceeds as if the printer were not being addressed. So, if you're expecting printer output and don't get it, make sure the printer is plugged in and turned on.

Finally, line 500 asks you for the plate name, and instructs you to use an asterisk (*) if you want all plates. Notice that all DATA lines have an asterisk as the first character in the string, except the last DATA line, which has only the character "z." In any case, the name you input (or the asterisk) becomes n$ and is PRINTed on the screen.

The Substring Search String

The "search" conducted within this program (lines 600–670) is very powerful, since it will identify all DATA strings that contain the "name" you are seeking. For example, if you wanted only the Wizard plates, you could use "Wiz" as the search word (with a capital W), or just "iz," or "ard" or "iza" or any other "substring." (You don't need to put in the quotes, since the prompt at the bottom of the screen does this for you.) Even a single character can be used as the search "target," but it would more than likely lead to finding some unwanted strings that contained that same letter.

The danger of the substring search is that the same substring might exist in other plate names, but its advantage is that you don't have to specify the entire name. This can be a very handy search routine to use in mailing list programs, for example, where you might just want all the names starting with the letter A.

Line 510 puts some column headings on the screen and puts the

report date and headings out to the printer if you answered "y" to the line 420 prompt.

The Search Module

A well-written program has "modules"—sections of a program that perform certain functions. Lines 600–670 in this program perform a search for the name you specify.

To illustrate how the search works, let's assume you typed "Wiz" as the name you want in response to line 500. This makes n$ = "Wiz." You want, in other words, all the plates that have "Wiz" in their name. Obviously, this will be the Wizard series. But how does the computer find them?

Line 600 initializes the search by first setting the variable x equal to the LENgth of n$. In this case ("Wiz") the LENgth is 3, so x equals 3. Then variables ct (for "cost total") and rt (for "retail total") are set to zero, and the variable c (array "counter") is set to 1.

Line 610 READs the first complete DATA line. Notice that line 610 READs three variables—string variable a$, array c(c) and array r(c)—and there are three items in each DATA line. Laying out the number of items in DATA lines to match READ statements makes it easy to write and modify a program. In this example, since the array counter c is equal to 1 at this point (line 600) the c(c) is c(1) and r(c) is r(1). Looking at the first DATA line (2000) we see that a$ will be "*Lafayette#1", c(1) is 14.82 and r(1) is 48.

Now line 610 continues to set the variable y to the LENgth of a$, which, for "*Lafayette#1" is 12 characters. Line 620 looks to see if a$ is equal to "z." If it is, that means the DATA list has been completed (since "z" is the first item in the last DATA line), and the program would go to line 800 for totals. But a$ at this point is not "z," so line 620 is ignored.

Line 640 establishes a FOR-NEXT loop, with s as the counter, to examine the string READ from DATA to see if it's one we're looking for. The maximum number of times through the loop is y−x+1—in this case 12 (the LENgth of a$ from line 610) minus 3 (the LENgth of n$ from line 600) plus 1, which equals 10. If you work this out, you find this is exactly the number of times that three consecutive characters will fit into twelve consecutive characters. In other words, the program is preparing to compare characters three at a time, starting at the left and moving to the right one character at a time. That's exactly what the next program line does.

Three at a Time

Line 650 is the "brains" of this search. It takes the current value of the loop, s, and "slices" a chunk of a$ from that character position in a$ to the character of s+x−1. With our example, x−1 is equal to 3−1 or 2. So the string slicing covers from s to s+2—a total of three characters. Isn't this ingenious? Of course, since we're just starting this loop, s (from line 640) is 1, so the "slice" of "*Lafayette#1" is from the first to the third character—"*La." This is obviously not equal to n$ (which is "Wiz"), so the program falls through to line 660, which increases the value of s to 2. Now characters 2–4 of a$ ("Laf") are compared with "Wiz." No match. This goes on until the 10 tries of the loop started in line 640 are complete. No match is found, so the program proceeds to line 670—which GOes TO line 610 for another READ.

All of these iterations happen in milliseconds (that's faster than a charging rhino!) even though it seems to take forever to explain. In this example, the program will bypass all the DATA words until "*Wizard#1" is reached. There, on the second try, line 650 finds a match! Why not the first try? Because the asterisk occupies the first character space, but it is not part of the name ("Wiz") we asked for. Also, it is important to realize that at this point the other two items READ from DATA, c(1) and r(1), have the values of 19 and 85, respectively. (See line 2400.) All previous values of C(1) and r(1) were "thrown away" since a match was not found.

The Asterisk Connection

Do you see why just typing the asterisk as your desired name makes a "hit" on every DATA item? It's the first character in every DATA line, and is matched on the first try on line 640, regardless of what characters follow.

Whenever there's a "match" on line 640, the program jumps to line 700, where several things happen. First of all, there's a short BEEP to alert you to a match. Then a new variable p (for "percentage") is set equal to the retail value divided by the cost, minus one. This is actually the percentage of change in value when comparing retail value with cost. If the result is a loss, the number is negative.

However, computers always calculate with lots of extra decimal places, so the next statement in line 700 converts the value of p to a rounded-off whole number. Line 710 displays the plate name, cost,

value, and percentage increase in value on the screen. If a printout has been asked for, line 720 does the same on the printer.

Running Totals

Line 730 does some internal bookkeeping, keeping track of running totals. Back in line 600 variables ct and rt were set to zero. In line 730, the last values of a match are added to the then current values of ct and rt to give new totals. These are kept in memory until a little later, as you'll see. Line 730 also advances the array counter by one, so the next match values will be held in the next array location. Actually, this array-storage of numbers would allow other manipulation not used in this program, but is shown as an example of how an array can be constructed to only hold selected values. Only the matched strings cause the array counter to advance, so the unmatched strings are therefore ignored.

Line 740 sends the program back to 610 for another READ and search.

Final Totals

When the program finally gets to the last DATA statement and READs a "z" for a$, line 620 jumps the program to line 800, where a dashed line is drawn across the entire screen. Line 810 does the same on the printer. Line 820 does a simple calculation to determine the overall percentage of total retail value compared to total cost, and assigns the result to variable pt (for "percentage total"). Lines 830 and 840 print the final results, and line 900 STOPs the program.

MODIFICATIONS

Since the DATA presented here is for collector plates you probably don't even own, the first "modification" is to DELETE 2000,2650 and then put in DATA statements which represent something you collect. Maybe you collect records, video tapes, jewelry, posters, or butterflies. If you can establish a cost and present value, use the program as is. With some thought you can add additional DATA, such as date acquired, date sold, buyer, etc. Be aware, however, that this is a simple data base and is not designed to be efficient when expanded too far. As the natives say out here in the jungle, "You can't make a silk umgawa out of a hippo's ear."

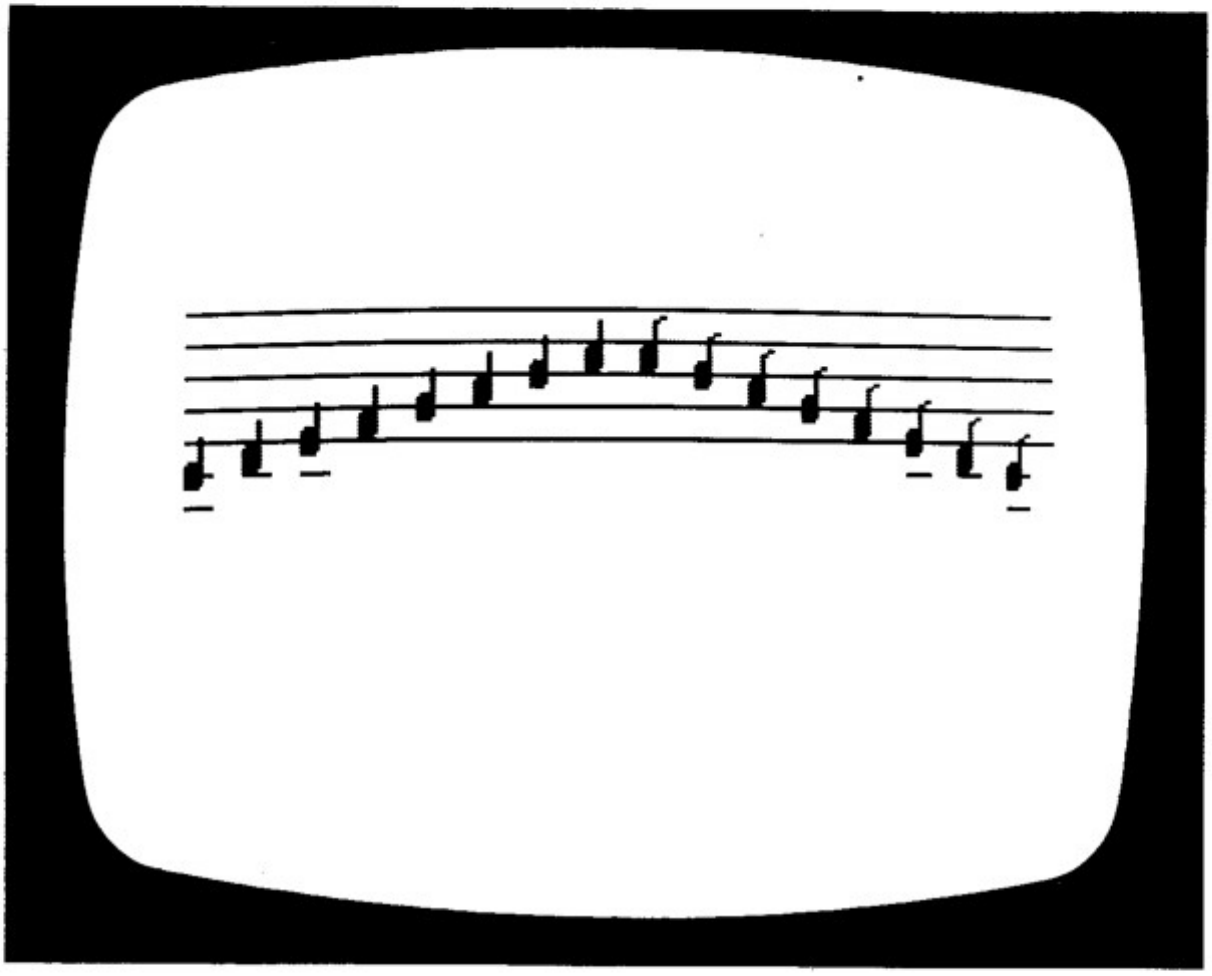
QUESTIONS

1. What general category does this program fall into?
2. Are arrays necessary in this program?
3. What happens if you respond to the question in line 420 with an uppercase Y?
4. What happens if you request printer output but the printer is not turned on?
5. If you use "Women" as your name desired, what is the value of x in line 600?
6. Why is the INTeger statement used in line 700?
7. Does the search require a complete name? What is the minimum number of characters that must be specified?
8. Can you use more than 100 items in this program without change? Why?
9. How is the percentage shown if it's a loss rather than a gain?

VARIABLES

a$	string from DATA
c	counter for arrays
c(c)	cost of DATA item
c(100)	cost-array with 100 locations
ct	cost total
d$	date
n$	name desired
p	percentage of retail to cost gain or loss
pt	percentage average for totals
q$	printer query
r(c)	retail of DATA item
r(100)	retail-array with 100 locations
rt	retail total
s	search loop counter
x	length of n$
y	length of a$
z$	dummy string to continue program

16
The Music Maker



Special graphic musical notes.

INTRODUCTION

In Chapter 10 we introduced the BEEP command for playing "music" and used a keyboard search routine which programmed some of the keys to produce standard musical tones. My own form of musical notation was introduced (Fig. 10-2). In this chapter we'll actually program the T/S 2068 to play some simple tunes, using a technique that will allow you to program your own melodies. We'll also be able to display graphic "notes" on the screen as the music plays.

PROGRAM DESCRIPTION

LISTing 16-1 is a BASIC program that plays three different tunes: "Home, Sweet Home," "Twinkle, Twinkle, Little Star," and "Three Blind Mice." You were expecting, maybe, Beethoven's Fifth Symphony?

LISTing 16-1

```
  10 REM * Sounds of Music? *
  20 REM * (C) Copyright Fred Bl
echman 1983 *
 100 READ d,p: LET d=d/16
 110 IF p=999 THEN GO TO 1000
 120 BEEP d,p: GO TO 100
 200 REM * Home, Sweet Home *
 210 DATA 2,0,2,2,8,4,4,5,4,9,8,
7,4,4
 220 DATA 4,7,6,5,2,4,4,5,4,2,8,
4,4,69
 230 DATA 2,0,2,2,8,4,4,5,4,9,8,
7,4,4
 240 DATA 4,7,6,5,2,4,4,5,4,2,8,
0,16,69
 300 REM * Twinkle, Twinkle Litt
le Star *
 310 DATA 2,0,2,0,2,7,2,7,2,9,2,
9,4,7,2,5,2,5,2,4,2,4,2,2,2,2,4,
0
 320 DATA 2,0,2,0,2,7,2,7,2,9,2,
9,4,7,2,5,2,5,2,4,2,4,2,2,2,2,4,
0
 330 DATA 2,7,2,7,2,5,2,5,2,4,2,
4,4,2,2,7,2,7,2,5,2,5,2,4,2,4,4,
2
 340 DATA 2,0,2,0,2,7,2,7,2,9,2,
9,4,7,2,5,2,5,2,4,2,4,2,2,2,2,4,
0,16,69
 400 REM * Three Blind Mice *
 410 DATA 4,4,4,2,4,0,4,69,4,4,4
,2,4,0,4,69
 420 DATA 4,7,2,5,2,5,4,4,4,69,4
,7,2,5,2,5,4,4,4,69
 430 DATA 2,7,1,12,1,12,1,11,1,9
,1,11,2,12,1,12,2,7
 440 DATA 2,7,1,12,1,12,1,11,1,9
,1,11,2,12,1,12,2,7
 450 DATA 2,7,1,12,1,12,1,11,1,9
,1,11,2,12,1,12,2,7
 460 DATA 2,5,4,4,4,2,4,0,16,69
 999 DATA 999,999
1000 RESTORE : GO TO 100
```

To help you follow these programs (and to write your own musical adaptations), Fig. 16-1, Fig. 16-2, and Fig. 16-3 show my musical notation for these songs. On the left side of the musical staff you'll find the conventional designations (C–E) of the note positions, as well as the number used to produce that note on the 2068. The numbers below each note specify the relative duration and note value. For example, 2,0 means 2 beats of Middle C. How long is "2 beats"? Would you accept "twice as long as one beat"? Actually, the beat duration is set in the program, as you'll see.

DISCUSSION

By now you've discovered that "playing" the keyboard of your T/S 2068 with the BASIC program in Chapter 10 is less than perfect. Actually, with assembly language you could write a much more flexible program, but assembly language will not be introduced in this book. It will be covered in Jeffrey Mazur's *T/S 2068 Intermediate/Advanced Guide.*

Meanwhile, let's see how we can program the T/S 2068 to play tunes like the player pianos designed many years ago. They used punched holes in rolls of paper to "program" the notes. Instead, we'll use the BASIC computer language and save the roll paper for our computer printers!

PROGRAM ANATOMY

At first glance, this looks like a long and relatively complex program. Actually, the program consists of only four functional lines and a bunch of DATA. Line 100 READs two pieces of DATA (starting at line 210, the first DATA in the program) and assigns the first value to variable d (for "duration") and the second to variable p (for "pitch"). I should have used "f" for "frequency" or "t" for "tone" to be consistent with what I've been using—but the computer doesn't care.

Tempo Control

Next, line 100 sets the value of d to equal d/16. Why? So that we could use nice, round numbers in the DATA for duration, and yet not make the BEEPs too long. In other words, the entire tempo of the tune is controlled at one time by this little ploy. To play faster, divide

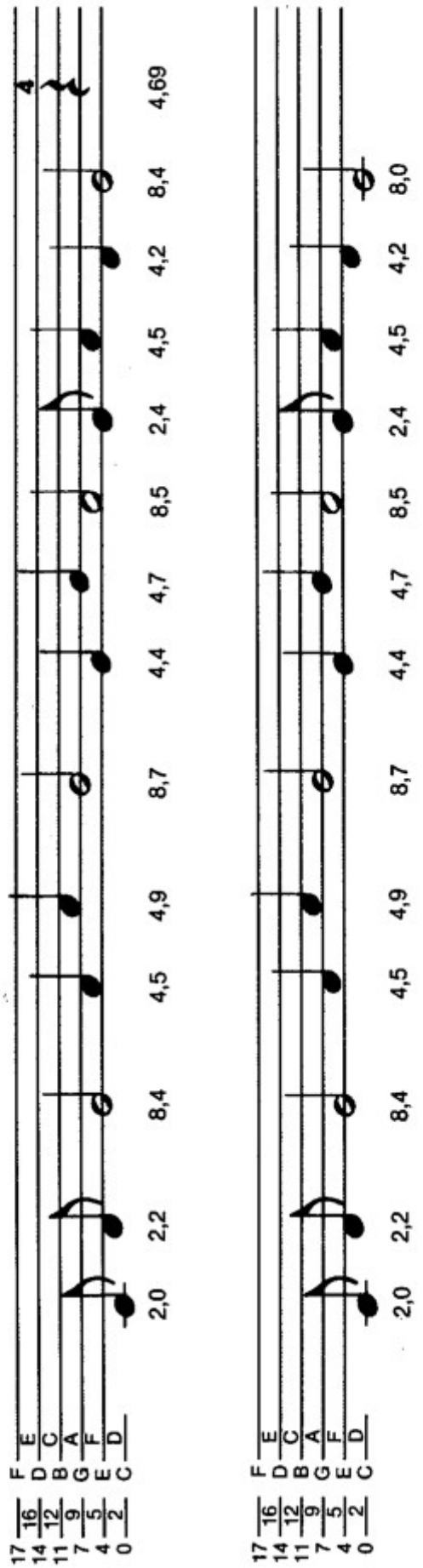


Fig. 16-1. "Home, Sweet Home."

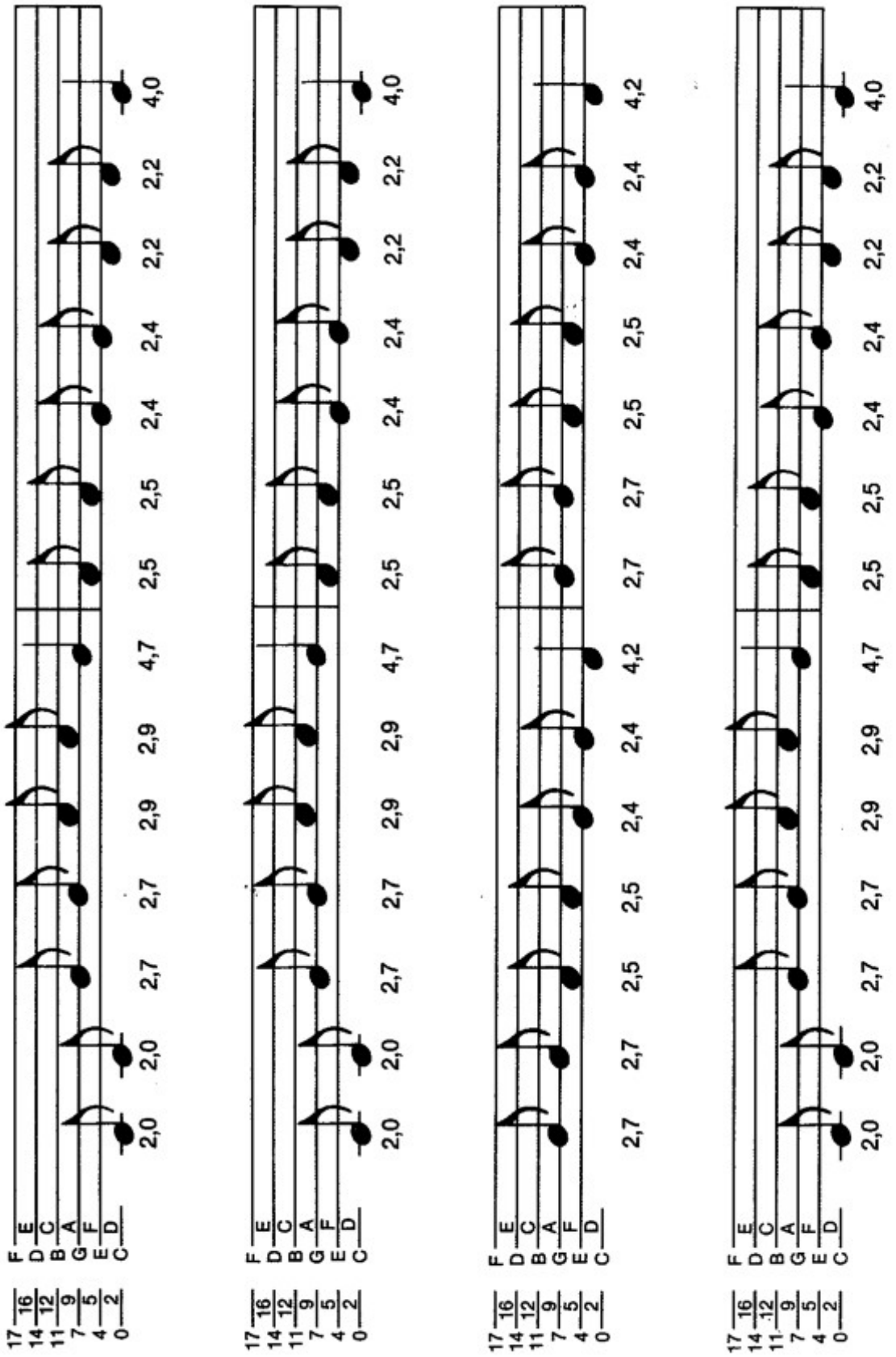


Fig. 16-2. "Twinkle, Twinkle, Little Star."

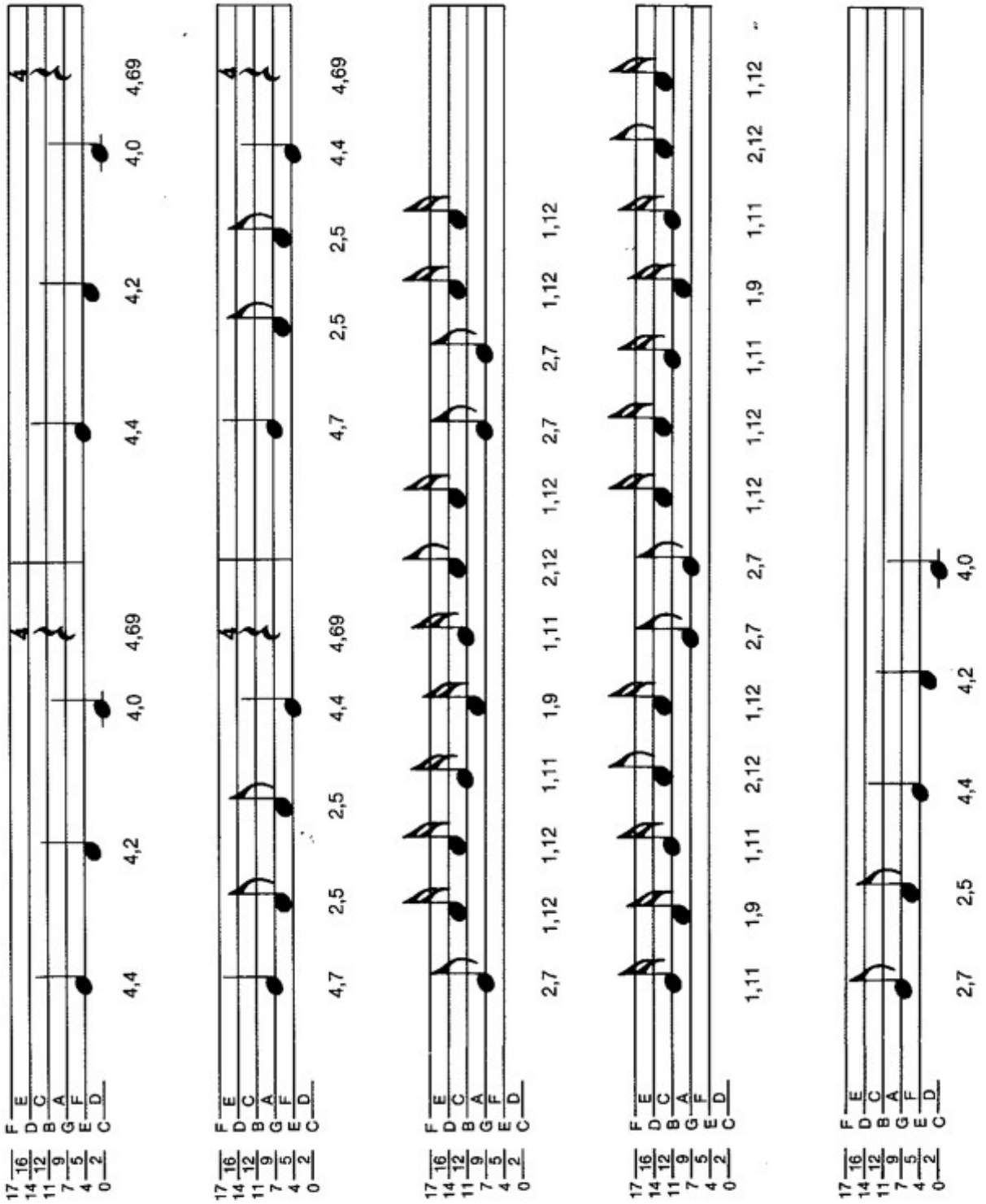


Fig. 16-3. "Three Blind Mice."

d by a larger number; for slower playing, divide d by a smaller number.

Line 110 looks to see if the end of DATA has been reached. See line 999? That's what line 110 is looking for. If the end of DATA *is* reached, then line 1000 RESTOREs the DATA and goes back to line 100 for another run-through. How do you stop the program? Use the BREAK key or the SPACE key with CAPS SHIFT.

Jeepers Beepers!

Now we come to the line that does the work. Line 120 BEEPs for d seconds (remember, d is now the original DATA d divided by 16) at musical tone p. Look at the first two DATA items: 2,0. That means the first time through line 100, d will be set to 2 and p will be set to 0. Then d is divided by 16 to equal $\frac{2}{16}$, which equals $\frac{1}{8}$, which in turn equals .12. Since the 0 for p represents middle-C on the T/S 2068, the computer will generate a middle-C for .12 second. Then it goes back to line 100 for another READ. Get the idea? If not, put down your rifle, sit down by the side of the trail, take a swig from your canteen, relax, and read this paragraph again and again (GOTO "Now we come to the line . . .").

All makes sense until we come to the end of line 240. 69? Huh? What tone is that? Well, it's a tone that's almost inaudible, and the highest number you can specify for a tone. Since you don't hear it, it acts as a "pause" or "rest."

The remainder of the program plays the other tunes and then repeats. To change the speed, change the number at the end of line 100, as already described.

For Patriots Only!

In Chapter 9, "Grappling With Graphics," we described a program for drawing the American Flag, in color, and it ended with "America, The Beautiful" just below the flag. Why not have the T/S 2068 play the melody right after drawing the flag? All you have to do is enter LISTing 16-2, which programs the computer for the music shown in Fig. 16-4. Notice that the line numbers have purposely been made high so that they won't conflict with the Flag Program. In fact, you can SAVE this program on tape just for the music, and then MERGE it to the flag program (using the MERGE command described in Chapter 4). The PAUSE 120 in line 2400 simply delays the music replay for about two seconds.

LISTing 16-2

```
2000 REM * America, The Beautifu
l! *
2010 REM * (C) Copyright Fred Bl
echman 1983 *
2100 READ d,p: LET d=d/16
2105 IF p=999 THEN GO TO 2400
2110 BEEP d,p: GO TO 2100
2200 DATA 4,7,8,7,2,4,4,4,4,7,8,
7,2,2,4,2,4,4,4,5,4,7,4,9,4,11,8
,7,8,69
2210 DATA 4,7,8,7,2,4,4,4,4,7,8,
7,2,2,4,2,4,14,4,13,4,14,4,16,4,
9,8,14,8,69
2220 DATA 4,7,8,16,2,16,4,14,4,1
2,8,12,2,11,4,11,4,12,4,14,4,11,
4,9,4,7,8,12,8,69
2230 DATA 4,12,8,12,2,9,4,9,4,12
,8,12,2,7,4,7,4,7,4,9,4,12,4,7,4
,14,16,12
2300 DATA 999,999
2400 RESTORE : PAUSE 120: GO TO
2000
```

Rolling Your Own

Obviously, you can program your own music fairly easily if you have the music, or can play it on a piano and make a record of the notes and timing. Then, using my notation or whatever is most comfortable to you, put the tone on a musical staff, note the appropriate duration and tone numbers, and then put them in DATA statements. Irving Berlin, watch out!

Sight and Sound

What's this? Did I hear you say this is not challenging enough for you? Then how about having the notes appear on a musical staff on the screen as they are being played? You can do this by using the graphic musical note combinations described in Chapter 9 and Appendix C.

Look at Fig. 16-5 and you'll see an example of the traditional do-re-me musical scale—first upscale at one speed, and then downscale at twice the speed. (It's always tougher going uphill than downhill, isn't it?)

LISTing 16-3A performs this by putting a graphic note on the screen, playing the note, then doing the next one. The programming

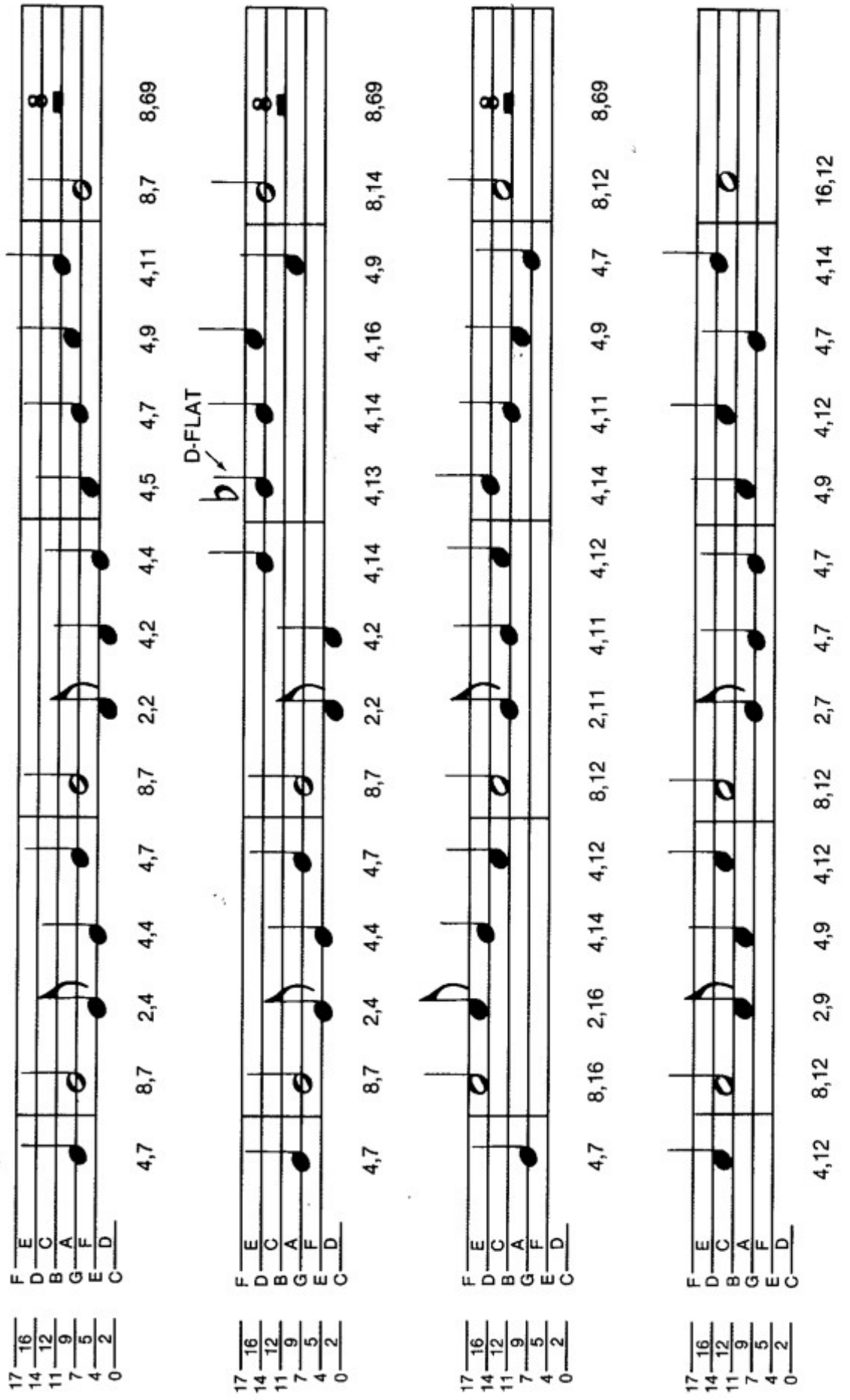


Fig. 16-4. "America, the Beautiful."

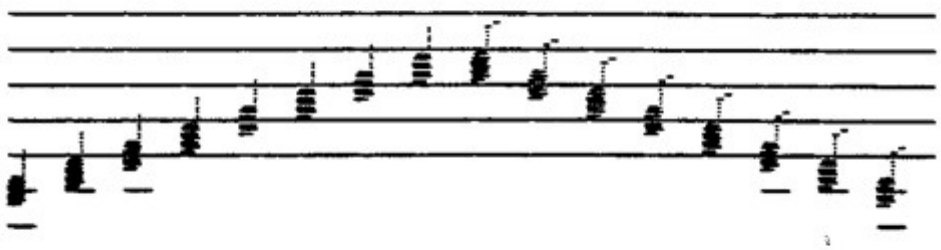


Fig. 16-5. Screen display of "Sights and Sounds."

of these graphic notes (lines 200–810) was described in Chapter 9, so we won't go over that again here.

Lines 1000–1500 create the screen display and BEEP each note after it is put on the musical staff using PRINT AT statements for positioning.

The only tricky thing when entering this program from the keyboard is to be sure that the letters in quotation marks on Lines 1100–1420 in LISTing 16-3A are entered with the GRAPHICS G-cursor on the screen (SYMBL SHIFT 9). Listing 16-3B shows lines 1000–1500 after RUN has POKEd the special characters into the specified character locations. These special graphics will remain programmed into those character locations until they are intentionally changed, or the computer is turned off.

LISTing 16-3A

```
 100 REM * Sights and Sounds *
 110 REM * (C) Fred Blechman 198
3 *
 200 FOR n=0 TO 7: READ p: POKE
USR "c"+n,p: NEXT n
 210 DATA BIN 01111000,BIN 11111
000,BIN 11111000,BIN 11111000,BI
N 11111000,BIN 11111000,BIN 1111
0000,BIN 11111111
 300 FOR n=0 TO 7: READ p: POKE
USR "d"+n,p: NEXT n
 310 DATA BIN 00000000,BIN 00000
000,BIN 00001000,BIN 00001000,BI
N 00001000,BIN 00001000,BIN 0000
1000,BIN 11111111
 400 FOR n=0 TO 7: READ p: POKE
USR "e"+n,p: NEXT n
 410 DATA BIN 00000000,BIN 00000
011,BIN 00001100,BIN 00001000,BI
N 00001000,BIN 00001000,BIN 0000
1000,BIN 11111111
 500 FOR n=0 TO 7: READ p: POKE
USR "j"+n,p: NEXT n
 510 DATA BIN 11111000,BIN 11111
000,BIN 11110000,BIN 00000000,BI
```

LISTing 16-3A—cont.

```
N 00000000,BIN 00000000,BIN 0000
0000,BIN 11111111
 600 FOR n=0 TO 7: READ p: POKE
USR "l"+n,p: NEXT n
 610 DATA BIN 00000000,BIN 00000
000,BIN 00000000,BIN 00000000,BI
N 00000000,BIN 00000000,BIN 0000
1000,BIN 11111111
 700 FOR n=0 TO 7: READ p: POKE
USR "m"+n,p: NEXT n
 710 DATA BIN 00001000,BIN 00001
000,BIN 00001000,BIN 00001000,BI
N 01111000,BIN 11111000,BIN 1111
1000,BIN 11111111
 800 FOR n=0 TO 7: READ p: POKE
USR "o"+n,p: NEXT n
 810 DATA BIN 00000000,BIN 00000
000,BIN 00000000,BIN 00000000,BI
N 00000000,BIN 00000011,BIN 0000
1100,BIN 11111111
1000>FOR x=6 TO 10
1010 PRINT AT x,0;"________________
________________"
1020 NEXT x
1100 PRINT AT 10,0;"L": PRINT AT
 11,0;"M": PRINT AT 12,0;"J"
1110 BEEP .5,0
1120 PRINT AT 10,2;"D": PRINT AT
 11,2;"C"
1130 BEEP .5,2
1140 PRINT AT 9,4;"L": PRINT AT
10,4;"M": PRINT AT 11,4;"J"
1150 BEEP .5,4
1160 PRINT AT 9,6;"D": PRINT AT
10,6;"C"
1170 BEEP .5,5
1180 PRINT AT 8,8;"L": PRINT AT
9,8;"M": PRINT AT 10,8;"J"
1190 BEEP .5,7
1200 PRINT AT 8,10;"D": PRINT AT
 9,10;"C"
1210 BEEP .5,9
1220 PRINT AT 7,12;"L": PRINT AT
 8,12;"M": PRINT AT 9,12;"J"
1230 BEEP .5,11
1240 PRINT AT 7,14;"D": PRINT AT
 8,14;"C"
1250 BEEP .5,12
1260 PRINT AT 7,16;"E": PRINT AT
 8,16;"C"
1270 BEEP .25,12
1280 PRINT AT 7,18;"O": PRINT AT
 8,18;"M": PRINT AT 9,18;"J"
1290 BEEP .25,11
```

LISTing 16-3A—cont.

```
1300 PRINT AT 8,20;"E": PRINT AT
 9,20;"C"
1310 BEEP .25,9
1320 PRINT AT 8,22;"O": PRINT AT
 9,22;"M": PRINT AT 10,22;"J"
1330 BEEP .25,7
1340 PRINT AT 9,24;"E": PRINT AT
 10,24;"C"
1360 BEEP .25,5
1380 PRINT AT 9,26;"O": PRINT AT
 10,26;"M": PRINT AT 11,26;"J"
1390 BEEP .25,4
1400 PRINT AT 10,28;"E": PRINT A
T 11,28;"C"
1410 BEEP .25,2
1420 PRINT AT 10,30;"O": PRINT A
T 11,30;"M": PRINT AT 12,30;"J"
1430 BEEP .25,0
1500 BEEP 2,69
1510 CLS : GO TO 1000
```

LISTing 16-3B

```
1000>FOR x=6 TO 10
1010 PRINT AT x,0;"________________
________________________"
1020 NEXT x
1100 PRINT AT 10,0;"▗": PRINT AT
 11,0;"▟": PRINT AT 12,0;"▀"
1110 BEEP .5,0
1120 PRINT AT 10,2;"▙": PRINT AT
 11,2;"▌"
1130 BEEP .5,2
1140 PRINT AT 9,4;"▗": PRINT AT
10,4;"▟": PRINT AT 11,4;"▀"
1150 BEEP .5,4
1160 PRINT AT 9,6;"▙": PRINT AT
10,6;"▌"
1170 BEEP .5,5
1180 PRINT AT 8,8;"▗": PRINT AT
9,8;"▟": PRINT AT 10,8;"▀"
1190 BEEP .5,7
1200 PRINT AT 8,10;"▙": PRINT AT
 9,10;"▌"
1210 BEEP .5,9
1220 PRINT AT 7,12;"▗": PRINT AT
 8,12;"▟": PRINT AT 9,12;"▀"
1230 BEEP .5,11
1240 PRINT AT 7,14;"▙": PRINT AT
 8,14;"▌"
1250 BEEP .5,12
1260 PRINT AT 7,16;"▛": PRINT AT
 8,16;"▌"
```

LISTing 16-3B—cont.

```
1270 BEEP .25,12
1280 PRINT AT 7,18;"▗": PRINT AT
 8,18;"▌": PRINT AT 9,18;"▀"
1290 BEEP .25,11
1300 PRINT AT 8,20;"▛": PRINT AT
 9,20;"▌"
1310 BEEP .25,9
1320 PRINT AT 8,22;"▗": PRINT AT
 9,22;"▌": PRINT AT 10,22;"▀"
1330 BEEP .25,7
1340 PRINT AT 9,24;"▛": PRINT AT
 10,24;"▌"
1360 BEEP .25,5
1380 PRINT AT 9,26;"▗": PRINT AT
 10,26;"▌": PRINT AT 11,26;"▀"
1390 BEEP .25,4
1400 PRINT AT 10,28;"▛": PRINT A
T 11,28;"▌"
1410 BEEP .25,2
1420 PRINT AT 10,30;"▗": PRINT A
T 11,30;"▌": PRINT AT 12,30;"▀"
1430 BEEP .25,0
1500 BEEP 2,69
1510 CLS : GO TO 1000
```

MODIFICATIONS

With a little imagination, you can now see how to play a tune and make the notes appear on screen at the proper position on the staff. This can be a really challenging project, but by combining all the musical note graphic characters developed in Chapter 9 (and shown in Appendix C), and using the music and programs in this chapter, you should be able to make "Home, Sweet Home" and the others into sight-and-sound programs. This can also lead to musical education programs—or perhaps a computer sing-a-long with lyrics and a bouncing-ball cursor in step with the timing of each note (as old-timers will remember from the days they did this in movie sing-a-longs).

QUESTIONS

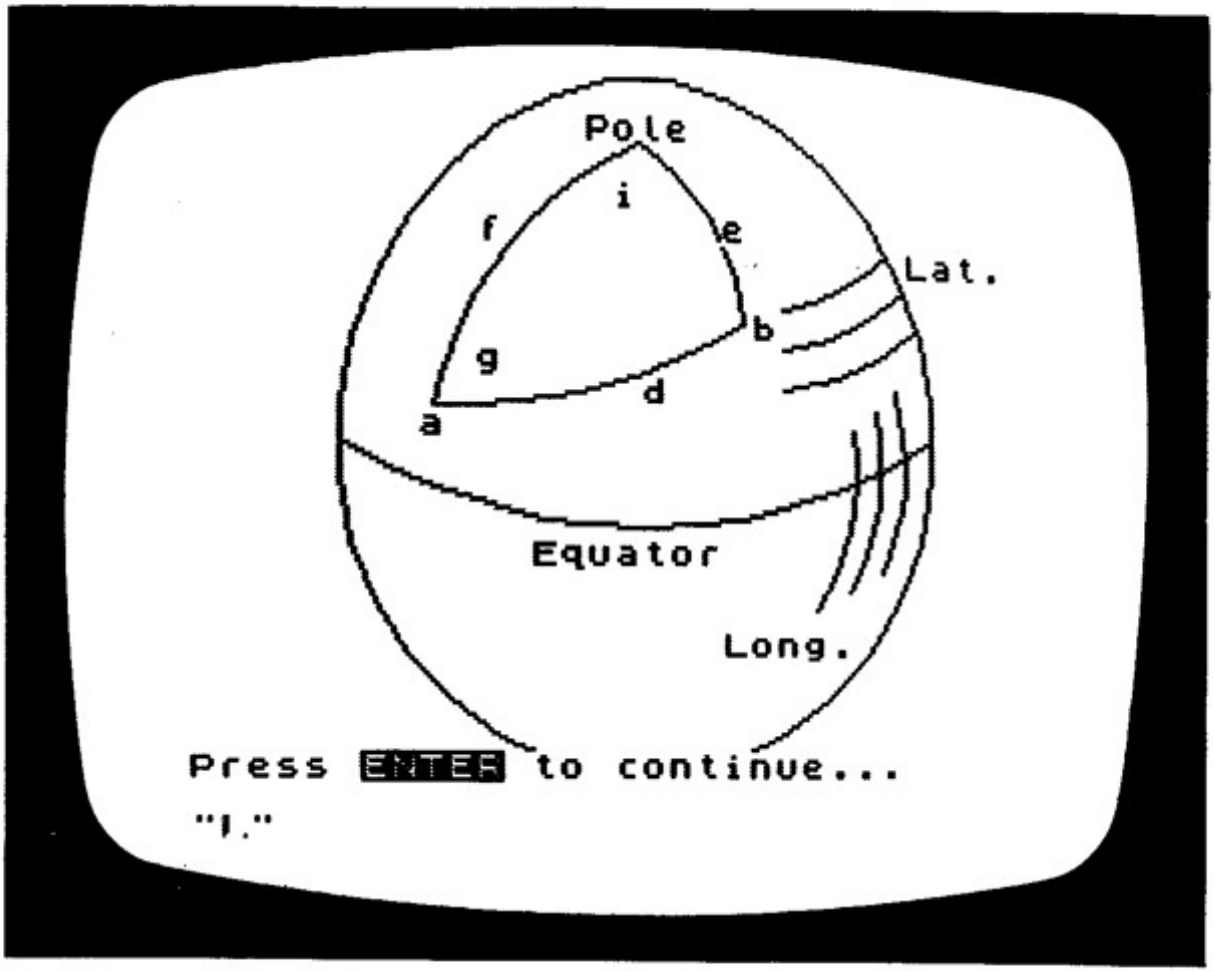
1. What does RESTORE do when used with READ and DATA?
2. What does the notation 2,0 under the first note in "Home, Sweet Home" (Fig. 16-1) mean?
3. How would you make the tunes (LISTing 16-1) play more slowly? Faster?
4. What would happen if line 999 were deleted from this program?
5. What BEEP frequency designator is used for a pause? Why?

6. Rewrite LISTing 16-3 to use READ, DATA, and RESTORE for just the BEEP parts.
7. Write your own original tune—like "Four Blind Mice."

VARIABLES

d BEEP duration
p BEEP pitch
n BIN POKE counter
x screen line number

17
Long-Distance Navigator



The navigator program at work.

INTRODUCTION

This program allows you to determine the shortest distance and direction between virtually any two points on earth, given the location of each. This is of particular value to aircraft, long-distance boating, and amateur radio ("ham") operators, as we'll explain later.

The program uses spherical trigonometry to calculate the lengths and angles of a spherical triangle between the starting point, the destination, and the North Pole. This is a fairly sophisticated mathematical use of the computer, but requires no knowledge of math to use. It will be explained in detail, however, for those that wish the "mental exercise" and mathematical verification.

PROGRAM DESCRIPTION

The program shown in LISTing 17-1, with a typical RUN shown in Fig. 17-1, starts with instructions on the screen, followed by a graphic drawing of the earth which defines the terms used. A typical spherical triangle is drawn and labeled, for those that wish to follow the mathematics of the program.

LISTing 17-1

```
  10 REM * Long-Distance Navigat
or Program *
  20 REM * Copyright Fred Blechm
an 1983 *
  30 DIM a(11)
 100 PRINT "    LONG-DISTANCE NAV
IGATOR"
 110 PRINT : PRINT "   To determ
ine the shortest"
 120 PRINT "distance from Point
A (Origin)"
 130 PRINT "to Point B (Destinat
ion), and"
 140 PRINT "the direction from A
 to B, just"
 150 PRINT "enter the latitude a
nd longitude"
 160 PRINT "of each point in dec
imal degrees"
 170 PRINT "as found from a map,
 globe, or"
 180 PRINT "encyclopedia."
 200 PRINT : PRINT "   The drawi
ng that follows     shows the sp
herical triangle    involved in
this calculation.   It also defi
nes latitude and    longitude li
nes, and all the    angles."
 210 GO SUB 3000
 220 GO SUB 2000
 300 CLS : PRINT "Origin? ";: IN
PUT "Origin?(12 Char.max.) ";a$:
 PRINT a$
 310 PRINT "Destination? ";: INP
UT "Destination?(12 char.max.) "
;b$: PRINT b$
 320 REM * Triangle Direction *
 330 PRINT : PRINT "Is ";b$;" Ea
st or"
 340 PRINT "West of ";a$;" ? ";:
 INPUT "E or W?";p$: PRINT p$
 350 REM *Input Locations *
 360 PRINT : PRINT a$;" Latitude
 ";: INPUT "Latitude? ";a(1): P
```

LISTing 17-1—cont.

```
RINT a(1);" ";
 370 INPUT "North or South? ";j$
: PRINT j$
 380 PRINT : PRINT a$;" Longitud
e: ";: INPUT "Longitude? ";a(2):
 PRINT a(2);" ";
 390 INPUT "East or West? ";k$:
PRINT k$
 400 PRINT : PRINT b$;" Latitude
: ";: INPUT "Latitude? ";a(3): P
RINT a(3);" ";
 410 INPUT "North or South? ";l$
: PRINT l$
 420 PRINT : PRINT b$;" Longitud
e: ";: INPUT "Longitude? ";a(4):
 PRINT a(4);" ";
 430 INPUT "East or West? ";m$:
PRINT m$
 440 FOR x=1 TO 100: NEXT x
 500>CLS : LET f=0: IF p$="e" OR
 p$="E" THEN LET f=1: GO TO 520
 510 PRINT b$;TAB 20;a$: GO TO 5
30
 520 PRINT a$;TAB 20;b$
 530 PRINT "  *
          *"
 540 PLOT 16,163: DRAW 208,0
 550 IF f=1 THEN PRINT a(1);" ";
j$;" ";a(2);" ";k$;TAB 20;a(3);"
 ";l$;" ";a(4);" ";m$: GO TO 570
 560 PRINT a(3);" ";l$;" ";a(4);
" ";m$;TAB 20;a(1);" ";j$;" ";a(
2);" ";k$
 600 REM * Determine f & e in de
grees *
 610 LET f=90-a(1)
 620 IF j$="s" THEN LET f=f+2*a(
1)
 630 LET e=90-a(3)
 640 IF l$="s" THEN LET e=e+2*a(
3) -
 650 REM * Determine i in degree
s *
 660 LET i=0: IF k$="w" AND m$="
w" THEN LET i=a(4)-a(2)
 670 IF k$="e" AND m$="e" THEN L
ET i=a(4)-a(2)
 680 IF i<0 THEN LET i=-i
 690 IF k$="e" AND m$="w" THEN L
ET i=a(4)+a(2)
 700 IF k$="w" AND m$="e" THEN L
ET i=a(4)+a(2)
 710 IF i>180 THEN LET i=360-i
 720 IF i<0 THEN LET i=-i
 800 PRINT : PRINT "North Polar
```

LISTing 17-1—cont.

```
Angles:"
 810 PRINT " To ";a$;": ";f;" de
grees"
 820 PRINT " To ";b$;": ";e;" de
grees"
 830 PRINT " Between points: ";i
;" degrees"
 900 REM * Find arc d in degrees
 *
 910 REM * COS d=COS e*COS f+SIN
 e*SIN f*COS i *
 920 LET j=.0174533: REM * Conve
rts from degrees to radians *
 930 LET a(5)=COS (e*j)
 940 LET a(6)=COS (f*j)
 950 LET a(7)=SIN (e*j)
 960 LET a(8)=SIN (f*j)
 970 LET a(9)=COS (i*j)
 980 LET d=(a(5)*a(6)+a(7)*a(8)*
a(9))
 990 LET y=ACS d
1000 LET y=y*180/PI: REM * Conve
rts radians to degrees *
1010 LET a(10)=y
1100 REM * Convert arc d to mile
s *
1110 REM * 1 degree=60 nautical
miles on a Great Circle *
1120 PRINT : PRINT "Nautical Mil
es: ";INT (y*60*100+.5)/100
1130 LET y=y*60*1.1515: LET y=IN
T (y*100+.5)/100: REM * Statute
Mile=1.1515*Nautical Mile *
1140 PRINT "Statute Miles: ";y
1200 REM * Determine Polar Angle
 g in degrees *
1210 REM * COS g=(COS e-COS d*CO
S f)/(SIN d*SIN f) *
1220 LET a(11)=SIN (a(10)*j)
1230 LET g=(a(5)-(d*a(6)))/(a(11
)*a(8))
1240 LET y=ACS g
1250 LET y=y*180/PI: LET y=INT (
y*100+.5)/100
1260 IF p$="W" THEN LET y=360-y
1270 PRINT : PRINT "Direction: "
;y;" degrees."
1999 STOP
2000 CLS : CIRCLE 128,88,87
2010 PLOT 42,88
2020 DRAW 173,0,1
2030 PLOT 128,160
2040 DRAW -60,-64,1
2050 DRAW 90,20,.5
2060 DRAW -30,44,1
```

LISTing 17-1—cont.

```
2100 PLOT 200,55: DRAW 4,45,.8
2110 PLOT 190,50: DRAW 8,45,.8
2120 PLOT 180,45: DRAW 11,45,.8
2200 PLOT 170,100: DRAW 40,15,.5
2210 PLOT 170,110: DRAW 37,15,.5
2220 PLOT 170,120: DRAW 32,14,.5
2300 PRINT AT 1,14;"Pole"
2310 PRINT AT 10,8;"a"
2320 PRINT AT 7,20;"b"
2330 PRINT AT 8,10;"g"
2340 PRINT AT 4,10;"f"
2350 PRINT AT 4,19;"e"
2360 PRINT AT 3,15;"i"
2370 PRINT AT 9,16;"d"
2380 PRINT AT 5,26;"Lat."
2390 PRINT AT 17,19;"Long."
2400 PRINT AT 14,12;"Equator"
2410 PLOT 127,160: PLOT 128,160
2420 PRINT AT 20,0;
3000 PRINT : PRINT "Press ENTER
to continue...": INPUT z$
3010 RETURN
```

Next you're asked for the names of the starting point ("origin") and destination, and their latitudes and longitudes. You must also specify if the destination is East or West of the starting point.

The program summarizes your input information with a simple drawing, then quickly gives you the angles of each point from the North Pole, the nautical and statute miles between the two points,

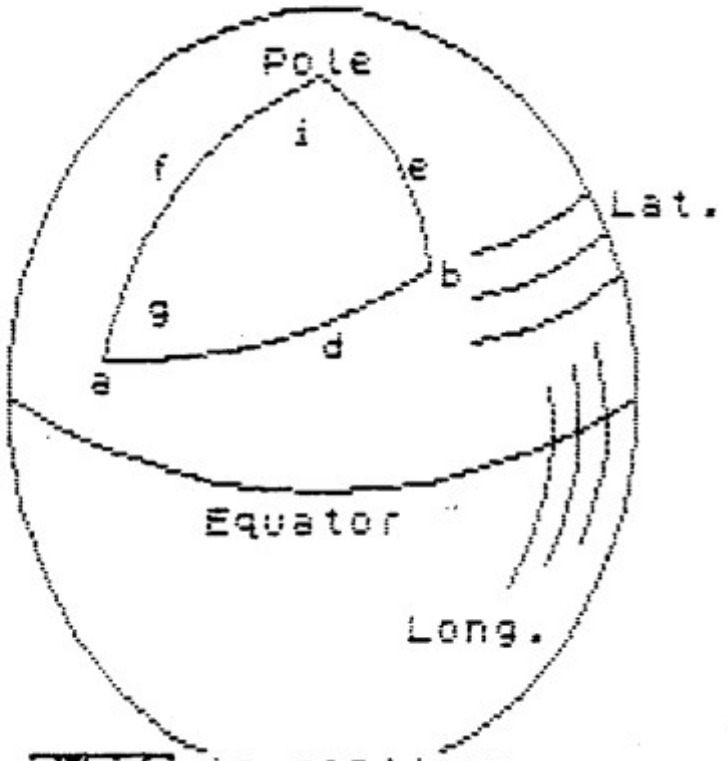
```
LONG-DISTENCE NAVIGATOR

   To determine the shortest
distance from Point A (Origin)
to Point B (Destination), and
the direction from A to B, just
enter the latitude and longitude
of each point in decimal degrees
as found from a map, globe, or
encyclopedia.

   The drawing that follows
shows the spherical triangle
involved in this calculation.
It also defines latitude and
longitude lines, and all the
angles.

Press ENTER to continue...
```

Fig. 17-1. Navigator program RUN.




```
Press ENTER to continue...

Origin? Los Angeles
Destination? Moscow

Is Moscow East or
West of Los Angeles ? e

Los Angeles Latitude: 34 n

Los Angeles Longitude: 118.5 w

Moscow Latitude: 55 n

Moscow Longitude: 37 e

Los Angeles                Moscow
  *------------------------------*
34 n 118.5 w               55 n 37 e

North Polar Angles:
 To Los Angeles: 56 degrees
 To Moscow: 35 degrees
 Between points: 155.5 degrees

Nautical Miles: 5312.8
Statute Miles: 6117.69

Direction: 13.76 degrees.
```

Fig. 17-1—cont.

and the direction in degrees, measured clockwise from the North Pole.

DISCUSSION

While most readers will not find direct use for this program, it does illustrate the sophistication that can be accomplished with a plain ole' under-$200 T/S 2068. The graphic capabilities and trig functions are liberally exercised, and the techniques used can provide examples for programs of your own.

Airplane Drivers, Sailors, and Hams

If you have an airplane, do any long distance sailing or boating, operate DX (long-distance) on your ham radio, or do much short-wave listening, this program will be of very definite value. Why? Because maps are notoriously inaccurate for long distances.

Pilots and sailors can use this program to accurately determine the distance and direction from where you are to any point on Earth. Hams and SWLs (short-wave listeners) can use this program to determine the proper direction to aim their antenna for maximum signal strength.

Many charts are available that give the distance and direction between major cities, but not from where you are to where you want to go, unless your origin and destination are major cities. With this program, you can "start" and "stop" anywhere!

Maps Are Wrong!

When I went through navigation courses as part of my flight training to become a Naval Aviator, I became aware of the compromises made in trying to reduce a spherical surface (the Earth) to a flat sheet of paper (a map). It just can't be done accurately for long distances. If the angles are correct, then the distances are distorted, and vice versa.

Probably the most common map of the world, or of large areas of the earth, is a Mercator Projection, where all the latitude (horizontal) and longitude (vertical) lines are straight and perpendicular to each other. While this is easy to read, it is notoriously inaccurate for navigational purposes. Although the old expression that "the shortest

distance between two points is a straight line" is certainly true in spatial relations, it is totally false on most maps!

Down to Earth

In the case of a spherical solid object, such as the Earth, the shortest distance along the surface is called "a great circle," and is the intersection of the surface with a flat plane that passes through the center of the sphere. You can estimate a great circle by using a globe and a string placed between the origin and the destination, but the accuracy is limited by the size of the globe. This program is accurate to a small fraction of 1 percent!

PROGRAM ANATOMY

The program is identified in lines 10 and 20, and line 30 DIMensions the a-array for eleven locations, a(1) to a(11). These array locations will be used later as a convenient place to hold variables in an organized fashion. Simple variables, such as x1–x11 could also be used with the T/S 2068. Why did I use arrays? I originally wrote this program for a computer that had only single-letter variables (A-Z), and it was necessary to use arrays as variables to keep some sense of order. When I translated the program to the T/S 2068 I retained this approach. Besides, it is good for you to get used to using arrays so you won't be afraid of them when you really need them!

Screen Briefing

It's common, before a flight, to get a briefing. That's what this program does in lines 100–200. Line 210 gives you a chance to read and understand it by GOing to the SUBroutine in line 3000. When you press ENTER, line 3010 shoots the program back to line 220, which jumps it to the subroutine starting in line 2000. Why all this jumping around? Why not? The computer doesn't care, and it allows you to "modularize" your program. Since a line renumber capability is not part of the T/S 2068 BASIC language, whenever you want to add a significant part of the program it makes sense to make it a module with higher line numbers and use GO SUB or GO TO to get there.

The Earth Display

Part of the briefing is to explain terminology. Since a picture is supposed to be worth a thousand words, the computer is used to draw the Earth with identification of the terms used in the program.

Chapter 9, "Grappling with Graphics," explains how PLOT and DRAW statements operate. Lines 2000–2220, and line 2410, give you plenty of examples. Lines 2300–2400 and 2420 provide examples of the power of the PRINT AT statement.

Why no RETURN after line 2420? This is a GO SUB, isn't it? Yes, but it's allowed to continue on to line 3000 for the "Press ENTER to continue . . ." prompt, which provides a RETURN in line 3010. This illustrates how subroutines can be combined.

Before we leave the graphic globe, however, we need to explain line 2410. When the word "Pole" is printed, by line 2300, a couple of pixels at the upper corner of the triangle are wiped out. This is because the entire characters are printed, changing whatever was in those spaces. Line 2410 replaces the "lost" pixels, and illustrates one way of "customizing" a character location on the screen.

Terminology

As shown by the display, "longitude" lines are those parallel lines going north and south, and (although not shown in this diagram) passing through the poles. These are "great circles" used to specify angles east or west of Greenwich, England, which is considered zero longitude. A typical longitude, specified in decimal degrees, might be "longitude 115.35 E."

"Latitude" lines are lines going east and west, parallel to the equator, but are *not* great circles. (The equator, latitude zero, *is* a great circle.) They are used to specify an angle north or south of the equator, such as "latitude 37.5 N."

A typical spherical triangle is shown, with the North Pole, each corner, each side, and two angles labelled. If point a is the "origin" and point b the "destination," then side d would be the great circle distance between them, and angle g would be the direction measured clockwise from north. Side f would represent the angle from point a to the North Pole, side e the angle from point b to the North Pole, and angle i would be the angular distance between them at the Pole. These are not important to use the program, but they are used in the solution of the spherical triangle, as you'll see.

Input Prompts

Lines 300–430 ask you for the origin and destination positions, and the relative direction between them. You'll find that better maps and encyclopedias give locations in latitude and longitude, using degrees and minutes of arc. A last choice is a globe, since you'll have to estimate from the latitude and longitude markings.

The program is designed to handle inputs from either the northern or southern hemisphere, and longitudes east or west of Greenwich, but it can get confused if both the origin and the destination are on the same longitude. In this case, you don't need the program at all, since longitude lines are great circles. One degree of arc is equal to 60 nautical miles, and the direction between them must be 0 or 180 degrees! To use the program if this is the case, however, you may offset either the origin or destination longitude by one degree; the error will be insignificant.

Decimal Degrees

Although earth coordinates are usually specified in degrees, minutes, and seconds of arc for longitude and latitude, we will use decimal degrees only. This greatly simplifies the calculations without significantly affecting the accuracy. There are 60 minutes in one degree, and 60 seconds in one minute of arc. For our purposes, we can ignore seconds entirely.

You'll need to convert minutes of arc to decimal parts of a degree for use in this program. One minute of arc equals .01667 degree. For example, 16 degrees and 42 minutes would be 16.7 decimal degrees. (You multiply 42 times .01667 and get .7.) To convert the decimal part of a degree to minutes, divide the decimal by .01667.

Calculating the Results

Line 440 is a delay loop which allows you to see your last input for a short time before line 500 clears the screen. Next line 500 sets a "flag," variable f, to indicate if your destination is east of the origin. If it is, line 520 PRINTs the origin on the left side of the screen and the destination on the right side (since most of us think of west as left, and east as right).

If the destination is west, then line 510 PRINTs the destination on the left side of the screen and the origin on the right side.

Either way, we get to line 530, where two asterisks (with a space

before the first one and 26 spaces between them) are placed on the screen. Line 540 DRAWs a line between them.

Depending on the east/west relationship, as indicated by the value of flag f, either line 550 or line 560 PRINTs the coordinates under the origin and destination names. Notice that the end of line 550 says "GO TO line 570"—and there is no line 570! Many computers would STOP with an error at this point, but the T/S 2068 rolls merrily along to the next statement, line 600. So, either way, we get to line 600.

The Infernal Triangle

The remainder of the program is concerned with solving the spherical triangle between the North Pole, the origin, and the destination, and then PRINTing the results. The REMark statements show the formulae. Lines 600–640 first determine the lengths of arcs f and e, even if they are in different north-south hemispheres. Similarly, lines 650–720 calculate the angle at the North Pole, i, between the origin and the destination, taking into consideration their east-west locations. These calculations are also designed to define the smallest of two possible triangles, since one could go around the globe in a reverse direction.

Lines 800–830 PRINT the "North Polar Angles," and then lines 900–990 do the calculation for the length of the arc d. The formula, from a spherical trigonometry textbook, is shown in line 910, with line 920 showing the conversion factor from degrees to radians (since the computer does all trig calculations in radians). Line 1000 converts back to degrees, then line 1120 calculates nautical miles and line 1130 converts to statute miles (which are 1.1515 times nautical miles). Lines 1120 and 1140 PRINT the results on the screen.

Direction Between Points

Lines 1200–1260 calculate the direction between the origin and destination, using the formula in line 1210. The results are PRINTed by line 1270, and line 1999 STOPs the program before it can redraw the globe.

Do It Yourself

If you are interested in the intricate detail of the calculations, by this time you should be able to figure out for yourself exactly how

they are accomplished. It's too hot out here in the jungle to go into that much detail for the seven people who read this book that really care. If you're one of the "Magnificent Seven," you have my blessing.

MODIFICATIONS

If you are a true perfectionist, modify the program to allow input of degrees, minutes, and seconds rather than decimal degrees. Also, you might wish to add a DATA list of typical locations and their latitudes and longitudes so that you can type in the city name and have the program look in DATA for the coordinates. For your assistance, Table 17-1 shows some of the approximate geographic coordinates of some well-known cities. You can also use these to practice with the program as it is.

Table 17-1. Typical Cities

Origin	Destination	Approx. Latitude	Approx. Longitude	Nautical Miles	Statute Miles	Direction (From North)
Los Angeles 34 N 118.5 W	Hong Kong	22 N	112 E	6373.67	7339.28	311.83
	Tokyo	35 N	140 E	4759.12	5480.13	305.23
	Sydney	35 S	151 E	6544	7535.41	240.07
	Zanzibar	8 S	40 E	8839.01	10178.12	42.23
	Miami	26 N	81 W	1994.99	2297.23	93.69
	New York	41 N	74 W	2135.82	2459.41	65.34
	London	53 N	0	4677.85	5386.55	32.74
	Moscow	55 N	37 E	5312.8	6117.69	13.76
	Paris	45 N	2 E	5062.97	5830.01	37.75
Tokyo 35 N 140 E	Sydney	35 S	151 E	4245	4888.11	170.47
	Miami	26 N	81 W	6462.66	7441.75	38.24
	Moscow	55 N	37 E	4118.67	4742.65	323.12
Paris 45 N 2 E	Hong Kong	22 N	112 E	5260.21	6057.14	60.69
	Zanzibar	8 S	40 E	3782.38	4355.41	136.84
	New York	41 N	74 W	3217.76	3705.25	294.57
Moscow 55 N 37 E	New York	41 N	74 W	4051.5	4665.3	310.32
	Sydney	35 S	151 E	7882.35	9076.52	94.28

QUESTIONS

1. What kind of trigonometry is used in this program?
2. What's wrong with ordinary maps for accurate long-distance navigation?
3. Define a "great circle."
4. Are longitude lines normally vertical or horizontal on maps? Are they great circles?
5. Are latitude lines normally vertical or horizontal on maps? Are they great circles?
6. Zero longitude goes through what country? What town?
7. What latitude is the equator?
8. How many minutes of arc in one degree? How many degrees in one minute of arc?
9. How many nautical miles in one degree of a great circle? How many statute miles?

VARIABLES

a(1)	origin latitude
a(2)	origin longitude
a(3)	destination latitude
a(4)	destination longitude
a(5)	COS(e*j)
a(6)	COS(f*j)
a(7)	SIN(e*j)
a(8)	SIN(f*j)
a(9)	COS(i*j)
a(10)	arc d
a(11)	SIN(d*j)
a$	origin
b$	destination
d	origin/destination arc
e	destination polar angle
f	east/west flag, origin polar angle
g	origin/destination angle
i	origin/destination polar angle
j	degree to radian conversion factor
j$	origin north/south flag
k$	origin east/west flag
l$	destination north/south flag
m$	destination east/west flag
p$	origin/destination east/west flag
y	arc whose COSine is d, arc whose COSine is g
z$	dummy screen delay INPUT

APPENDICES

APPENDIX A
Differences Between the T/S 1000/1500 and the T/S 2068

New keys on T/S 2068:
- Second Caps Shift Key
- New Space Bar

T/S 1000/1500 Commands/Statements/Functions and Punctuation Not on T/S 2068:

- " " (DOUBLE QUOTE)
- UNPLOT
- SCROLL
- ** (UP-ARROW ON 2068)
- [S] CURSOR ([?] CURSOR ON 2068)
- [F] CURSOR ([E] CURSOR ON 2068)
- FUNCTION KEY

T/S 2068 Commands/Statements/Functions and Punctuation Not on T/S 1000/1500:

- CAPS LOCK
- DEF FN
- CLOSE #
- POINT
- VERIFY
-] (RIGHT BRACKET)
- RESET
- \ (BACKSLASH)
- CIRCLE
- BEEP
- FLASH
- INVERSE
- #
- & (AMPERSAND)
- DATA
- DRAW
- [E] CURSOR (WAS F-CURSOR)
- TRUE VIDEO
- LINE
- MOVE
- CAT
- MERGE
- IN
- FREE
- ON ERR
- SCREEN $
- INK
- BRIGHT
- @ (AT SIGN)
- ! (EXCLAMATION)
- CAPS SHIFT
- RESTORE
- ' (APOSTROPHE)
- [C] CURSOR
- INV. VIDEO
- OPEN #
- ERASE
- FORMAT
- [(LEFT BRACKET)
- OUT
- STICK
- SOUND
- ATTR
- PAPER
- OVER
- ↑ (WAS ** ON 1000)
- % (PERCENT)
- READ
- BIN
- { (LEFT BRACE)
- } (RIGHT BRACE)
- BORDER
- © (COPYRIGHT)
- FN
- _____ (UNDERLINE)
- [?] CURSOR (WAS S-CURSOR)

APPENDIX B
Block Graphics Characters

Character (■ = INK)	Key(s) #	Key(s) Caps Shift?	True Video or Inv. Video?	CHR$ #
	8	NO	TRUE	128
	1	NO	TRUE	129
	2	NO	TRUE	130
	3	NO	TRUE	131
	4	NO	TRUE	132
	5	NO	TRUE	133
	6	NO	TRUE	134
	7	NO	TRUE	135
	7	YES	INV.	136
	6	YES	INV.	137
	5	YES	INV.	138
	4	YES	INV.	139
	3	YES	INV.	140
	2	YES	INV.	141
	1	YES	INV.	142
	8	YES	INV.	143

APPENDIX C
Special Music Characters

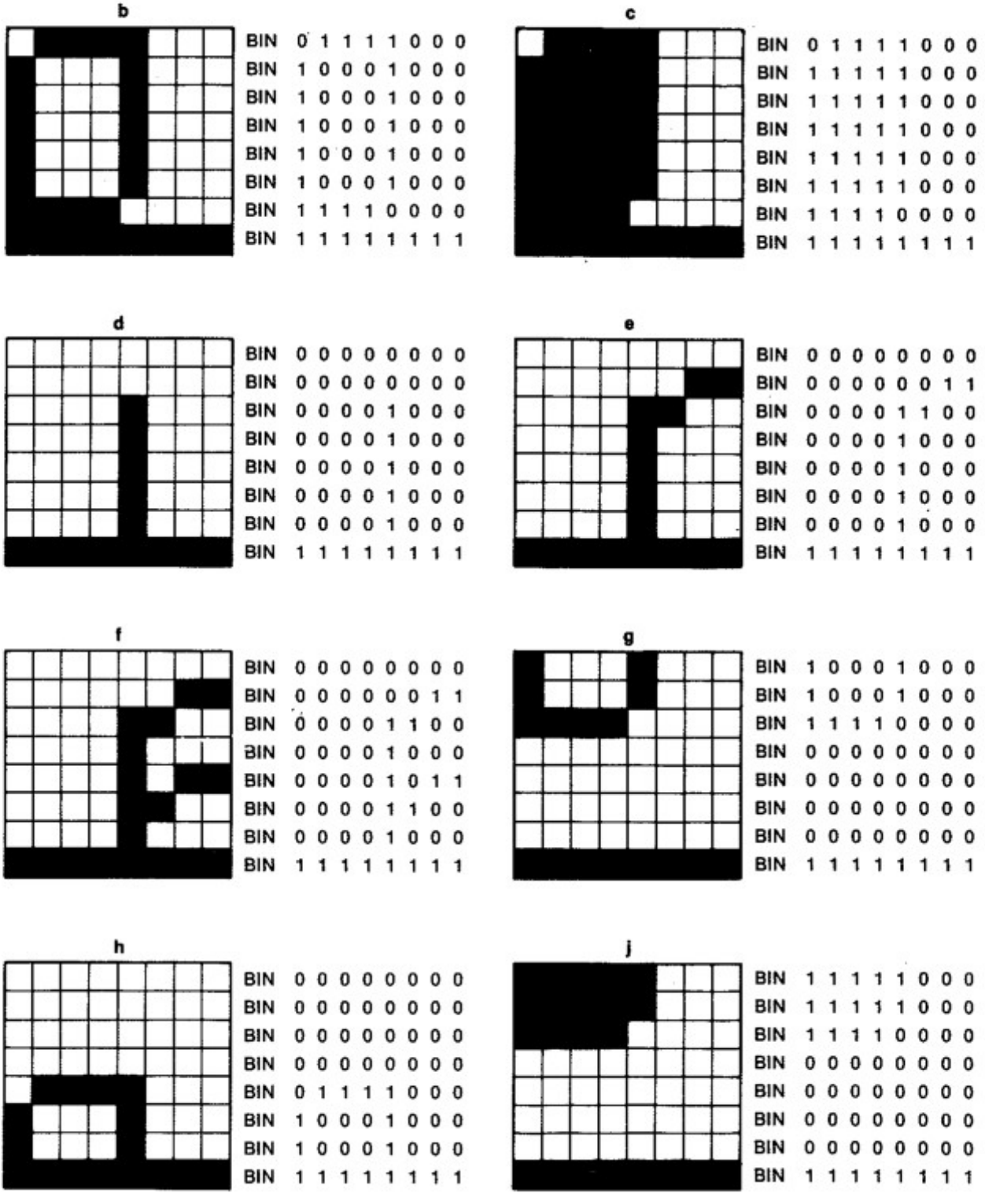
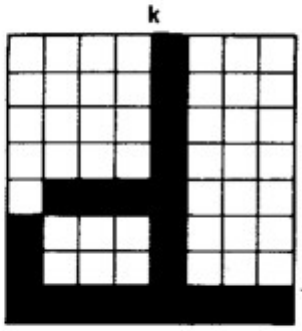
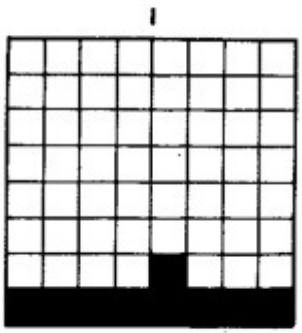


Fig. C-1. Special Music Characters.



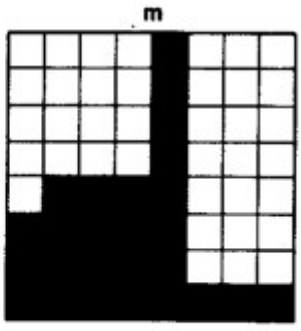
k

BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 1 1 1 1 0 0 0
BIN 1 0 0 0 1 0 0 0
BIN 1 0 0 0 1 0 0 0
BIN 1 1 1 1 1 1 1 1



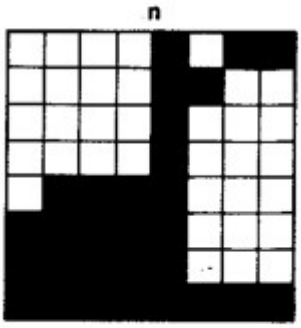
l

BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 1 1 1 1 1 1 1 1



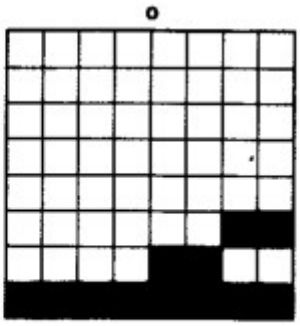
m

BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 1 1 1 1 0 0 0
BIN 1 1 1 1 1 0 0 0
BIN 1 1 1 1 1 0 0 0
BIN 1 1 1 1 1 1 1 1



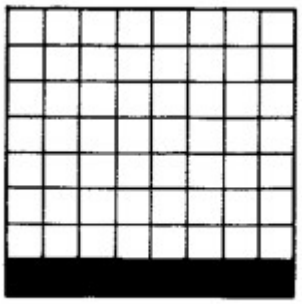
n

BIN 0 0 0 0 1 0 1 1
BIN 0 0 0 0 1 1 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 0 0 0 1 0 0 0
BIN 0 1 1 1 1 0 0 0
BIN 1 1 1 1 1 0 0 0
BIN 1 1 1 1 1 0 0 0
BIN 1 1 1 1 1 1 1 1



o

BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 0 0
BIN 0 0 0 0 0 0 1 1
BIN 0 0 0 0 1 1 0 0
BIN 1 1 1 1 1 1 1 1

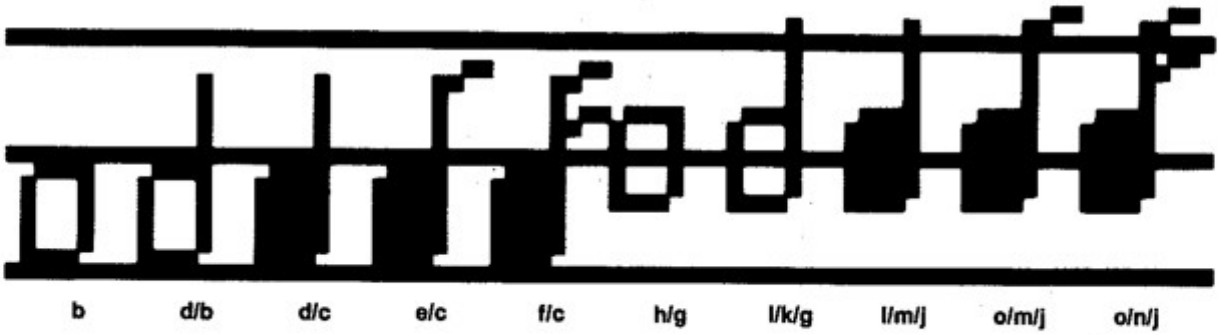


UNDERLINE CHARACTER (SHIFTED 0)

KEYBOARD CHARACTER (USE FOR STAFF LINES)

Fig. C-1. Special Music Characters—cont.

Graphic note combinations. First PRINT staff lines, then use PRINT AT to place notes on staff. Most notes take 2 or 3 vertical spaces.



b d/b d/c e/c f/c h/g l/k/g l/m/j o/m/j o/n/j

Fig. C-2.

```
   A musical staff is drawn and
graphic representations of notes
are drawn at typical locations.

Press ENTER...
```

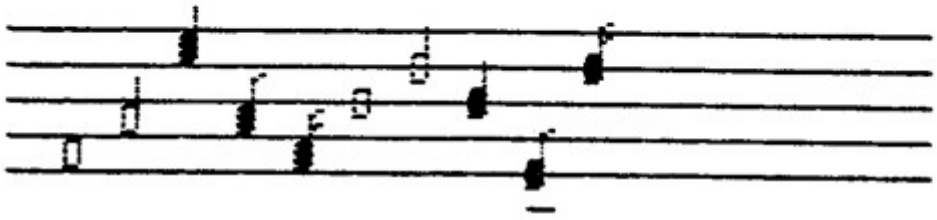


Fig. C-3. Typical Notes.

LISTing C-1

```
 100 REM * Musical Notes *
 110 REM * (C) Fred Blechman 198
3 *
 120 PRINT : PRINT "   A musical
 staff is drawn and"
 130 PRINT "graphic representati
ons of notes"
 140 PRINT "are drawn at typical
 locations."
 150 PRINT : PRINT "Press ENTER.
..";: INPUT a$: CLS
 200 POKE USR "d"+0,BIN 00000000
 210 POKE USR "d"+1,BIN 00000000
 220 POKE USR "d"+2,BIN 00001000
 230 POKE USR "d"+3,BIN 00001000
 240 POKE USR "d"+4,BIN 00001000
 250 POKE USR "d"+5,BIN 00001000
 260 POKE USR "d"+6,BIN 00001000
 270 POKE USR "d"+7,BIN 11111111
 300 POKE USR "b"+0,BIN 01111000
 310 POKE USR "b"+1,BIN 10001000
 320 POKE USR "b"+2,BIN 10001000
 330 POKE USR "b"+3,BIN 10001000
 340 POKE USR "b"+4,BIN 10001000
 350 POKE USR "b"+5,BIN 10001000
 360 POKE USR "b"+6,BIN 11110000
 370 POKE USR "b"+7,BIN 11111111
 400 POKE USR "c"+0,BIN 01111000
 410 POKE USR "c"+1,BIN 11111000
 420 POKE USR "c"+2,BIN 11111000
 430 POKE USR "c"+3,BIN 11111000
 440 POKE USR "c"+4,BIN 11111000
 450 POKE USR "c"+5,BIN 11111000
 460 POKE USR "c"+6,BIN 11110000
 470 POKE USR "c"+7,BIN 11111111
 500 POKE USR "e"+0,BIN 00000000
```

LISTing C-1—cont.

```
 510 POKE USR "e"+1,BIN 00000011
 520 POKE USR "e"+2,BIN 00001100
 530 POKE USR "e"+3,BIN 00001000
 540 POKE USR "e"+4,BIN 00001000
 550 POKE USR "e"+5,BIN 00001000
 560 POKE USR "e"+6,BIN 00001000
 570 POKE USR "e"+7,BIN 11111111
 600 FOR n=0 TO 7: READ p: POKE
USR "f"+n,p: NEXT n
 610 DATA BIN 00000000,BIN 00000
011,BIN 00001100,BIN 00001000,BI
N 00001011,BIN 00001100,BIN 0000
1000,BIN 11111111
 620 FOR n=0 TO 7: READ p: POKE
USR "h"+n,p: NEXT n
 630 DATA BIN 00000000,BIN 00000
000,BIN 00000000,BIN 00000000,BI
N 01111000,BIN 10001000,BIN 1000
1000,BIN 11111111
 640 FOR n=0 TO 7: READ p: POKE
USR "g"+n,p: NEXT n
 650 DATA BIN 10001000,BIN 10001
000,BIN 11110000,BIN 00000000,BI
N 00000000,BIN 00000000,BIN 0000
0000,BIN 11111111
 660 FOR n=0 TO 7: READ p: POKE
USR "l"+n,p: NEXT n
 670 DATA BIN 00000000,BIN 00000
000,BIN 00000000,BIN 00000000,BI
N 00000000,BIN 00000000,BIN 0000
1000,BIN 11111111
 680 FOR n=0 TO 7: READ p: POKE
USR "k"+n,p: NEXT n
 690 DATA BIN 00001000,BIN 00001
000,BIN 00001000,BIN 00001000,BI
N 01111000,BIN 10001000,BIN 1000
1000,BIN 11111111
 700 FOR n=0 TO 7: READ p: POKE
USR "j"+n,p: NEXT n
 710 DATA BIN 11111000,BIN 11111
000,BIN 11110000,BIN 00000000,BI
N 00000000,BIN 00000000,BIN 0000
0000,BIN 11111111
 720 FOR n=0 TO 7: READ p: POKE
USR "m"+n,p: NEXT n
 730 DATA BIN 00001000,BIN 00001
000,BIN 00001000,BIN 00001000,BI
N 01111000,BIN 11111000,BIN 1111
1000,BIN 11111111
 740 FOR n=0 TO 7: READ p: POKE
USR "o"+n,p: NEXT n
 750 DATA BIN 00000000,BIN 00000
000,BIN 00000000,BIN 00000000,BI
N 00000000,BIN 00000011,BIN 0000
```

LISTing C-1—cont.

```
1100,BIN 11111111
 760 FOR n=0 TO 7: READ p: POKE
USR "n"+n,p: NEXT n
 770 DATA BIN 00001011,BIN 00001
100,BIN 00001000,BIN 00001000,BI
N 01111000,BIN 11111000,BIN 1111
1000,BIN 11111111
 800 PRINT : FOR x=1 TO 5
 810 PRINT "                    "
                    "
 820 NEXT x
1000 PRINT AT 5,2;"B": PRINT AT
3,4;"D": PRINT AT 4,4;"B"
1010 PRINT AT 1,6;"D": PRINT AT
2,6;"C"
1020 PRINT AT 3,8;"E": PRINT AT
4,8;"C"
1030 PRINT AT 4,10;"F": PRINT AT
 5,10;"C"
1040 PRINT AT 3,12;"H": PRINT AT
 4,12;"G"
1050 PRINT AT 1,14;"L": PRINT AT
 2,14;"K": PRINT AT 3,14;"G"
1060 PRINT AT 2,16;"L": PRINT AT
 3,16;"M": PRINT AT 4,16;"J"
1070 PRINT AT 4,18;"O": PRINT AT
 5,18;"M": PRINT AT 5,18;"J"
1080 PRINT AT 1,20;"O": PRINT AT
 2,20;"N": PRINT AT 3,20;"J"
```

APPENDIX D

Answers to Questions

Chapter 1

1. Sinclair Spectrum; yes.
2. ROM.
3. 64K; 64K × 256 = 16.384 megabytes.
4. 16K.
5. LOAD“ ” or LOAD“filename”.
6. Arrangement of letters on the keyboard. QWERTY describes the first six characters on the second row of a standard typewriter.
7. 16; 21.
8. 32; 24; 256; 176.
9. Four.

Chapter 2

1. Use SYMBL SHIFT key.
2. E-cursor.
3. E-cursor followed by SYMBL SHIFT and C keys.
4. First (usually, depending on previous conditions.)
5. 10 to the 38th power.
6. Yellow.
7. 0; keyword DELETE.
8. Syntax error on line entry.
9. Respond to prompt with 0, or SYMBL SHIFT and A keys.

Chapter 3

1. Display Mode 1; “Timex 2000 Intermediate/Advanced Guide.”
2. 32; 24; 256; 176.
3. Picture elements; 256 × 176 = 45056.
4. Defeats “scroll?” interrupt.

5. Puts next PRINT in following character space; moves next PRINT to next half-screen; advances cursor two screen lines.
6. Vertical; horizontal.
7. Line 3 column 4.
8. 10 pixels to right and 20 pixels up from lower left corner.

Chapter 4

1. 0 and 9999; no.
2. LIST 50; this becomes the "current line."
3. DELETE 100,200.
4. Two; four.
5. COPY.
6. The commands are ignored.
7. Changeable value; numeric or string.
8. MIC; EAR.
9. MERGE; line numbers.

Chapter 5

1. POKE23692,255.
2. For x = 50 TO 100 STEP 5.
3. Incrementing.
4. TO; THEN; RETURN.
5. Same functions performed more than once.
6. 30.
7. Enter a letter.
8. Inside out.
9. Temperature variation.

Chapter 6

1. Strings can have any characters, not just numbers.
2. With L-cursor, press CAPS SHIFT and SYMBL SHIFT for E-cursor, then SYMBL SHIFT and P key; same with F key; same with G key.
3. Use two sets of quotes before and after quoted text.
4. est.
5. 88; 63.

6. Test.
7. Asterisk; up-arrow.
8. 234; 234; −13, −13.
9. No; yes.

Chapter 7

1. Sequential.
2. Yes; no.
3. Yes.
4. Yes.
5. Yes; delimiters; yes—for strings.
6. Either.
7. Only limited by memory; same.
8. String array with 3 rows, 4 columns, 5 levels and 6 characters in each location.
9. Yes.

Chapter 8

1. Eight; two each.
2. Shades of gray.
3. 32; 256.
4. One; one.
5. Two; 64.
6. Seven; no.
7. PAPER red; INK white.
8. Graphic block with left-half PAPER color, right-half INK color.
9. 0 to 15; no.

Chapter 9

1. Eight; PAPER; SYMBL SHIFT with G-cursor.
2. Change ARF! to MEOW!
3. See text.
4. 256; 45056.
5. Over 18 billion-billion!
6. The green INK changes to red; the yellow INK changes to red;

the red INK remains red; the white PAPER is unchanged; the red PAPER makes the red INK invisible!
7. One (the color of the thermal paper used).
8. 2 times PI; 360; 57.29578.
9. Radians.

Chapter 10

1. Duration, frequency.
2. One-half.
3. C-sharp for two seconds.
4. 10.49.
5. 69; −69.
6. 12.
7. C-sharp is one semi-tone above C, D-flat is one semi-tone below D, and these are the same frequencies.
8. 12.
9. 440 cycles per second.

Chapter 11

1. See text.
2. Until replaced or computer turned off.
3. Draws the racer's lanes.
4. To assure random selection of runners.
5. To assure variable r is a whole number for use with BEEP.
6. Calculated branching.
7. See text.
8. Nothing. The program accepts it.
9. Unnecessary line that does nothing and hurts nothing.

Chapter 12

1. Phone book or phone company.
2. Three minutes.
3. Minutes; half-minutes.
4. INV.VIDEO.
5. Instructions will be bypassed.

6. Keeps FLASH from affecting later PRINTing. If FLASH 0 left out, all new PRINTing would FLASH.
7. Slower.
8. See text.
9. Unlimited.

Chapter 13

1. Physical = 23; Sensitivity = 28; Intellectual = 33.
2. Divisible by four with no remainder, except century years.
3. Yes. Divisible by 400 with no remainder.
4. Yes; much faster.
5. .0174533; multiply.
6. See text.
7. Program would continue thru to line 1040 and then end with an error report.
8. 1700, 1800 and 1900 not leap years, which would introduce error.
9. INK dots only.

Chapter 14

1. Hidden under RESET (P key); With L-cursor, press CAPS SHIFT and SYMBL SHIFT keys to get E-cursor, then press SYMBL SHIFT and P keys.
2. The string is completely empty.
3. Program stops with error report.
4. 127, 73, 73, 65, 65.
5. Space between banner characters.
6. A's.
7. 28.
8. Add more ":LPRINT p$" in line 340; Use less a$ on line 500 and same lesser number of blanks on line 510.
9. Yes. Put proper numbers in DATA statements.

Chapter 15

1. Data base.
2. No

3. Printer ignored.
4. Printer ignored.
5. 5.
6. To get a whole number.
7. No; one.
8. No; array dimensioned for 100.
9. Minus sign.

Chapter 16

1. Returns the DATA pointer to the first DATA statement.
2. Two beats of middle-C.
3. Change the 16 in line 100 to a smaller number; change the 16 in line 100 to a larger number.
4. Program would run out of DATA and stop with error report.
5. 69; above normal hearing range.
6. Put BEEP values in DATA and READ.
7. Good luck, Irving!

Chapter 17

1. Spherical.
2. Not accurate for distance or direction—or both!
3. Intersection of the surface of a sphere with a flat plane passing through the center of the sphere.
4. Vertical; yes.
5. Horizontal; no (except equator).
6. England; Greenwich.
7. 0.
8. 60; .01667.
9. 60.

INDEX

HOWARD W. SAMS CRASH COURSE IN MICROCOMPUTERS (2nd Edition)

An outstanding book for those who need to know a lot in a hurry about microcomputers and programming! New chapters cover 16-bit microcomputing and BASIC programming, new photos and illustrations aid your understanding, and an expanded applications chapter emphasizes software. By Louis E. Frenzel, Jr. 320 pages, 8½ × 11, comb-bound. ISBN 0-672-21985-9. © 1983.

Ask for No. 21985 . . . . . $21.95

USER'S GUIDE TO MICROCOMPUTER BUZZWORDS

A handy quick-reference for those people who don't care what happens inside a microcomputer when it's turned on, yet find they must know enough to be able to communicate with others who do. Contains many illustrations. By David H. Dasenbrock. 110 pages, 5½ × 8½, softbound. ISBN 0-672-22049-0. © 1983.

Ask for No. 22049 . . . . . $9.95

HOW TO MAINTAIN AND SERVICE YOUR SMALL COMPUTER

Shows you some easy maintenance and operating procedures that can sharply reduce personal-computer problems and down-time. Also shows you how to diagnose what's wrong, how to find the faulty part, and how to make most simple, money-saving repairs yourself. By John G. Stephenson and Bob Cahill. 224 pages, 8½ × 11, softbound. ISBN 0-672-22016-4. © 1983.

Ask for No. 22016 . . . . . $17.95

MEGABUCKS FROM YOUR MICROCOMPUTER

Shows you how to make money doing your own writing, reviewing, and programming with your microcomputer. You'll learn about some dangers, get some tips on choosing the right microcomputer, and maybe develop some talents you didn't even know you had! By Timothy Orr Knight. 80 pages, 8½ × 11, softbound. ISBN 0-672-22083-0. © 1983.

Ask for No. 22083 . . . . . $3.95

ELECTRONICALLY SPEAKING: COMPUTER SPEECH GENERATION

Teaches you the techniques for generating synthetic speech and includes a synthesizer overview, advice on what you can do about possible problem areas, and a history of synthetic speech research since the 1800s. By John P. Cater. 232 pages, 5½ × 8½, softbound. ISBN 0-672-21947-6. © 1982.

Ask for No. 21947 . . . . . $14.95

USING COMPUTER INFORMATION SERVICES

Shows you how to use your microcomputer to communicate with the national computer networks. Clearly explains what's available, how to program your computer to retrieve it automatically, how to use your computer as a powerful communications tool, and more. By Larry W. Sturtz and Jeffrey R. Williams. 240 pages, 5½ × 8½, softbound. ISBN 0-672-21997-2. © 1983.

Ask for No. 21997 . . . . . $12.95

These and other Sams Books and Software products are available from better retailers worldwide, or directly from Sams. Call 800-428-SAMS or 317-298-5566 to order, or to get the name of a Sams retailer near you. Ask for your free Sams Books and Software Catalog!

Prices good in USA only. Prices and page counts subject to change without notice.

SAMS

Timex Sinclair 2068 Beginner/Intermediate Guide

Written specifically for the Timex 2068 Personal Color Computer, this book features a learn-by-doing format that benefits both the beginning computer user and the user with some programming practice.

For BEGINNERS, it:

- Shows how to get the Timex Sinclair 2068 up and running.
- Demonstrates the basics of BASIC programming on the 2068.
- Uses practical, easy-to-follow programs to acquaint you with the 2068's keyboard and commands.

For INTERMEDIATE users, it:

- Includes programs for practical home and hobby use which demonstrate the full range of functions and capabilities of the 2068.
- Shows how to add "frills"—such as color, music, and graphics—to your programs for added impact and attractiveness.

As an added educational aid, review questions are included at the end of each chapter so that you can test your knowledge and programming skills.

Howard W. Sams & Co., Inc.
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

$9.95/22225

ISBN: 0-672-22225-6