THE TI 99/4A USER'S GUIDE

CAROL ANN CASCIATO AND DONALD J. HORSFALL

The TI 99/4A
User's Guide

Carol Ann Casciato and **Don Horsfall** are the principals in International Technical Communications, Inc., a computer systems research and consulting firm in the Philadelphia area. For the last 12 years, they have done management and systems consulting, research, and writing for a variety of Fortune 500 clients.

Their first exposure to professional writing came when they produced more than 25 manuals for a large technical documentation project. Since that time, they have written in-depth computer industry research reports, detailed technical product analyses, and manuals for microcomputer manufacturers.

When not reading, writing, or consulting on computers, Carol Ann prepares elaborate chocolate desserts and cares for her large collection of exotic plants. Don's interests include science fiction, restoring his Victorian home, and collecting space art.

The TI 99/4A User's Guide

Carol Ann Casciato

and

Donald J. Horsfall

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1983

International Standard Book Number: 0-672-22071-7
Library of Congress Catalog Card Number: 86-61067

Edited by *Welborn Associates*
Illustrated by *R. E. Lund*

Printed in the United States of America.

PREFACE

In the last year, *millions* of home computers were sold. As an owner of a TI 99/4A, you are yourself *one in a million.*

This is a *guidebook* for new owners of the TI 99/4A. Its purpose is to introduce you to the many options available for improving and expanding your TI 99/4A.

We touch on many topics in this book. We do this because new computer owners need a *broad survey* of many new ideas to orient themselves in an unfamiliar territory.

It is often difficult to understand the world of computers. We ease you into this world, explaining what common computer jargon means. We give you information so that you can get what you need, for the best possible price, in the TI marketplace and in the overall computer market.

We give you the best sources for computer information, programs, and education to help you get the most out of your TI 99/4A.

We also explore such things as buying additional software, writing your own, and the fundamentals of the exotic features of the TI 99/4A (graphics, sound, and voice). This review is intended as a survey of what you can do with your TI 99/4A—a survey that allows you to choose the things that you find interesting or useful from the vast array of possibilities.

Most of you will want to add to your TI 99/4A at some time. We tell you what the add-on pieces do and how much they cost. We provide the information you need to make reasonable plans for expanding your system.

We give you some tips on handling your TI 99/4A and the equipment and supplies that you use with it.

For those who are interested, we have included an *introduction* to some of the technical aspects of the TI 99/4A. We discuss bytes, bits, K's, RAM, ROM, addresses, the 9900 microprocessor, hexadecimal, and other mysterious words.

And, finally, to get you started with sound and graphics on your TI 99/4A, we have included several programs. Enter the programs and see how they work. Then, customize them to meet your own needs. Maybe change the wording in the questions. Maybe change the song in the sound programs. These programs were tested so you can be sure they will work (if you don't make any typing errors entering them).

CAROL ANN CASCIATO
DONALD J. HORSFALL

List of Trademarks

UCSD Pascal and *UCSD p-System*—Regents of the University of California

MultiPlan—MicroSoft, Inc.

PLATO—Control Data Corporation

TI 99/4A and the various products—Texas Instruments, Inc.

DEC—Digital Equipment Corporation

TEXNET—Service Mark of Texas Instruments Inc.

The Source—Service Mark of Source Telecomputing Corporation, a subsidiary of Reader's Digest Association, Inc.

Note

Throughout this book, reversed letters (e.g., FCTN) designate keystrokes. That is, when this designation is used, it means the reversed letters are a single keystroke on the keyboard (or keys that are pressed simultaneously to produce a desired result) instead of being input as individual characters.

CONTENTS

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8

1

FIRST THINGS

You are reading this book because you have a TI 99/4A home computer. We will try to make it easy for you to use your computer to do what you want it to do.

In this chapter, we tell you where you can find more information about your TI 99/4A and products for it, where to learn programming, and some of the common problems beginners encounter with the TI 99/4A.

1.1 INTRODUCTION

This book is a *guidebook* designed to help you through the complex maze of new ideas, new components, and new software that you will find for your TI 99/4A.

We provide advice (and warnings) that you would expect to hear from friends experienced in the computer field. It's the kind of stuff that you should know before you buy any more equipment or software.

We don't intend to replace the information that comes with your computer. The manuals that come with your computer tell you how to connect your equipment and start it up. *Read the installation instructions you get with the computer—and follow the instructions.*

We hope you will enjoy your experiences with your home computer and use this book as a way to plan for its continued use and expansion.

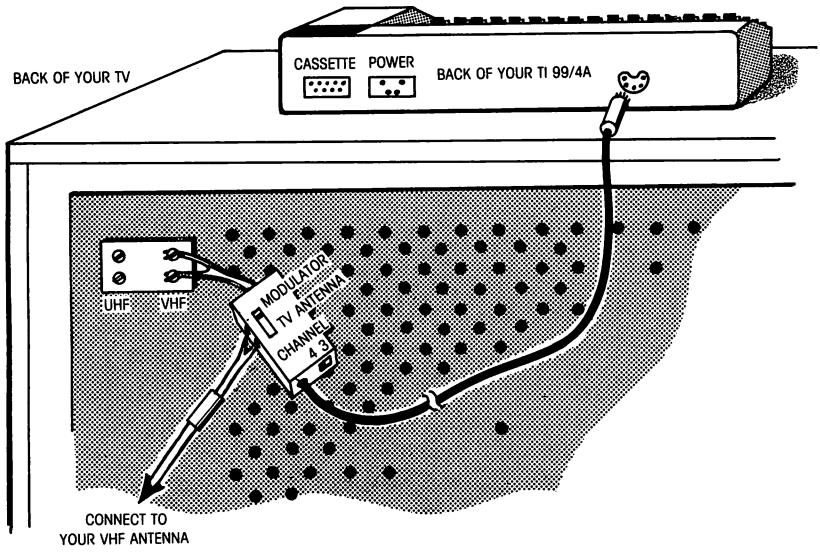


Fig. 1-1. Connecting rf modulator to computer and tv set.

1.2 SOME COMMON PROBLEMS

Before we go any further, there are some problems with the TI 99/4A that are so common we have to discuss them right away. We are not talking about the intricacies of programming. We are talking about mechanical mistakes relating to the cassette cables, the joysticks, and the keyboard.

1.2.1 Connecting Your TV

If your tv has the standard two-wire antenna connector, called a 300-ohm connector, you can easily hook up the rf modulator that comes with the TI 99/4A. Fig. 1-1 shows you how to connect the rf modulator to a tv with standard antenna connections.

Many newer tv sets come *cable ready*. They don't have a 300-ohm two-wire connector. If you have one of these you will need a:

300- to 75-ohm converter

These are only $3.00 to $5.00 and you can find them at many electronics and appliance stores (Radio Shack is a good source). Fig. 1-2 shows you how to connect a 300- to 75-ohm converter to your rf modulator.

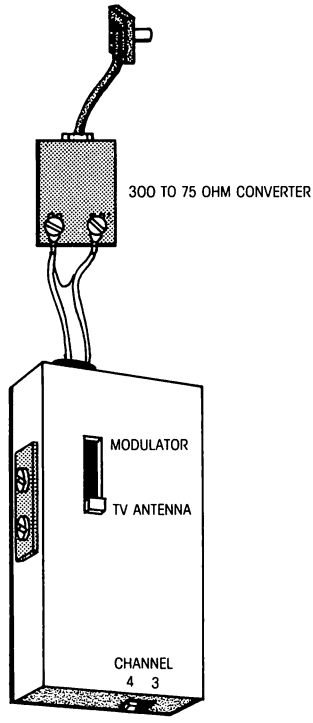


Fig. 1-2. Connecting rf modulator to a 300 to 75 ohm converter.

If your tv runs off a coax cable, you need a:

75- to 300-ohm converter

Fig. 1-3 shows you how to use the 75- to 300-ohm converter to connect your input coax cable to the rf modulator. If you also have a cable-ready tv, you need a 300- to 75-ohm converter as shown in Fig. 1-2.

1.2.2 Joystick and Cassette Cables

Problem number one is the cassette cable and the joysticks have the *same type of connector.*

Plug the cassette cables and joysticks into the correct places on the TI 99/4A or they will not work.

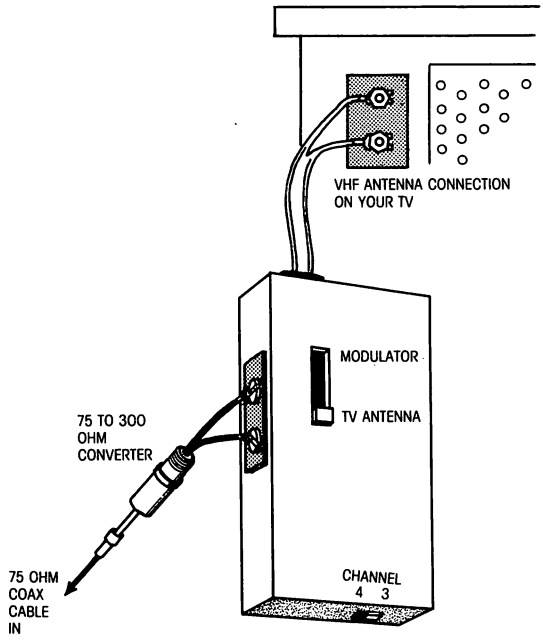


Fig. 1-3. Connecting rf modulator to a 75 to 300 ohm converter.

The *cassette cable* goes into the *back* of the console. The *joysticks* go into the *left side* (nearest the Q key) of the console. Fig. 1-4 shows you the various connections on the sides and back of your TI 99/4A.

1.2.3 Attaching a Cassette Recorder

And then there's the cassette cable itself. The five wires, on two leads, can connect to two recorders. On the TI 99/4A, the:

Cassette attached to the 3-wire lead is called *CS1*
Cassette attached to the 2-wire lead is called *CS2*
Red lead to the microphone jack (MIC)
Black lead to the remote jack
White lead to the speaker jack (monitor)

The *red lead* (in the microphone jack) is used to write data and programs on the cassette tape.

The *black lead* (in the remote jack) is used for remote control operation (starting and stopping) of the cassette recorder—*Not all cassette recorders have a remote control capability.* If your recorder does

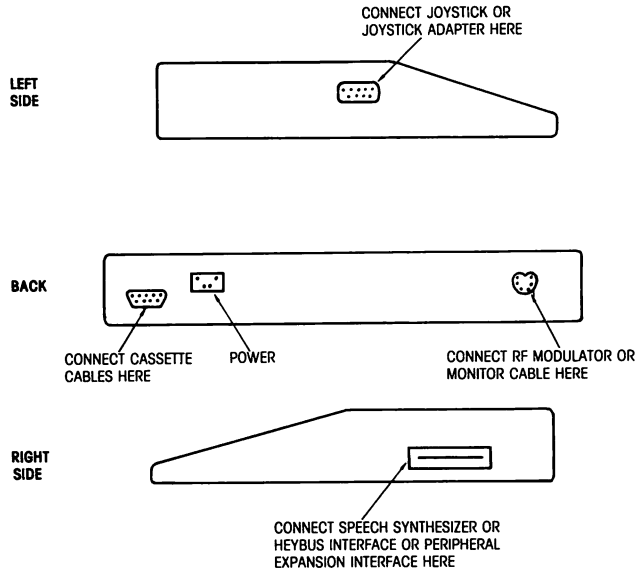


Fig. 1-4. TI 99/4A connections.

not have a remote jack, just ignore the black lead. All this means is *you will* have to start and stop the tape manually.

The *white lead* (in the speaker or monitor jack) is used to read data from a cassette tape. The CS2 connection does not have a white lead, which is why you cannot read from a cassette on CS2.

1.2.4 Using Cassette Tapes

There is nothing magic about computers and cassette tapes. If you have a recorder, you know how to record music on a tape. Cassette tapes with programs are not really different from cassette tapes with songs. Just like when you re-record a song, if you write over a program, it's gone. There is no way to recover the original program.

Rule 1: *Remember* where you put it.

When you write a program or data to a tape, remember to look at which side (1 or 2) of the tape you are recording on. If your recorder has a tape counter (the numbered wheel that moves as the tape moves), keep a list of what you have on the tape and where it is.

Rule 2: Follow the *leader.*

Many cassette tapes have long *leaders—you cannot record on the leader.* Make sure the cassette tape is past the leader before you try to save something on it.

Rule 3: Recorders don't *pick sides.*

Sometimes people become confused about the difference between cassette recorder one (CS1) and side one of a tape. Just remember: you read and write *on* the tape and *to/from* the recorder. You read from and write *to recorder* CS1. You write *to recorder* CS2. You write *on tape side* one or tape side two.

Rule 4: Forget *bargains.*

Don't buy cheap tape! Use good quality cassette tapes no longer than 60 minutes—30 minutes is better.

Data recording is much more sensitive to bad recording than audio recording is. Cheap tapes will disappoint you.

Rule 5: *Be kind* to your tapes.

Store your tapes properly. They should be in a cool, dry place away from magnetic fields. Don't put them on top of the console or the television.

1.2.5 Joystick Interference

Even if you have the joystick plugged into the correct connector (on the left side of the TI 99/4A console), it still might not work right.

> When you use the joysticks, the ALPHA LOCK key (lower left corner of the keyboard) *MUST* be in the *up* or unlocked position. Otherwise, you will not be able to get objects to move in the up direction.

1.3 THE KEYBOARD

Since you use the keyboard to communicate with your computer —it's how you "talk" to your TI 99/4A—you should be aware of what it is and what its keys mean. We cover the keyboard in some detail, with special emphasis on the FCTN and CTRL keys.

1.3.1 Keyboard Layout

The TI 99/4A has a Qwerty keyboard layout (that is the one where the letters on the top row are arranged QWERTYUIOP). The TI 99/4A

keyboard is a full ASCII keyboard, capable of generating *all* the ASCII character codes. This may not mean much to you now, but it means a great deal to someone writing advanced software for the TI 99/4A.

If you don't know how to type, you can learn how. It's much easier to get your programs into the computer once you know how to type. You don't have to be the fastest typist in the world. You just have to remember where the keys are. And, however, it may seem at times, they really don't move around the keyboard.

Fig. 1-5 shows you what the keyboard looks like. You will notice some *extra keys* (like a FCTN key and a CTRL key) and that some keys are darker than the rest. The *outlined keys* work with the FCTN key, which we will talk about a little later.

You can enter both upper and lower case letters from your TI 99/4A keyboard. If you want *upper case* letters or the *special characters* on the top row of the numbers, hold down the SHIFT key when you press the letter or number.

> You can use the ALPHA LOCK key to make your keyboard enter *only* upper case letters. You still get the numbers (not the special characters above the numbers) when you have the ALPHA LOCK key locked.

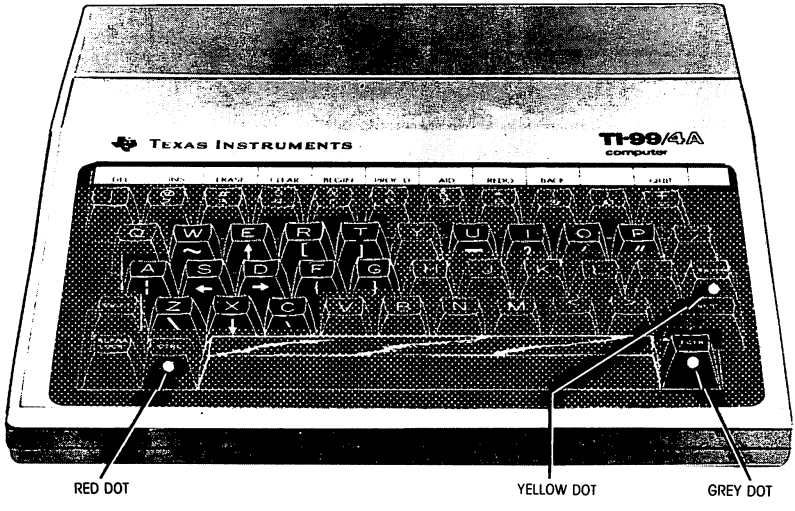


Fig. 1-5. Keyboard diagram.

1.3.2 The FCTN Key and the Letter Keys

The FCTN key is called the *function* key. It is located in the lower right corner of the keyboard and has a *gray dot* on its front.

The FCTN key works a lot like the SHIFT key. When you hold down the FCTN key and press another key that has a *gray character* on the front of it, you get whatever is on the *front* of the key. For example, if you hold the FCTN key and the P key, you will get a double quote (").

1.3.3 The FCTN Key and Overlays

The FCTN key also works with the number keys. Several thin plastic strips, called *overlays*, came with your TI 99/4A console. These overlays fit into the slot above the numbers on your console.

Most of the overlays have two rows of squares (one above each number) and two dots. The *top* row has a *red* dot and the *bottom* row has a *gray* dot. You activate the row with the gray dot using the FCTN key, which has a gray dot on its front.

You put an overlay into the holder above the number row so that the squares line up with the number keys. The leftmost square should be over the 1 key.

One overlay is different—it has a row of squares and a row with words in the squares. Fig. 1-6 shows you what these two overlays look like. The one with the words printed on it is important. You use this one often and may want to copy the words to other overlays.

When you press the FCTN key and one of the numbers, you get the action represented by the word above the number. This means you use only a single keystroke to tell your computer to do some things like insert characters or erase the line.

Some programs use the FCTN values for the numbers to do

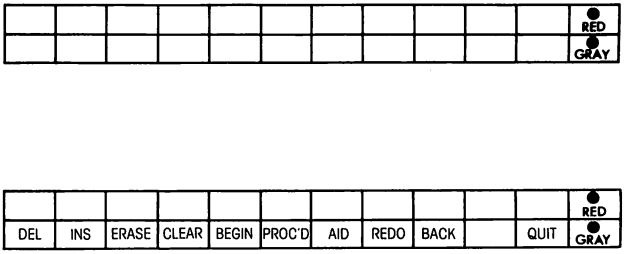


Fig. 1-6. Keyboard overlays.

something. You can receive the values sent by the FCTN keys in BASIC programs and use these keys for special tasks.

Games, for example, may say "Press PROC'D or BACK". If you don't know that you are supposed to hold down the FCTN key while you press 6 (PROC'D) or 9 (BACK), you will probably have some trouble understanding why your computer is not obeying your commands. Table 1-1 shows you the commonly used functions for the number keys.

> Remember to hold down the FCTN key when you want to use one of these functions.

But *BEWARE*. One key causes you to lose whatever you have in your computer's memory. When you press the FCTN *key and the* = (equals) key, you will *QUIT* whatever you are doing and your computer will return to the main screen (the one that you see when you first turn it on).

> If you are entering or running a BASIC program and you hit the FCTN = key, you lose it all. You suddenly find yourself looking at the main title screen.

1.3.4 The CTRL Key

The CTRL key works in much the same way as the FCTN key, except that it causes different codes to be sent. As with the FCTN and SHIFT keys, you hold down the CTRL key while pressing another key.

Some programs, like the Terminal Emulator and Editor/Assembler, use the CTRL key along with other keys to get special control codes from you.

Table 1-1 FCTN/Number Keys

Key	What It Does
FCTN 1	DELETE (delete a character)
FCTN 2	INSERT (insert a character)
FCTN 3	ERASE (erase what's on the line)
FCTN 4	CLEAR (stop the BASIC program at whatever statement it's currently running)
FCTN 5	BEGIN (used by some programs)
FCTN 6	PROCEED (used by some programs)
FCTN 7	AID (used by some programs)
FCTN 8	REDO (used by some programs)
FCTN 9	BACK (used by some programs)
FCTN =	QUIT (stop everything and return to the main title screen)

You will notice that the CTRL key has a *red dot* on it. This corresponds to the red dot marked row on the *overlay* that slides into the slot above the number keys. Programs that use the CTRL key usually provide a special overlay for you to use. When you hold the CTRL key down and press one of the number keys, you get what's shown in the red dot line of the overlay you are using.

1.4 GETTING INTO IT

Now that you have got your computer up and running, where can you get information to help you use your TI 99/4A more effectively? From an amazing number of places.

1.4.1 The International 99/4 User's Group

You can join the *International 99/4 User's Group*. Their address is listed in Appendix A. Write to them for more information. It's really worth the $12.00 per year.

The User's Group prints a *newsletter* that tells you what other users are doing, has reviews of new products, and gives you programming tips. There are about eight newsletters per year. They also send out new product announcement bulletins whenever significant new products are announced. So you will always know what to expect.

The User's Group has a wonderful *Software Exchange Program* for Owner Written and Translated Software. Many TI 99/4A users have sent them programs in TI BASIC, TI Extended BASIC, 9900 Assembler, or TI LOGO. You can get copies of these programs on tape, disk, or paper for a modest price. If *you* send them programs—working programs, that is—you get free any four programs in their library.

The User's Group has an *information and referral service* that helps you get answers to hardware and software questions. You call and get an answer right away.

And, on top of all that, the User's Group offers *discounted prices* on all Texas Instruments products. How can you go wrong?

1.4.2 Local User's Groups

Wherever there are computers, you will find users banding together into *local user's groups*. These user's groups can be a wonderful source of information, both on programming and on products.

There is usually a mix of novice, intermediate, and advanced users in any group. The more knowledgeable users in the group are always ready to help and teach those who know less. If you want to learn

more about your TI 99/4A, joining a local user's group is an easy, inexpensive, way to do it.

Check in your area for a TI 99/4A User's Group. The International 99/4 User's Group has a list of local groups that you might want to get. Usually, the computer stores in your area know of at least one user's group. You may even want to start one.

1.4.3 Magazines

Another wonderful way to learn more about your TI 99/4A is through *99'er Magazine*. The address is listed in Appendix A. You may find the magazine at some newsstands.

99'er Magazine describes itself as "Covering the TI 99/4A and other Texas Instruments Personal Computer Products." And it does, wonderfully.

Every month you will get a well-written magazine that discusses only the TI 99/4A and other TI computers. So you will get information geared to your computer, not to home computers in general. There are articles for novices and for experienced programmers. And there are always several programs that you can enter and run yourself.

Other home computer magazines also cover the TI 99/4A, though not in as great detail as *99'er Magazine*. Look over the selection at your newsstand and pick those you feel comfortable reading. Appendix A lists some home computer magazines that have covered the TI 99/4A.

Don't think that you have to find magazines that write directly about the TI. You will learn a lot from reading about what computers can do. And a lot of free programs are sitting around in these magazines, just waiting for you to enter them into your computer. Entering programs from magazines is a great way to learn more about your computer and how BASIC works. Just remember to "translate" from whatever BASIC to TI BASIC when you enter the programs.

1.4.4 Education

Some people will want to learn more about programming. There are good BASIC and other *courses* available directly from TI. You can also try local computer stores or community colleges for courses in BASIC, LOGO, and other popular languages.

We already discussed the learning opportunities available from joining a local TI 99/4A *user's group*.

There are *books* that teach BASIC programming that you might find helpful. Go to your local bookstore and browse. See which ones you feel comfortable with and try them. When you review them, look for many examples. It's easiest for a new programmer to learn from examples.

You can learn a lot from *friends* who already have TI 99/4As or who know how to program (even if they have some other type of computer). They will be able to help you over some of the rough spots you hit when you are just starting out.

2

HOW DOES YOUR TI WORK?

In this chapter, we tell you something about how the TI works, how it thinks about your programs and data, and a little bit about thinking like a computer.

2.1 HOW MUCH DO YOU HAVE TO KNOW ABOUT HOW A COMPUTER WORKS?

You don't have to understand how the TI 99/4A works in order to use it. If you don't care one little bit (heh, heh—that's a sample of bad computer humor for you) how it works, don't read this chapter.

Many people who buy the TI 99/4A would like to know at least something about what goes on inside the mysterious little box. There is a widespread belief that this knowledge is so arcane as to exclude ordinary mortals entirely and forever from obtaining it. Don't believe a word of it. This is nothing more than propaganda put out by "experts" who confuse everyone by using jargon words they don't explain.

In the sections that follow, we will define some common computerese terms and tell you something about what is inside your TI 99/4A, what those magic words—memory, RAM, ROM, 16K—mean, and why you would want to count in binary or hexadecimal.

2.2 MEMORY

The memory we are talking about is the type that comes on little silicon chips inside your TI 99/4A. There is another kind of memory,

sometimes called mass memory or external storage, that lives outside the TI 99/4A—on a cassette tape or a disk.

The first question most people ask about a home computer is "How much memory?". Those who are a little more knowledgeable also ask "What kind?". So we will answer those questions right away. Your TI 99/4A console has:

16K of RAM and 26K of ROM

There, isn't that enlightening? It is *if* you know what memory is, what a "K" is, what "RAM" is and what "ROM" is. If you don't, it's just so much gibberish.

2.2.1 Ks, RAMs, and ROMs

A "K" is a measurement of the amount of memory you have in your computer:

1K = 1024

Why this funny number? you might ask. Well . . . it's because 1024 is 2 raised to the tenth power (as in 2^{10}). Computers deal naturally in numbers that are powers of 2, even if people don't.

"K" was chosen because it's close enough to 1000 not to matter and 1000 is a number that people can get a handle on pretty easily. Anyway, if you think of a K as being about 1000, you will be doing the same thing computer professionals do.

Now you know. A K is about 1000. So what are you counting? You are counting *bytes,* where:

1 byte = 1 character

Thus, 1K of storage holds a little more than 1000 characters (letters, digits, and punctuation marks) or somewhere around 40 lines of "average" BASIC code.

Obviously, the more Ks you have, the larger and more complex the programs you can run. To be useful to you, those Ks must be RAM.

There are two types of memory, *RAM* and *ROM*. These words are mnemonics:

ROM is *R*ead *O*nly *M*emory
RAM is *R*andom *A*ccess *M*emory

ROM *cannot be written to.* That is where it gets its name—Read Only Memory. It already has a program permanently stored in it. The

26K of ROM in the TI 99/4A holds, mostly, the TI BASIC interpreter (more about that later) and a thing called the *operating system.*

RAM can be both *written to and read from.* The 16K of of RAM in your TI 99/4A is used to hold your programs and other changeable items, like what is displayed on the screen, or the color of the characters. Unlike ROM, RAM forgets everything when you turn off the power.

If you add a Memory Expansion card to your TI 99/4A, you get an additional 32K of RAM for your programs to use. If you add the Mini-Memory Module, you add another 4K of RAM.

2.2.2 Bits and Binary

Memory comes in smaller pieces than a byte. The fundamental unit of memory is the *bit* (*b*inary dig*it*):

1 byte = 8 bits

What is a bit? Physically, a bit is a small cell on a silicon memory chip. That small cell either has an electrical charge, or does not have a charge. If it has a charge, it is "on," and if it does not, it is "off."

Therefore, a bit can be one of two things: on or off. We write these states like this:

an "on" bit is written as a 1
an "off" bit is written as a 0

Ultimately, everything in every computer in the world is reduced to bits and to on or off, 1 or 0. You can't do much with a single bit, but you can do a great deal with a lot of them. In the console RAM of your TI 99/4A you have a lot of them:

131,072 bits of memory

2.2.3 Counting

What do you do with bits? You can, for example, count. *You* know how to count. From before calculators, remember:

0 1 2 3 4 5 6 7 8 9 10

When you count this way, you are counting in the *decimal* system, which is based on 10 unique digits, 0 through 9—amazingly enough the same as the number of fingers you have.

But suppose you are a computer. You don't have 10 unique digits, you only have two—0 and 1. How do you count then? The same way as with 10. You just write it a little differently:

0 1 10 11 100 101 110 111 1000 1001 1010 binary
is the same as
0 1 2 3 4 5 6 7 8 9 10 decimal

Binary counting is based on *two,* just as *decimal* is based on *10.* For example, suppose you count the fingers on one hand and write the result in both binary and decimal:

101 fingers binary
or
5 fingers decimal

No matter what the numbers look like, you have the same number of fingers. It does not make much difference whether you write the number in binary or in decimal.

Of course, once you count something in binary, you can easily (if you are a computer) convert it into decimal for printing or to display on the screen.

2.2.4 Making Characters

The other thing you can do with bits is *make characters.* Characters are letters, digits, and punctuation marks that appear on your screen. Each *character* occupies *one byte,* or *8 bits.* The 8 bits are set to a unique on/off pattern for each character, much like Morse code has a unique set of dots and dashes for each letter.

The choice of which pattern to use for which character was decided, rather cleverly, and published as the *ASCII character codes.* In Table 2-1, you will see that you can look at each character either as a bit pattern *or* as a numeric value.

We said the ASCII codes were cleverly constructed. Isn't it nice that the numeric values of the letters correspond to their order in the alphabet. This makes it very easy to arrange (sort) character data into alphabetic order; you simply compare the numeric values of the character codes.

2.2.5 More Counting—Hexadecimal

Although the computer actually thinks in binary, binary is not a very convenient way to write numbers. It's an awful lot of ones and zeros to write and you could very easily make a mistake. Not to mention trying to read a number.

We could use decimal—and usually do—instead, but decimal numbers don't fit very well on an 8-bit binary base. Look at these examples of 8-bit binary numbers:

11000101 = 197 decimal
or
00011111 = 31 decimal

As you can see, there is no natural correspondence between a binary number and its decimal counterpart. Since a fundamental unit of your computer's memory is the 8-bit byte, it would be nice to have a compact way to write numbers so that they correspond directly to the bits in the byte.

That's done with *hexadecimal* numbers. You have already seen that binary is a numbering system based on 2. Hexadecimal is a numbering system based on 16. The binary system has two digits; the decimal system has 10 digits; and the hexadecimal system has 16 digits.

Hexadecimal digits begin with the familiar 0 to 9. But that's only 10 digits—we need 16. The originators of the hexadecimal system *could* have invented six entirely new characters, never seen before. Fortunately they were not that stupid. To maintain compatibility with existing hardware—like printers for example—they chose to use the letters A to F to represent the new hexadecimal digits. Here's the way you would count in hexadecimal:

0 1 2 3 4 5 6 7 8 9 A B C D E F hexadecimal
is the same as
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 decimal

So what? you might well ask. Well, it turns out that a single hexadecimal digit can represent any 4-bit value, like this:

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F

One *hexadecimal digit* represents a *4-bit* value, called a *nibble*. (What else would you call half a byte?) It follows that any *two* hexadecimal digits can represent the value of an 8-bit byte. Our previous decimal/binary examples become:

11000101 = C5 hexadecimal
00011111 = 1F hexadecimal

Table 2-1 ASCII Codes for Characters

Character	Bit Pattern	Decimal Value	Hexadecimal Value
(space)	00100000	32	20
!	00100001	33	21
"	00100010	34	22
#	00100011	35	23
$	00100100	36	24
%	00100101	37	25
%	00100110	38	26
'	00100111	39	27
(	00101000	40	28
)	00101001	41	29
*	00101010	42	2A
+	00101011	43	2B
'	00101100	44	2C
−	00101101	45	2D
.	00101110	46	2E
\	00101111	47	2F
0	00110000	48	30
1	00110001	49	31
2	00110010	50	32
3	00110011	51	33
4	00110100	52	34
5	00110101	53	35
6	00110110	54	36
7	00110111	55	37
8	00111000	56	38
9	00111001	57	39
:	00111010	58	3A
;	00111011	59	3B
<	00111100	60	3C
=	00111101	61	3D
>	00111110	62	3E
?	00111111	63	3F
@	01000000	64	40
A	01000001	65	41
B	01000010	66	42
C	01000011	67	43
D	01000100	68	44
E	01000101	69	45
F	01000110	70	46
G	01000111	71	47
H	01001000	72	48
I	01001001	73	49
J	01001010	74	4A
K	01001011	75	4B
L	01001100	76	4C
M	01001101	77	4D
N	01001110	78	4E
O	01001111	79	4F
P	01010000	80	50
Q	01010001	81	51
R	01010010	82	52

Table 2-1-(Cont.) ASCII Codes for Characters

Character	Bit Pattern	Decimal Value	Hexadecimal Value
S	01010011	83	53
T	01010100	84	54
U	01010101	85	55
V	01010110	86	56
W	01010111	87	57
X	01011000	88	58
Y	01011001	89	59
Z	01011010	90	5A
[	01011011	91	5B
\	01011100	92	5C
]	01011101	93	5D
^	00101011	94	5E
_	01011111	95	5F
`	01100000	96	60
a	01100001	97	61
b	01100010	98	62
c	01100011	99	63
d	01100100	100	64
e	01100101	101	65
f	01100110	102	66
g	01100111	103	67
h	01101000	104	68
i	01101001	105	69
j	01101010	106	6A
k	01101011	107	6B
l	01101100	108	6C
m	01101101	109	6D
n	01101110	110	6E
o	01101111	111	6F
p	01110000	112	70
q	01110001	113	71
r	01110010	114	72
s	01110011	115	73
t	01110100	116	74
u	01110101	117	75
v	01110110	118	76
w	01110111	119	77
x	01111000	120	78
y	01111001	121	79
z	01111010	122	7A
{	01111011	123	7B
|	01111100	124	7C
}	01111101	125	7D
~	01111110	126	7E
(DEL)	01111111	127	7F

Remember we said the *value* of the byte was important in some circumstances while the *pattern* of the bits was important in others. With hexadecimal, we have a method to express both the value and the pattern of a byte in a simple two digit number.

If you do any work with re-defining the image of characters or with sprites (moving graphics objects), you must be familiar with hexadecimal numbers in their pattern indentification role. Chapter 7 talks about graphics in more detail. (To re-define characters, you use the CHAR subprogram in BASIC.)

2.3 THE CENTRAL PROCESSING UNIT

Inside every computer lives a brain called the *Central Processing Unit* (CPU). This is the part of your computer that actually does the work, faithfully carrying out your instructions like a dear, old, feeble-minded retainer.

The most important thing for you to remember about the CPU is *it's a moron*. It does *nothing* you don't tell it to do. And it does *everything* you do tell it to do. No matter how ridiculous that may be.

If you have written a program (those little examples from the book don't count), you will understand this. For those who haven't, we will give you an example. If you are writing a tax program and you tell the computer to subtract your personal deduction from your birth date, it will do it without a complaint.

2.3.1 The TMS9900 in Your TI 99/4A

The CPU in your TI 99/4A is a very sophisticated *16-bit microprocessor* called the TMS9900. The TI 99/4A and its predecessor, the TI 99/4, are the first home computers in their price range to include 16-bit mircroprocessors.

The advantage of a 16-bit processor is not apparent unless you are programming in Assembly Language. For those who know something about this, the TMS9900 processor features:

- 16 general purpose 16-bit registers
- 5 very flexible addressing modes
- 16-bit hardware multiply and divide

If you don't know what this means, don't worry about it. These are standard features in most 16-bit microprocessors, but completely unheard of in the world of 8-bit microprocessors most frequently found in home computers.

So it's fancy. What does it mean to you if you don't intend to program in Assembly Language? It means that software developers who have to program in Assembly Language will like doing it on the TI 99/4A, and so produce a pile of nice, useful, fun programs for you.

2.4 ADDRESSING

Addressing is what a computer does when it *references* something in its *memory*. There is a limit to the amount of memory a computer can access and that limit depends on the number of bits it uses to count memory locations. The TI 99/4A uses 16 bits to count memory.

Memory is arranged into a continuous, long string of bytes. Each byte has a unique number—called an *address*—assigned to it (much like each apartment in an apartment building has its own number). For the TI 99/4A, addresses range from:

0 to 65,535 decimal
or
0000 to FFFF hexadecimal
or
0000000000000000 to 1111111111111111 binary

This is a range of 64K bytes. Now if you add up all the memory you can hang on your TI 99/4A, including that in cartridges, you will find it amounts to a lot more than 64K.

Take, for example, Extended BASIC. The cartridge contains 36K of memory. It can directly use 24K of the 32K Expansion Memory and about 12K of the memory in the console. On top of that, it has 8K (the balance of the 32K Expansion Memory) to use for running Assembly Language routines. That comes to 80K and does not even count operating system support routines and device handlers in your TI 99/4A console memory.

To fit all that into 64K, computer designers use a trick. They keep some of the memory "off to the side." When it's needed, it's brought in and used. When it's not, it's left off to the side. You don't really have to understand how this is done; you just have to know that it *is* done on your TI 99/4A.

2.4.1 Program Cartridges and Memory

Those cartridges, from TI and other sources, that plug into the console slot usually contain machine language programs embedded in ROM (Read Only Memory). One exception is the Mini-Memory Car-

tridge, which contains 4K of RAM (Random Access Memory) in addition to 10K of various sorts of ROM.

When you plug in one of these cartridges, the memory in it becomes part of the TMS9900 microprocessor's memory at addresses 6000 to 7FFF hexadecimal (24,576 to 32,767 decimal). That's 8K (8192) bytes of memory.

In addition to this directly addressable TMS9900 memory, cartridges often contain quantities of GROM (Graphics Read Only Memory). GROM is one of those "off to the side" memories that the 9900 can access only through some special procedures. (For those of you who may be interested, and who may know something about this, GROM is accessible as a memory mapped device.)

2.5 TMS9900 ASSEMBLY LANGUAGE

A microprocessor can only understand instructions written in its own language—*machine language*. Machine language is written in binary (ones and zeros) and is just about impossible for any person to understand.

The machine language instructions that a microprocessor understands are called it's *instruction set*. The TMS9900 has a powerful instruction set that makes it easier to program than some other microprocessors. To make it easier for people to use the power of machine language instructions without the incredible task of talking in ones and zeros, computer designers developed Assembly Language. And the TMS9900 has a very rich language.

Assembly Language programs are converted into machine code (or *object code*) by a program called the *Assembler*. There are two Assemblers for the TI 99/4A. There is the full Editor/Assembler package that requires expanded memory and a disk drive and there is the mini-Assembler that comes in the Mini-Memory cartridge. If you would like an introduction to Assembly Language, or if you want to write some small Assembly Language subprograms, the Mini-Memory cartridge is the least expensive way to go.

We are not going to try to teach you 9900 Assembly Language. That needs a whole book by itself. We do not want to give you some idea of what an Assembly Language program looks like, so we have included this little example of TMS9900 Assembly Language. This example shows you how to add two 16-bit signed numbers together:

```
NUMI   DATA   >0068          first number to add
NUM2   DATA   >00CC          second number to add
       A      @NUM1,@NUM2    add the value at NUM1 to
                             the value at NUM2 and put
                             result in NUM2
```

NUM1 and *NUM2* are *labels*. Labels have a value equal to the address of the statement they are attached to.

DATA is an *Assembler directive* that tells the Assembler program to reserve a two byte area with a value stored in it. In this example, the values are >0068 and >00CC hexadecimal, or 104 and 206 decimal. The ">" sign indicates the number that follows is a hexadecimal value.

A is the *add instruction*. It tells the computer to add the value *at* (@) label NUM1 to the value at label NUM2 and place the result at NUM2.

This is a very simple example of Assembly Language programming, really only a fragment of an Assembly Language program. If you want to do something ambitious, like write to a cassette tape, it becomes a lot more complicated. Many things that BASIC does for you automatically, you have to do for yourself when you program in Assembly Language.

Still, there's a definite feeling of command of the machine and its substantial resources when you succeed in getting an Assembly Language program to run. If you want to get into the nitty-gritty of it all, Assembly Language, for all its problems and frustrations, is how to get there.

2.6 DEDICATED CHIPS

The TI 99/4A console actually contains *three processors*. We have already discussed the general purpose TMS9900 microprocessor. The other two processors are called *dedicated chips* because they perform *only one task:*

- the TMS9918A is the Video Display Processor.
- the TMS9919 is the Sound Generator Controller.

The *TMS9918A* spends all its time maintaining the picture you see on your screen. In fact, it is this chip that gives the TI 99/4A its name. The older TI 99/4 was equipped with the TMS9918 Video Display Processor, which lacked some of the features of the newer TMS9918A.

Maintaining the screen image on any computer system is a full time, specialized, job. We examine the TMS9918A in more detail in Chapter 7, Graphics.

Just as maintaining the screen is a specialized job, so is making sounds. So, the *TMS9919* Sound Generator Controller is included in the console. We talk more about this chip in Chapter 8, Sound.

Some of the peripherals that you buy also contain dedicated chips. Most notable of these is the *TMS5200 Voice Synthesis Processor* in the Speech Synthesizer peripheral.

2.7 PROGRAMS

Not all programs are equal. Some are directly executed by the TMS9900 microprocessor while others are only indirectly executed.

Those directly executed by the 9900 microprocessor are *machine language* programs. You can create a machine language program with the Editor/Assembler, the Mini-Memory Module, or with a *compiler* like FORTH.

These products all produce *object code* (machine language instructions) that can be loaded and run with no other program between it and the 9900 microprocessor (see Fig. 2-1).

You will notice that BASIC is not in this group. That's because BASIC is an *interpreted language,* as is Pascal, PILOT, and LOGO (see Fig. 2-2). We will talk mostly about BASIC here, but the others are very much the same.

BASIC programs are stored in memory in a form very close to the way they appear on the screen. The major difference is that the BASIC keywords are *tokenized.* This means that rather than storing the full text of the keyword, like GOTO, the BASIC editor turns it into a one- or two-byte hexadecimal value. The GOTO keyword, for instance, is stored internally as the two-byte hexadecimal sequence 86 C9 so that the statement:

```
800 GOTO 720
    becomes
0320 86C9 02D0 00
```

The 0320 is the line number of the statement (800); the 86C9 is the tokenized form of the GOTO statement; 02D0 is the line number to go to (720); and the 00 is an end of statement indicator.

Most of the variable names and constants in a statement appear similar to the way you typed them. Obviously, this stuff cannot exe-

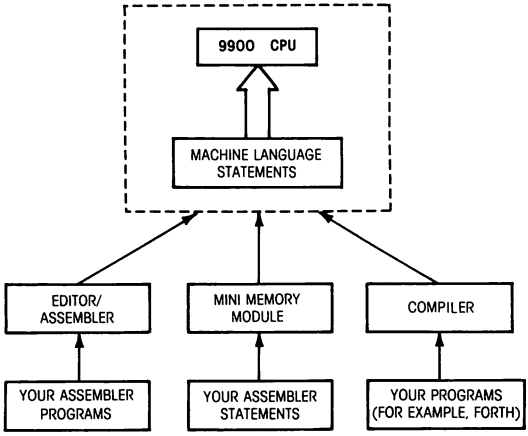


Fig. 2-1. Object code diagram.

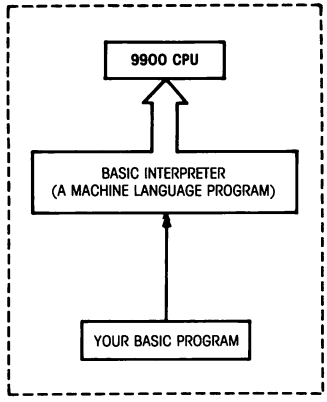


Fig. 2-2. Interpreter diagram.

cute directly on the 9900 CPU—it's not even close to machine language.

The BASIC program is actually executed by another program called the *TI BASIC interpreter*. Your TI 99/4A includes 26K of ROM, a good part of which is taken up by the TI BASIC interpreter.

The TI BASIC interpreter fetches your coded BASIC statements from memory, decodes them, and does what they say. It is also an editor, allowing you to enter and change BASIC programs, and a file handler with facilities for saving and retrieving programs stored on cassette or disk.

Table 2-2 BASIC/Assembler Comparison

BASIC	Assembler
ERROR HANDLING	
Traps all errors and prints an error message to the screen. Lets you examine your variables, look at and change your program, and resume execution of your program from the statement where it stopped.	You trap and diagnose your own errors. There is no easy way to look at the value of your variables. Because of the nature of Assembler programs, it's easy to write all over critical memory areas and not discover the destruction until later in program execution.
IMMEDIATE EXECUTION OF STATEMENTS AND COMMANDS	
When you enter a statement or command without a line number, it is interpreted and executed immediately.	All statements must be processed through the Assembler, then loaded and run separately.
DATA CONVERSION	
Conversions of data from character to numeric or numeric to character format are handled automatically (in PRINT and INPUT statements) or through functions (CHR$ and VAL).	You must set up and perform the conversions yourself. Some support routines are available, but you must carefully arrange the arguments to them. Error handling in the support routines is very limited.
MEMORY MANAGEMENT	
All management of memory resources is handled by the interpreter. String data, for example, is allocated as it is needed. If you run out of memory, the interpreter does a "garbage collect" to recover any memory not in active use.	You must manage memory use yourself. If you are doing computations that require large arrays of numbers, you can save a considerable amount of memory by using integer data types. Integer numbers take only two bytes of memory, as opposed to eight bytes for floating point numbers, but they must be whole numbers in the range −32,768 to 32,767.

Table 2-2-Cont—BASIC/Assembler Comparison

BASIC	Assembler
DEBUGGING SUPPORT	
It's comparatively easy to debug (find the errors in) a BASIC program. When an error occurs, the interpreter gives you the number of the statement that failed. You can PRINT the content of variables in the statement, or any other variables in your program. You can change variable values and CONTINUE execution of the program, or you can edit the program to correct the error and immediately re-execute it. To help you find a logic error, BASIC supports a TRACE command that prints each statement number before the statement is executed.	It's often difficult to isolate the cause of an error in an Assembly Language program. Assembler supports a run-time debugging utility but it requires quite a lot of knowledge to use. When you do find the error, or what you think is the error, you cannot just edit a line and re-run it right away. You must edit the original Assembler source language file, re-assemble it and only then see whether you did fix the error.
EXECUTION SPEED	
BASIC is relatively slow. It is, after all, a program running another program. It does a lot for you, looking at every statement as it is executed to make sure it is correct, and making the details of the TI 99/4A machine environment invisible to you.	Executing in native 9900 machine code, well-written Assembler programs run *very* quickly. In some situations, an Assembler program can execute as much as 2000 times faster than a comparable BASIC program. This is a good reason to write in Assembler, or another compiled language. Some programs simply require execution speed.
FILE HANDLING	
The BASIC statements OPEN, PRINT, and INPUT make tape, disk, printer, and RS-232 file access easy. You don't need to know much about these devices in order to use them from BASIC. On the other hand, there are files that BASIC cannot read. BASIC requires its files to be formatted in a special way; if they are not, you get a processing error.	File handling is much more complex, requiring you to build and maintain control blocks (PABs) and logical record buffers. You must also look for and handle any file errors that occur.
ACCESS TO ALL TI 99/4A FEATURES	
You cannot use all of your TI 99/4A's potential in a BASIC program. You cannot, for example, use the full range of graphics capability, or the high speed integer arithmetic available in the 9900 microprocessor.	Obviously, you can do everything the TI 99/4A is capable of doing.

Although it's much slower executing than a machine language program, there are many things you can do with BASIC that are impossible to do with an assembled or compiled language. For example, you can enter a BASIC statement with no line number and it is executed immediately. This is possible only with an interpreted language. Table 2-2 contains a comparison of some of the features of the BASIC interpreter and the Assembler.

In general, compiled and assembled languages require you to know much more than do interpreted languages. This is especially true in the area of program debugging.

3

SOFTWARE: MAKE IT OR BUY IT

For the beginner and expert alike, software is the most mysterious and difficult part of any computer system. The TI 99/4A is no exception.

In this chapter, we look at the options you have for finding software that does what you want and getting it up and running on your TI 99/4A.

3.1 WHAT IS SOFTWARE?

Software is the instructions, sometimes called programs, that tell your computer what to do. Although it cannot be seen or touched, software is *the most important part* of any computer system. Without instructions to get your computer to do what you want, all you have got is a pile of highly refined sand.

Software is classified into two broad categories:

- *Operating system software* controls the entire machine environment, providing uniform access to and control of peripherals, such as disks, tapes, printers, the keyboard, and the screen.
- *Applications software* is what you buy or write to make your computer do what you want. Word processors, instructional programs, and arcade games are all examples of applications software.

Many business microcomputers are supplied with an operating sys-

tem that must be loaded from disk before the computer can be used to run applications software.

In contrast, most home computers, including the TI 99/4A, supply an operating system that is built into the computer in ROM. The advantage of an operating system in ROM is that you don't need to buy a disk to use the machine. Simply turn it on and it's ready to run.

With the TI 99/4A, you have a choice only when it comes to applications software. The *Question* is: should you make your own software (write it in BASIC or another language), or should you buy packaged software to do the job you want done?

3.2 MAKING YOUR OWN

You probably bought your TI 99/4A assuming you would learn how to write programs in BASIC.

If you have not, or don't intend to, learn programming (or if your 7-year-old has humiliated you so badly that you will never try again), you obviously are not going to make any software. You folks can proceed right to the section on buying it.

For those of you who can program, who think you can program, or who would like to learn, take note. Writing small, special purpose programs, entering and running programs from books, and fiddling with these programs is a heck of a lot of fun. Writing a major and complex system, with ambitious goals, is hard and lengthy work. (Why do you think those programmers make so much money?)

3.2.1 Where to Begin

Regardless of your real (or imagined) level of proficiency, any program that you can copy out of a book is an excellent choice.

Is this programming? you say. Yes. For the novice programmer, it is instructive; for the experienced programmer, it is fast and less wearing than starting from scratch.

As you enter the program, you learn how it works. This is a good way to discover programming techniques and tricks that you might never learn on your own. This applies equally to experienced, even professional, programmers.

Study the program as you enter it. Critique it. See how it was designed (put together). Did the author use meaningful variable names? Is the thing a tangle of GOTO statements (GOTO statements make a

program more difficult to follow)? Do you see other, perhaps better, ways of doing some of the things it does?

The most important thing to look for, though, is ways to expand it, add features to it, make it do *exactly* what you want it to do. This is where the learning stops and the programming begins.

If you are a beginning programmer, modifying a program that you know works is the best way to learn. If you make a change and the program no longer works, you can be pretty certain why it does not. And you will know which statements have the highest probability of containing errors—those you changed.

When you make any changes, no matter how trivial, to a program, you can introduce errors that are either *syntax errors* or *logic errors*. Syntax errors are statement format errors and are easy to find. Your computer tells you what it does not like. Logic errors, on the other hand, often require a bit of detective work on your part to find out why the program's suddenly not doing what *you* want it to do.

3.2.2 What Should You Program?

As a general rule, you should program only what you cannot find already programmed, but want desperately. For example, if you need to keep track of the player statistics of all of the pitchers in the National League with batting averages over 300, you can write a program in TI BASIC to do that.

If, on the other hand, you need an object-code-generating Extended BASIC compiler, forget it. Even if you know what an object-code-generating Extended BASIC compiler is, it would take a first class Assembler programmer years, full time, to write it.

You should also consider the amount of time and effort involved in designing, writing, and debugging even a relatively straightforward application program. On your first attempt to write your own program, you will discover how stupid your computer really is. The darn thing does *exactly what you tell it to do*—not what you *meant* to tell it.

The important point is to pick projects that are manageable, given your level of knowledge and experience and the amount of time you have to work on them.

3.3 WHAT YOU NEED TO WRITE YOUR OWN

The TI 99/4A computer (console) contains all you really need to write simple programs for yourself. This implies that you don't need

to save the program once you run it. You may have copied it from a book and so can re-enter it if you ever want to run it again. Or perhaps it has served its purpose and you no longer need it.

You can store your programs—whether copied or created—onto the cassette for later recall, providing the programs are not very large and don't require significant amounts of data. Cassettes are fairly reliable (if you have a decent cassette recorder) and cheap, but they are slow.

3.3.1 Minimum System

Working with just the console and a cassette recorder is possible for writing short programs or for *copied* programs many screens (a screen is 24 lines of 28 characters) long. However, if you are creating your own programs, or making major modifications to an existing program, you are going to want a *hardcopy* (a printed listing of the program).

It simply is not possible to debug (get running without errors) any very long program—say, more than 6 or 7 screens—without a printed listing of the program. You cannot really see how the parts of the program interact when you are looking at the program one screen at a time. Of course, the actual limit depends on whether you copied most of the program out of a book (you can look there for some of the listing), on how complicated the program is, and on how much experience you have.

To get hardcopy, you need a *printer*. Fortunately, the TI 99/4A allows you to use almost any printer available in the marketplace (we talk about expanding your system in Chapter 5). Of course, a printer is not limited to printing program listings. Anyone doing anything very substantial will probably need to print the results from the program anyway.

So, the bottom line for doing anything at all complicated is:

- The TI 99/4A console
- A cassette recorder (to store programs and data)
- A printer (to list your programs and data)

3.3.2 More Ambitious Programming

The more ambitious you become, and the less time you have to spend, the more you will begrudge the limited memory available in the console and the fumbling and delay involved in using a cassette.

The 16K of memory in the console seems like a lot when you get

it—and indeed it is. But some programs just cannot run in 16K. They may require large data arrays, or maybe they are just very long and complicated programs. The only answer to this is to get the 32K Memory Expansion card for your Expansion Box (take a look at Chapter 5 for more information on this).

Of course, it's not as simple as that (it never is). The console TI BASIC does not know how to use the extended memory. For that, you need Extended BASIC (or another of the advanced languages available for your TI 99/4A).

This is not as bad as it seems. Extended BASIC is a genuine advance, not only over TI BASIC but also over *any* BASIC available on any low-cost home computer. Extended BASIC is an excellent programming language and we talk more about it in the next section and in Chapter 4.

Now that you have the memory to do the job, you will want to cut the delay and inefficiency inherent in using cassettes. The only way to do that is to get a disk drive. Fortunately, since you have already got the Expansion Box, this is not outrageously expensive, only mildly so. And worth every penny of it.

The disk drive offers three main advantages over the cassette:

1. Data or program transfer from the TI 99/4A to a disk is about 30 times faster than the same transfer to a cassette.
2. Files (programs) on a disk are directly, or randomly, accessible. No search through a tape, no keeping track of tape counter numbers, no endless fast-forwards. All you do is plop the correct disk in and seconds later you are running your program.
3. If you are working on a large program or if you have a lot of data for a program, you should regularly make copies to protect yourself from accidental destruction of the original. This is so cumbersome using cassettes that you find you don't do it. Even with only one disk, backing up (making security copies of) key program and data files is easy and takes only a few minutes.

As with any job, proper tools make the work easier, faster, better, and more enjoyable. This applies doubly to programming—proper tools not only make it easier, faster, better, and more enjoyable, they make it possible. For those with advanced programming ambitions, you will need:

- Memory Expansion card (so you will have 48K for your programs and data)
- A printer (for program and data listings)

- A disk drive and controller (for easier and faster storage of program and data)
- An advanced language processor (i.e., Extended BASIC)

3.4 THE LANGUAGE OF CHOICE

Most of you will undoubtedly use BASIC in all the programming you do. There are some places, though, where your TI 99/4A offers you a better, or more appropriate, choice depending on the application you are going to program.

As we write this, the TI 99/4A supports the following languages:

- TI BASIC
- Extended BASIC
- UCSD Pascal
- PILOT
- LOGO II
- FORTH
- TMS9900 Assembly Language

Because the 9900 microprocessor in the TI 99/4A is a powerful 16-bit chip, other languages are likely to appear (it's easier, and more rewarding, for software developers to write language processors for powerful machines).

We have a review of these languages in Chapter 6 where we discuss their strengths, weaknesses, and appropriateness to a given task. Here, we will tell you what's available to help you in your programming.

3.4.1 BASIC

TI BASIC and *Extended BASIC* are suitable for most common programming on the TI 99/4A. If you need a little more help, you can get the Programming Aids I, II, and III BASIC language assist packages from TI. These give you things like a sort, file dump, and merge that make it easier to manage and debug programs.

3.4.2 LOGO II

LOGO II is something else altogether. A delightful introductory programming language developed at MIT, LOGO uses graphics and sound to effortlessly introduce children to the concepts of programming and logic. LOGO is so entertaining, as it instructs, that children

sit for hours manipulating "turtles" and other moving objects (sprites) on the display.

Although it was intended for young children, it's not bad for computer-shy adults either. It not only teaches computer concepts, but also teaches logic, geometry, and determined problem solving.

LOGO is not, however, a general purpose programming language. It's great for graphic displays, but no good at all for balancing your checkbook.

3.4.3 Assembly Language

The final choice, of course, is *TMS9900 Assembly Language*. Why bother with Assembly Language? It's difficult to learn, tedious to write, a pain to debug, and requires you to know a great deal about how the TI 99/4A actually works.

You bother with Assembly Language because it's *fast*. Some applications, like many arcade games, absolutely require fast response from the program.

BASIC and Pascal, two other general purpose languages available on the TI 99/4A, are *interpretive*. This means that the code you write is not translated directly into 9900 machine language. Instead, it is translated into an intermediate format that is *interpreted* by the language processor as the program executes. This puts another level between your program and TI 99/4A machine language, slowing execution significantly.

You may also need access to some facilities of your machine that are not available through BASIC or Pascal. Some graphics functions, for example, are accessible only from Assembly Language.

And, of course, there is space. An Assembly Language program is much more *compact* than a corresponding BASIC or Pascal program. An Assembly Language program can use as much as 52K of RAM—16K in the TI 99/4A, 32K in the Memory Expansion card, 4K in the Mini-Memory Module—for program and data storage, significantly more than BASIC or Pascal. (Remember, you use Extended BASIC from a cartridge. You cannot use both the Mini-Memory cartridge and the Extended BASIC cartridge at the same time.)

Despite these advantages, going 100% Assembly Language is only for the dedicated. But, you have an alternative. Extended BASIC and Pascal both allow you to link to subroutines written in Assembly Language. Thus, you can obtain many of the advantages of Assembly Language, while retaining the ease-of-programming and debugging support of BASIC or Pascal.

3.4.4 Pascal

UCSD Pascal, which requires the Pascal P-Code card in your Expansion Box, is an industry-wide standard Pascal language interpreter. So what? you might ask. The wonderful thing about UCSD Pascal is that it is totally transportable. You can take a UCSD Pascal program from IBM's latest mega-mainframe computer with 64 million bytes of memory executing 25 million instructions per second and, with no changes at all, run exactly the same program in your TI 99/4A. (Well, maybe some changes to fit into the smaller memory you have.)

So when should you use Pascal? Well, if you are a Computer Science major, you might need Pascal to do your homework. If you are developing software for sale to a market broader than just the TI 99/4A crowd, UCSD Pascal is a good choice because it's implemented on an enormous number of machines. It also has some very nice support for program development.

But, it's expensive. You may have to buy it to support another package that you want to use on your TI 99/4A. For example, PILOT requires the P-Code card. If you don't need Pascal for something else though, Extended BASIC is a lower cost and quite viable alternative. With the extensions that TI has put into it, Extended BASIC is nearly as powerful as Pascal, makes more direct use of the TI 99/4A hardware, and is easier to learn.

3.4.5 PILOT

PILOT is a special purpose language for creating Computer Aided Instruction (CAI) programs. PILOT is itself written in UCSD Pascal and so requires the P-Code card to run. If you are a teacher, or just interested in computer aided instruction, PILOT is a good choice for generating CAI courses. As with UCSD Pascal, courses created using PILOT are completely transportable to many other machines.

PILOT is not a general purpose programming language. You should use it only if you are interested in Computer Aided Instruction programming.

3.4.6 FORTH

FORTH is a new language from TI targetted firmly on the third party software development crowd. It is a fully compiled language that is translated into 9900 machine code. It is, therefore, an alternative to writing in Assembly Language.

FORTH is an excellent systems development language but it's very

difficult to learn. Use FORTH only if you intend to produce marketable software—and are a very talented programmer.

3.5 PACKAGED SOFTWARE

Purchasing software is certainly the easiest—and for some people, the only—way of getting it. Purchased software comes from *software publishers* in units called *packages*. A single software package can be one or several programs and may include data files.

The applications software packages you can buy for the TI 99/4A consist of a program or programs designed to perform a specific function or set of related functions.

The primary thing to remember about all microcomputer software is: *you run it as it comes*. There is very little you can do to tailor a package to your needs and almost no support from software publishers for any kind of change. This is the only way to keep costs low enough for you to afford their products.

And software for the TI 99/4A is incredibly cheap. Most programs—even those in Command Cartridges—cost less than $100. If you can get the software on cassette or disk, it typically costs from $10.00 to $50.00.

The TI 99/4A has an enormous amount of software available to run on it. More than 1500 software packages are available in

- Arcade games
- Board game simulations
- Thought games (adventure type games)
- Early education
- Secondary education
- Personal finance
- Spreadsheet analysis programs
- Communications
- Word processing
- Graphics
- Science and engineering
- Business accounting
- Mailing list processing

This is only a small sample of the rapidly growing library of software available for the TI 99/4A.

Much of this software is published by Texas Instruments itself. This

software is written either internally or under special arrangement with cooperating software publishers and is generally of very high quality.

The microprocessor in the TI 99/4A (the TMS9900) is a first rate 16-bit microprocessor that offers a powerful instruction set. This does two things. First, it makes the TI 99/4A attractive to software developers because it's easier to write for a powerful microprocessor than for a weak one. And second, it makes it possible to implement things on the TI 99/4A that would be very difficult, or impossible, to implement on less capable microcomputers.

To illustrate this point, consider three highly regarded software systems that are available in TI 99/4A versions:

- The *UCSD Pascal P-Code system,* an advanced design system normally found on business microcomputers and originally created for minicomputer systems.
- *MicroSoft's MultiPlan,* a super-sophisticated spreadsheet analysis program that was named software product of the year by *InfoWorld* (a first class industry newspaper that you can often find in computer stores).
- More than 400 of Control Data Corporation's widely acclaimed *Plato* series of computer aided instructional programs. This software was developed on giant mainframe computers for more than a decade before it was moved to the TI 99/4A.

A number of independent software houses (companies that write programs for sale) are also producing programs for the TI 99/4A. Many of these companies entered the TI 99/4A software market when fewer than 100,000 TI 99/4As had been sold. By the middle of 1983 there were more than one million TI 99/4As out there. This kind of growth will draw many new companies into the TI 99/4A software marketplace.

This is good news for those of us who own TI 99/4As. We will have much more software to choose from—and a little price competition never hurt anyone.

On the other hand, it also means you have to beware of software that has been cobbled together too quickly, with shoddy design and insufficient testing. And, of course, who can forget the great failing of home computer software—crummy documentation (instruction manuals).

3.6 BUYING SOFTWARE

Buying software is always a risky business. There is a lot of good stuff out there, but there is also a lot of trash. You can even, sometimes, make a mistake buying the good stuff (buying a Maserati is not the best choice if all you need, or know how to drive, is a Chevette).

With this in mind, we present the following *Software Acquisition Rules,* whose main thrust is: *look carefully before you buy.*

Rule 1. Decide *exactly* what you want the software to do for you.

If you are buying a game, you want to be entertained. Make sure you will be. If the game is for your child, make sure your child is old enough to understand it, but not too old to be bored by it.

If you are buying something to manage your investments or personal finances, take a look at the size of your holdings or the complexity of your personal finances. Buy a software product that is consistent with your real requirements (don't forget to factor in some realistic growth).

Sometimes a software package comes looking for you. You see it at a friend's house, or in an advertisement on tv, or in a magazine. Don't rush right out and buy it! There may be other packages available that do the same thing, except better, or cheaper. Look around first, compare it to other packages, then decide.

The object here is *get a good idea of your actual needs before you go looking.* It will be much less confusing to you if you do.

Rule 2. Decide *how much you want to spend.*

Great rule, this one. It's very easy to get mesmerized by your friendly neighborhood software salesperson demonstrating this really terrific software package. The thing's got 87 bells and 9 whistles—you pay for every one.

While you are deciding what you want the software to do for you, also consider what it's worth to you to get it done. Most software for the TI 99/4A costs between $15.00 and $200 for each individual package. A full system of integrated programs might cost more than that by the time you add up the separate costs of all the pieces.

Another component of cost, that we talk about in Rule 9, is the additional hardware requirements of some packages. For example, the LOGO language (about $75.00 at discount) requires the 32K Memory Expansion card. That means, at a minimum, you have got to

buy the Expansion Box and the Memory Expansion card—more than $300 at discount prices.

Rule 3. *Read* the reviews.

If you don't already subscribe to (or at least regularly purchase) a variety of microcomputer magazines, start now. These publications run frequent reviews of new or updated software packages. In general, the quality of the reviews is good and the integrity of the reviewers quite high.

If you can locate a review of the software package you are planning to buy, you will often find that most of your work has been done for you.

Rule 4. *Evaluate* and *compare* the capabilities of the software packages you are interested in.

By this time, you have decided the minimum requirements for the software package you want. It's simple then to compare those requirements to the capabilities of a particular package.

You may not always be able to find *exactly* what you need in some application areas. The art is in finding which of the available packages provides *most* of what you need.

Rule 5. Check for any *related software products* that might be of use to you later.

Some individual software packages are a single piece of a larger system of applications packages. For example, you can buy an individual early learning program (as a Command Cartridge) that is but a single entry in an entire spectrum of similar early learning programs. This applies to some games as well, like adventure type games.

Knowing of related products that may be of use to you is especially important when you are trying to decide between two or more very similar software packages. It may just tip the balance in favor of the package with better related products.

Rule 6. It better be *easy to use.*

Ease-of-use must be one of your major concerns in evaluating any software for your TI 99/4A. You will receive little support or training in the use of the software, so it had better be easy to learn and extremely user friendly.

There is often a trade-off between the power and complexity of the package and its ease-of-use. If you genuinely need the power, then you may have to accept a more difficult to use package.

Beware! Powerful packages are not the only ones that can be enormously difficult to use. As you gain software experience, you will be amazed to see how the simplest, most straightforward, applications are approached in the most difficult and obscure manner possible. This is the fault of the package designer.

Avoid packages that make simple things difficult to do.

Now you know you need user friendly software. So, how can you tell? Look for these qualities:

- User friendly software *leads you through the application* in an easy way that anyone familiar with the application—not the software package—can understand.
- User friendly software *never deserts you* in a moment of need. It intercepts errors, explains them in terms you can understand, then offers you a chance to fix the problem (if possible).
- User friendly software *never destroys your data,* and, furthermore, does everything possible to keep you from doing so.

Rule 7. Read the *documentation.*

Good documentation is *essential* for all but the most trivial of packages. Documentation is the bridge between you and your application package. It should be well written, well organized, nicely presented, complete, and appropriate to the intended reader. If it's an educational package for your child, the manual should not baffle you.

Look for examples. You can learn a lot from the examples in a manual, even if the examples are not *exactly* what you need. A marginal manual can be usable if it includes good examples; good manuals always include good examples.

In a manual longer than 30 pages, you should certainly expect to find an index. It should not be one of those keyword-only indexes either—they send you flipping all through the manual looking for what you need. The entries should tell you what you will find at the indicated page.

If the package is not adequately documented, reject it.

The documentation on TI's own packages is pretty good, though not always as complete as it could be. Documentation from independent software publishers is variable. Check carefully, *before* you buy.

Rule 8. *Test run* the software if at all possible.

Many retailers have TI 99/4A systems set up and will let you test any of the software they sell. Try it out. See how you like it.

While you are running it, look for "bugs" (errors in the program). Try it with the kind of data that *you* expect to be using. After you have seen how it runs, try to make the program fail. Give it wrong answers, tell it to do something stupid, and see how it handles the errors.

It's difficult to tell whether a package contains errors. If you know, or can get the names of, people who already have the package, you can ask them whether they have had any problems. As a general rule, new products contain some errors not revealed in testing. A package that has been in use for a year is probably error free.

Rule 9. Find out how much *hardware* the package *needs.*

Hardware requirements for application packages vary widely. Most game cartridges need only the TI 99/4A console to run. The Editor/Assembler, however, requires the Memory Expansion card and a disk drive.

Determine these requirements *before* you buy a package. You may find that you are not really interested in the software if you are not ready to invest in the hardware required to run it.

Be careful of listed "minimum requirements." The minimum may not be sufficient for your purposes or the package may be operating at such a handicap that it takes forever to do its job. For example, the Personal Record Keeping cartridge will work with a cassette data file, but it's slow if you have a lot of data to store.

Rule 10. Evaluate the *support* you will get.

As we have already said, software for home microcomputers is inexpensive. And software publishers cannot provide a lot of support for their users. Some software houses do offer *some* support.

With much of the software you buy, you won't need any support. If you do think you will need help, look at the support *before* you buy.

Many software publishers provide a "hot line" for you to call with questions concerning their products. Think up a question and try out the hot line service. If you cannot get an answer, or if the answer is wrong, or if you cannot get through at all, look elsewhere. Some software houses staff these positions with inexperienced people who know little about their products.

3.7 WHERE TO BUY SOFTWARE

You buy software in much the same way that you buy hardware. If you expect, or need, support from your dealer then you must buy your software from a reputable, knowledgeable, computer dealer. If you don't need help from a dealer and you want to save some money, order your software from a discount store or a mail order house.

You will pay more at a computer store, but *buying retail* affords you the opportunity to test the program before you take it home. Some retail dealers have sample TI 99/4A systems set up in their stores so that you can test software. If your retailer won't let you test it before you buy it, find another dealer.

If you cannot find a better retailer, you might as well order the software through the mail. A retailer who won't let you test a software package probably is not going to offer you much support anyway.

Discount store and *mail order* buying are definitely the cheapest ways to get software. You can expect 20% to 40% off the list price for software purchased this way.

Mass market discount stores often have very good prices on software. An advantage to buying in discount stores is taking the software home with you, instead of waiting for it to be delivered.

If you are looking at software at discount prices, remember to compare both prices and *shipping and handling charges*. There are wide differences between mail order houses in these "additional" charges that can make one a much better deal than the other.

There are a few things to watch out for when buying software:

- Make sure the software is the *most recent version* of the package. For example, LOGO II is out but there may be some old versions kicking around.
- Make sure you know the list price *before* you order. Some advertisements just plain lie. They scream "We have the lowest prices around" and then charge full list price for their software.

3.7.1 The Ethics of Discount Buying

A few final words about buying software. Be ethical and reasonable. Don't go into a computer store and spend a lot of the staff's time trying out the various products you want to buy and *then* order the software from a mail order house. This is not good business and will cost you in the long run.

Computer stores provide a valuable service along with their products. They provide support for those computer owners who need the help. You should not take advantage of this service without repaying them by buying their products.

If you are purchasing from a discount store or a mail order house, do whatever research you need to. Try the package out at a friend's house. Join a computer club and get whatever information you need. Read the reviews. Just don't waste the time of the computer store staff.

4

YOU, TOO, CAN BE A PROGRAMMER: BASIC ON YOUR TI

In this chapter, we talk about writing simple programs in BASIC and explain the differences between TI BASIC (the one that comes in the 99/4A) and Extended BASIC (that you buy in a cartridge).

We don't intend here to teach you everything you need to know to write BASIC programs. What we will do is tell you what you can expect to be able to do and what you will need to do it.

4.1 BASIC OVERVIEW

BASIC, *B*eginners *A*ll Purpose *S*ymbolic *I*nstruction Code, is the programming language most often used on home computers today. BASIC is relatively easy to learn, yet powerful enough that you can write substantial programs in it.

Like any programming language, BASIC is a set of instructions that tell your computer what to do. Texas Instruments provides a standard TI BASIC in ROM as part of the TI 99/4A Home Computer and, for more advanced programming tasks, an Extended BASIC cartridge.

If you really intend to do a lot of BASIC programming, you should get Extended BASIC. The additional features in Extended BASIC make it much easier to design and write BASIC programs, especially ambitious BASIC programs.

4.2 STANDARD TI BASIC

The standard TI BASIC that comes in ROM in your TI 99/4A console conforms to the *American National Standard for Minimal BASIC*. TI BASIC offers features beyond the minimal standard, like:

- *Color graphics:* you control up to 16 colors and can create user-definable characters.
- *Sound:* you control the duration (0.001 to 4.25 seconds), volume, and frequency of three independent tones, plus eight "periodic" or "white" noises. The frequency varies from 110 to 44,733 hertz (Hz) for tones.
- *Joystick control:* you determine the *position* of the joystick levers and the condition of the joystick *fire buttons* (pressed or not pressed).
- *Special keyboard scanning routines:* let you trap control character codes and/or split the keyboard into two (right and left) sections for multiple control (as in a two player game).
- *Special screen control and graphics routines:* let you easily define new characters, set character colors, and write at specific screen positions.
- *Arrays:* can have up to *three* dimensions.
- *Line Editor:* is built in and easy to use and includes *automatic line numbering* and *resequencing.*

These are only some of the excellent features available in standard TI BASIC. TI BASIC represents a good BASIC language set. It is more than sufficient for solving beginning programming problems.

However, TI BASIC cannot use more than the 16K of memory that comes in the TI 99/4A console. If you want, or need, to write programs larger than 16K, you must use Extended BASIC.

4.3 EXTENDED BASIC

While the TI BASIC that comes with the TI 99/4A is a very good version of BASIC, you may find that you need an even more powerful version. Texas Instruments developed their Extended BASIC language to take full advantage of many of the sophisticated features in the TI 99/4A Home Computer.

One important feature of Extended BASIC is that you can use the 32K Memory Expansion card. Programs that are stored and run in this memory can be larger, and often execute faster, than those run from the standard 16K RAM in the TI 99/4A console.

Extended BASIC can access up to 48K—the 16K that comes in the TI 99/4A plus the 32K from the Memory Expansion card. After everything is taken out, you end up with about 36,000 bytes of program and data space under Extended BASIC. This does not count the approximately 8000 bytes available for Assembler programs that you can link to from Extended BASIC.

Extended BASIC comes in a cartridge and offers these enhancements to the standard TI BASIC:

- *Sprites:* you define up to 28 independent graphics figures that can be moved around on the screen.
- *Speech:* your BASIC programs can speak through the Speech Synthesizer Peripheral.
- *Extended memory support:* lets you use more than the 16K RAM that comes in your computer. You can access up to 48K with the 32K Memory Expansion card.
- *Multiple statements on one line:* make it easier to enter programs, saves space since only the first statement on a line needs a line number, and results in faster execution of the program.
- *More functions:* like MAX, MIN, and PI.
- *Arrays:* of up to *seven* dimensions (increased from three).
- *Error handling:* lets you control error or warning conditions and take appropriate action within your program.
- *IF-THEN-ELSE enhancements:* allows you to enter multiple statements after the THEN and ELSE keywords, instead of only line numbers.
- *Assembly Language subroutine support:* you can load and link to TMS9900 Assembly Language routines. You will need the Editor/Assembler to enter Assembly Language statements.
- *Named subroutines with local variables and passed parameters:* make it easier for you to write programs using the most modern structured programming methods.
- *Enhanced input/output statements:* including formatted printing and cursor positioning control.
- *Merging of programs:* allows you to store commonly used routines on disk and merge them automatically into new programs as you write them.

Most programs written in TI BASIC will run in Extended BASIC. You should be aware of these differences between TI BASIC and Extended BASIC:

- If you have only the console memory to use, the maximum size of an Extended BASIC program is 864 bytes smaller than the maximum size of a TI BASIC program.
- Extended BASIC has some additional keywords that may conflict with variable names already in your program.
- The multiple statement option in Extended BASIC will conflict with TI BASIC PRINT statements that use :: (two colons) to indicate skip two lines. (In this case put a space between the colons, like this : :).

4.4 BASIC OPERATING MODES

As you use BASIC, you do different things at different times. Sometimes you get your program from a cassette tape. Sometimes you make changes to a particular line in a program. At other times you RUN a program.

TI BASIC lets you do these things by operating in three *modes:*

- In *Command mode,* you type commands, without line numbers, and the command is executed as soon as you press the ENTER key.
- In *Program mode,* you type statements that include line numbers, making them part of a program. The statements are not executed until you RUN the program.
- In *Edit mode* (see Section 4.7), you change lines in your program.

4.5 BASIC ELEMENTS

Some BASIC elements work only in Command mode, others only in Program mode. But they all have a customary use that classifies them as one of the following:

- Commands
- Statements
- Functions

Appendix E lists the TI BASIC and Extended BASIC commands, statements, and functions with a brief description of each.

4.5.1 Commands

Commands do something to your *program* or *disk files.* Commands do *not* operate directly on your data.

When used in Command mode, commands are executed *immediately* after you press the ENTER key. Some commands can also be used as statements, that is, as part of a BASIC program.

RESEQUENCE is an example of a command that *can't* be used as a statement. You use RESEQUENCE like this:

RESEQUENCE
or
RESEQUENCE 1000,50

DELETE is an example of a command that *can* be used as a statement. You use DELETE as a *command* like this:

DELETE "DSK1.AFILE"

You use DELETE as a *statement* in a program, like this:

```
500 DELETE "DSK1.AFILE"
or
900 DELETE "DSK2." & FILENAME$
```

4.5.2 Statements

Statements are part of a *program*. They are executed when the program is RUN. Some statements can also be used as commands, that is, entered and run directly from the keyboard—not as part of a BASIC program.

In TI BASIC, each statement included in a program must be entered on its own line, with its own line number. Extended BASIC lets you put more than one statement on a line.

Most BASIC statements can also be used as commands. For example, you use PRINT statements as *commands* in Immediate mode like this:

```
PRINT A+B
or
PRINT (750.59-255.36+45.93)*.34
```

You use PRINT statements in *programs* like this:

```
100 PRINT "HI THERE"
or
950 PRINT "THE TOTAL IS "; ANSWER
```

Some BASIC statements, like GOSUB and GOTO, cannot be used as commands because they are meaningless outside the context of a program. (Without a line number, where would you GOTO?)

4.5.3 Functions

Functions perform an operation and *return an answer* as though they were a *variable* in your program. Functions cannot stand alone. You must use them as part of a statement or command in the same way as you would use any other variable.

Functions can appear in commands and statements. For example, the square root function (SQR) can be used like this:

```
100 A = SQR (B^2 + C^2)
```

or in Immediate mode

```
PRINT SQR(256 + 398)
```

4.6 ENTERING BASIC PROGRAMS

You *enter* a BASIC program when you *type* it into your computer's memory. You can enter a BASIC program in either of these ways:

- Type a line number and one space followed by the BASIC statement, like this:

```
100 REM THIS IS A REMARK STATEMENT
```

or

```
1230 A = 4.5678
```

- Type the NUMBER (or NUM) automatic line numbering command and, optionally, the starting line number and increment values, like this:

```
NUMBER
```

or

```
NUM
```

(The first line number is 100 and the line numbers are incremented by 10.)

Or, like this:

```
NUMBER 250,25
```

or

```
NUM 250,25
```

(The first line number is 250 and the line numbers are incremented by 25.)

Continue to enter BASIC statements until your program is complete or until you want to make changes to what you have already entered. You use the Editor (described below) to make changes to the program in memory. Program Listing 4-1 shows how to enter a program.

Listing 4-1. Entering a BASIC program.

```
TI BASIC READY
>NUM 500,25<ENTER>
500 FOR I = 1 TO 16<ENTER>
525 CALL SCREEN (I) <ENTER>
550 NEXT I <ENTER>
575 <ENTER>
>RUN<ENTER>
```

Note: Things written like **THIS** are what the TI 99/4A says. Things written like *THIS* are what you type in. <ENTER> means you press the ENTER key.

Whatever you do, *remember to save the program* on tape or disk if you want to use it again without re-typing the entire program.

> CAUTION: BE CAREFUL WHEN YOU PRESS THE = KEY! The = key can do two things, depending on whether you are holding down the SHIFT key or the FCTN key when you press the = key.
> IF YOU PRESS FCTN = , you will return to the first (main title) screen and *lose the program in memory.*
> If you want to insert an = (like in A = B) MAKE SURE THAT YOU ARE HOLDING DOWN THE SHIFT KEY WHEN YOU PRESS THE = KEY.

You *will* make typing errors as you enter your programs. To correct those errors, you use the *line editor* commands shown in Table 4-1. Notice that you hold down the FCTN key (just like a SHIFT key) while pressing another key to perform these editing functions.

4.7 EDITING A BASIC PROGRAM

Your TI 99/4A has a built-in editor so that you can make changes to your BASIC programs. Once a program, or part of a program, is in memory, you use the editor by entering one of the following:

- EDIT *line-number* <ENTER>
- *line-number* FCTN E (up-arrow)
- *line number* FCTN X (down-arrow)

After you enter one of these commands, you will see the line that you asked for (*line-number*) displayed on your screen. You can make any changes that you want to the line or delete the line.

Table 4-1 Line Editing While Entering BASIC Programs

Key	Function
ENTER	Enter the program line. The line you are typing (line number and statement) is entered into the program currently in your computer's memory.
FCTN D (right-arrow)	Forwardspace one character. Move the cursor one character position to the right. No changes are made to any characters the cursor moves past. You use the FCTN D key to position your cursor when you want to add or delete characters on the line you are currently typing.
FCTN E (up-arrow)	Works just like the ENTER key. The program line you just typed is put into your computer's memory.
FCTN S (left-arrow)	Backspace one character. Move the cursor one character position to the left. No changes are made to any characters the cursor moves past. You use the FCTN S key to position your cursor when you want to add or delete characters on the line you are currently typing.
FCTN X (down-arrow)	Works just like the ENTER key. The program line you just typed is put into your computer's memory.
FCTN 1 (DEL)	Delete one character. Delete the character under the cursor. You usually use the FCTN S or FCTN D key to position the cursor to the character you want to delete.
FCTN 2 (INS)	Insert characters. Insert characters at the cursor position. You can use the FCTN S or FCTN D key to position the cursor to the position where you want to insert the characters. Unlike the other FCTN keys, INS puts you into *Insert Mode,* allowing you to insert as many characters as you need.
FCTN 3 (ERASE)	Erase the entire line. Does not erase the line number if you are in automatic line numbering mode (NUMBER command).
FCTN 4 (CLEAR)	Clear the current line. Cancels the line you are typing. If you are in automatic line numbering mode, FCTN 4 erases the current line and takes you back to command mode.
FCTN = (QUIT)	Quit. Leave BASIC and return to the main title screen. Memory is erased. If you have files opened, they are *not* closed. Use a BYE command if you want your files closed. Remember, you lose the program in memory if you have not saved it.

Several keys have a special meaning when you use them as function keys (hold down the FCTN key and the other keyboard key at the same time) in EDIT mode. Table 4-2 shows the keys you use to edit TI BASIC and Extended BASIC programs. Table 4-3 shows the extra editing function available in TI Extended BASIC.

Table 4-2 TI BASIC and Extended BASIC EDIT Mode Function Keys

Key	Function
ENTER	Enter the program line. The line you are editing (line number and statement) is entered into the program currently in your computer's memory.
FCTN D (right-arrow)	Forwardspace one character. Move the cursor one character position to the right. No changes are made to any characters the cursor moves past. You use the FCTN D key to position your cursor when you want to add or delete characters on the line you are currently editing.
FCTN E (up-arrow)	Enter the current line and edit the next lower numbered line. If there are no lines in the program that have lower line numbers, leave Edit Mode. This is very useful when you are "stepping" through a program making changes.
FCTN S (left-arrow)	Backspace one character. Move the cursor one character position to the left. No changes are made to any characters the cursor moves past. You use the FCTN S key to position your cursor when you want to add or delete characters on the line you are currently editing.
FCTN X (down-arrow)	Enter the current line and edit the next higher numbered line. If there are no lines in the program that have higher line numbers, leave Edit Mode. This is very useful for "stepping" through a program when you are making changes.
FCTN 1 (DEL)	Delete one character. Delete the character under the cursor. You usually use the FCTN S or FCTN D key to position the cursor to the character you want to delete.
FCTN 2 (INS)	Insert characters. Insert characters at the cursor position. You can use the FCTN S or FCTN D key to position the cursor to the position where you want to insert the characters. Unlike the other FCTN keys, INS puts you into *Insert Mode,* allowing you to insert as many characters as you need.
FCTN 3 (ERASE)	Erase the entire line. Does not erase the line number.
FCTN 4 (CLEAR)	Clear the current line. Erases the current line and takes you back to command mode.
FCTN = (QUIT)	Quit. Leave BASIC and return to the main title screen. Memory is erased. If you have files opened, they are *not* closed. Use a BYE command if you want your files closed. Remember, you lose the program in memory if you have not saved it.

Table 4-3 Additional Extended BASIC EDIT Mode Function Keys

Key	Function
FCTN 8 (REDO)	Print the contents of the most recently entered line and prepare to edit the line. The *last* line that you entered is redisplayed. You can change *any* of the data on the line, *including the line number.* This makes it easy to enter several lines that are similar or to move statements.

4.7.1 Renumbering the Lines in Your BASIC Program

After you make changes to your program, you will notice that your formerly orderly line numbers are now rather messy. It's easier to make changes to a program with nicely sequenced line numbers.

You can easily *renumber* your program's statements with a RESEQUENCE (RES) command. Renumbering, or resequencing, your BASIC program adjusts all of the line numbers so that the line numbers begin at the initial value you want and increase by the increment value you specify.

You resequence the BASIC program currently in your computer's memory by entering:

```
RES
```

which starts the line numbers at 100 and increments them by 10.

If, instead, you want to start your line numbers at 500 and increment them by 50, you enter:

```
RES 500,50
```

All the line numbers in your program are adjusted to the new values. Any line numbers in GOTO or GOSUB statements are also adjusted to reflect the new (changed) line numbers.

5

EXPANDING YOUR SYSTEM

There are many peripherals available for the TI 99/4A. You don't need every peripheral for your computer, but it's often difficult to know just what you should buy.

In this chapter, we tell you what peripherals are available for your TI 99/4A, explain what they do, and give you some suggestions on how you can expand your system.

5.1 BUILDING A SYSTEM

When you consider expanding your system, think of your TI 99/4A in the same way that you think of a component stereo system. First, you buy the basic parts, a receiver as the start of a stereo system, the TI 99/4A console as the start of your computer system. You then add peripherals and software to your TI 99/4A, just as you would add a turntable, speakers, and records to your stereo system.

The choices you make in expanding your computer system are similar to the choices you make in expanding a stereo system. Some people want more speakers; others would rather have an open-reel tape deck. Some of you will want more memory; others, a printer.

Most of us cannot afford to buy every piece we want all at one time. We have to plan our purchases, buying first those pieces that meet our immediate needs, or our limited budget.

5.2 WHAT IS A PERIPHERAL?

A *peripheral* is something that you attach to your TI 99/4A console to make it *do more* for you. Peripherals increase the capabilities of your computer system and include things like:

- Cassette recorders
- Joysticks
- Disk drives
- The Peripheral Expansion Box
- Printers
- Modem

You have probably already bought some peripherals for your TI 99/4A. Maybe you bought the joysticks, or the cassette cables and a recorder. These are useful, inexpensive peripherals that nearly everyone buys.

Each peripheral is designed to do something extra for you, like let you rapidly store and retrieve your programs and data, produce printed output, or communicate over the telephone lines.

5.2.1 Classifying Peripherals

The TI 99/4A supports the following kinds of peripherals:

- Those that *connect directly* to the TI 99/4A console
- Those that *fit into* the *Peripheral Expansion Box*
- Those that *connect to* the *Peripheral Expansion Box*
- Those that *connect to* the *Hexbus Interface*

TI developed the *Peripheral Expansion System* (Expansion Box) to make it easier for you to add new equipment and to lower the cost of the peripherals. Some of the older non-Expansion Box peripherals are still available. These plug into the side of the TI 99/4A console. However, they are more expensive than the corresponding Expansion Box peripherals and may be discontinued in the future.

The newly introduced *Hexbus Interface* lets you add low cost peripherals like a printer/plotter and Wafertape to your TI 99/4A. These peripherals are lower in cost than the Expansion Box peripherals and have less capability. They are a good way to expand your system without spending a lot of money.

5.2.2 Some Precautions

We talk about the peripherals in a general sense in this book. We don't intend to replace the documentation that comes with each peripheral, especially the installation instructions.

Every peripheral comes with an *instruction book. Read these books before you attempt to use the peripheral.* And *follow the instructions for installation.*

5.3 NONEXPANSION-SYSTEM PERIPHERALS

Some TI 99/4A peripherals connect directly to the TI 99/4A console itself. These nonexpansion-system peripherals include:

- RF Modulator (TV Adapter)
- Color Monitor and Cable
- Solid State Speech Synthesizer
- Wired Remote Controllers (Joysticks)
- Cassette Cables

5.3.1 The RF Modulator (TV Adapter) and Your Television

If you have your TI 99/4A hooked up to your television, you know about the rf modulator. It's the attachment that lets your TI 99/4A write characters and draw pictures on your television screen.

The *rf modulator* turns the RGB (Red-Green-Blue) signal coming from the TI 99/4A console into a standard broadcast tv signal. If you are using a regular television, you must use an rf modulator. You can skip the rf modulator if you have a monitor that accepts RGB signal input.

If your television is a *cable-ready* version, you will have to get an *adapter* from your local electronics store. The adapter is inexpensive ($3.00 to $5.00) and fits between your television and the rf modulator connections.

You don't need a color television if you don't want to see the colors available through your TI 99/4A. The rf modulator works perfectly well with a *black-and-white* set.

5.3.2 The Color Monitor and Cable

TI, along with several other manufacturers, offers high quality *color monitors* that draw clearer, brighter images than you can get on an

ordinary color television. Color monitors directly accept the RGB signals generated by the TI 99/4A console.

If you have a monitor, you will not need an rf modulator. You will need the special *monitor cables* to connect your TI 99/4A to the color monitor.

5.3.3 Solid State Speech Synthesizer

Your TI 99/4A has a great advantage over many of the other home computers available today. It can talk to you through its *Speech Synthesizer.* Fig. 5-1 shows you what it looks like.

The speech synthesizer produces sounds like a voice. This means that you can write or buy programs that talk to you. Very young children can use the TI 99/4A with educational programs that talk to them.

Many software packages are designed to use the speech synthesizer, including educational software, the Terminal Emulator, and some games.

The console TI BASIC cannot use the speech synthesizer. If you want to write programs that talk, you will need one of these:

- Extended BASIC
- Speech Editor
- Editor/Assembler

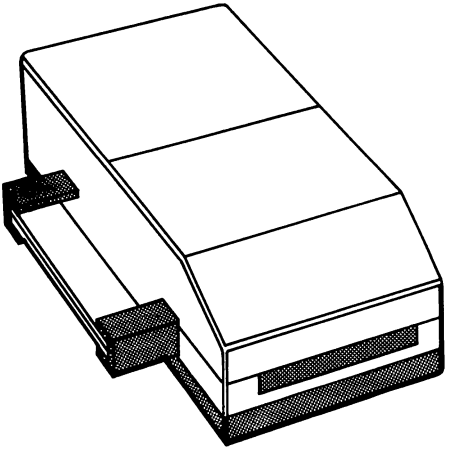


Fig. 5-1. Solid state speech synthesizer.

5.3.4 Joysticks or Wired Remote Controllers

TI's *Wired Remote Controllers* (called *joysticks* by arcade gamers) connect directly to the *left side* of your TI 99/4A console (the side nearer the Q key).

Many games use the joysticks to control the action on the screen. TI's joysticks come in a set of two which connect through a single connector to the TI 99/4A console. These are very sturdy joysticks that children love. However, some adults find them too small for their hands.

Other manufacturers make *adapters* that let you connect almost any joystick to your TI 99/4A. Fig. 5-2 shows you what some of the adapters and joysticks look like. Simply plug the adapter into the joystick connector on the left side of the console and plug the non-TI joysticks into the adapter.

You can choose whatever joysticks you want to connect to your TI 99/4A if you get an adapter. You can even use two different models for those of you who have strong preferences.

Remember, no matter which joysticks you decide to use, you must

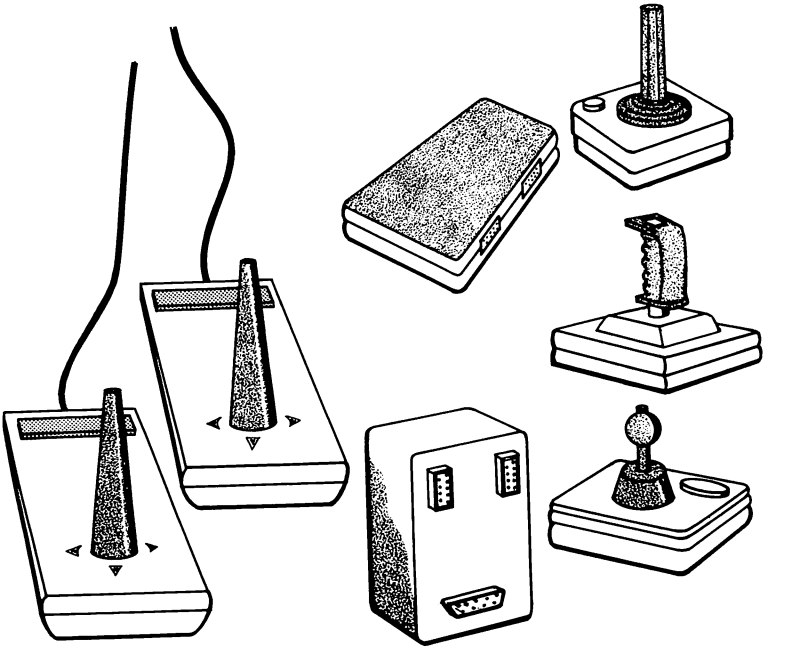


Fig. 5-2. Joysticks and adapters.

have the *Alpha Lock key* in its *unlocked* position when you are using the joysticks. If you have it Alpha Locked, your joysticks cannot move an object "up" on the screen (toward the top of the screen).

5.3.5 Cassette Cables and Cassette Recorders

You will need some way to store your own programs and data, and some way to get purchased programs into your TI 99/4A. The cheapest way to do that is with the cassette cables.

The cassette cables let you use just about any cassette recorder with your TI 99/4A. You don't need a special recorder, though some recorders work better than others.

Fig. 5-3 shows you what the cassette cables look like. Notice that there are two sets of leads. These two leads let you connect *two recorders* to your TI 99/4A at the same time. TI calls the *cassette recorders CS1* and *CS2*.

Recorder *CS1* connects to the lead with *three plugs* (red, black, and white). If your recorder does not have a remote jack outlet, use the red and white plugs and ignore the black plug. You can *read from* and *write to* recorder CS1.

Recorder *CS2* connects to the lead with *two plugs* (red and black). You can *only write to* recorder CS2.

You don't need to buy a special recorder for your TI 99/4A. If you are buying a new recorder, try to get one with a treble control because your TI 99/4A likes high treble better than bass.

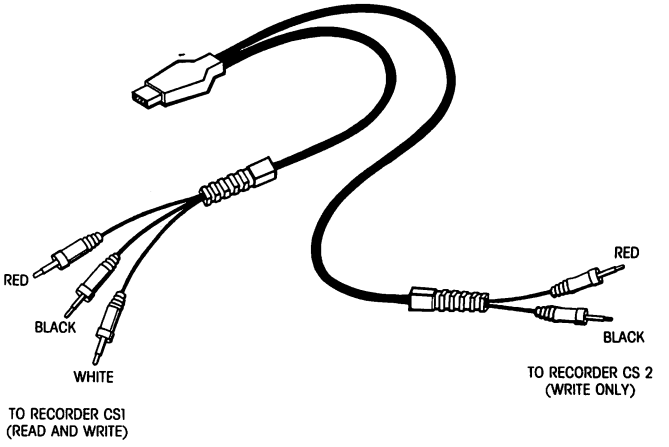


Fig. 5-3. Cassette cables.

5.4 EXPANSION SYSTEM PERIPHERALS

Most peripherals are part of the TI 99/4A Peripheral Expansion System. The Peripheral Expansion System starts with the *Peripheral Expansion Box* and the Peripheral Expansion Interface card, the link between the TI 99/4A and the Expansion System.

The Expansion Box can hold up to *eight "cards"* (one of them must be the Expansion Interface card) and *one expansion system disk drive.* Most of the cards connect external peripherals, like a printer, modem, and disk drives, to your computer.

There are some very good reasons for getting the Expansion Box:

- It takes much less space and is far more manageable than stringing out individual peripherals.
- It costs less than buying many individual peripherals.
- The design makes it easy for you to include non-TI peripherals in your system.

We will describe the most common components of the Peripheral Expansion System:

- Peripheral Expansion Box
- Memory Expansion Card
- Disk Controller Card and the Disk Manager Cartridge
- Expansion System Disk Drives
- External Disk Drives
- RS-232 Interface Card
- Printers
- Modems

5.4.1 Peripheral Expansion Box

We cannot talk about the Peripheral Expansion System without first talking about the Peripheral *Expansion Box* and the Peripheral *Expansion Interface card.* Fig. 5-4 shows you what the Expansion Box looks like.

Why do you want one of these boxes, you ask? What good is it? It holds all the other peripheral expansion interface cards and it connects some peripherals, like the printer and modem, to the console.

The Expansion Box comes with its own Peripheral Expansion Interface card. The Expansion Interface card is what really communicates with the computer. Other cards available for the Expansion Box are:

- Memory Expansion
- Disk Controller
- RS-232 Interface
- P-Code

5.4.2 Memory Expansion Card (32K)

Your TI 99/4A comes with 16K *of* RAM, more than enough for you to learn to program in BASIC and to use much of the cartridge, disk, and cassette software available.

But sometimes you cannot get by with only 16K. If you decide to use TI LOGO, PILOT, PLATO, or UCSD Pascal, you will need more memory. Some other software packages, like Multiplan or TI Writer, also require more memory.

If you use the Editor/Assembler, the Mini-Memory Module, or Extended BASIC, you can write programs that access more than 16K. The standard BASIC that comes with your TI 99/4A can only access the memory (16K) included in the console.

You can add *32K* more memory to your basic system by using the Memory Expansion card, shown in Fig. 5-5. This gives you a total of 48K of RAM.

There are *several manufacturers,* besides TI, making memory expansion cards to fit the Expansion Box. These may not be as sturdy as the ones from TI, but they are a good deal cheaper.

5.4.3 Disk System

Disk drives are wonderful devices that hold 90,000 characters of programs and data on a *single sided* disk. Other manufacturers offer *double sided drives.* These drives record data on both sides of the

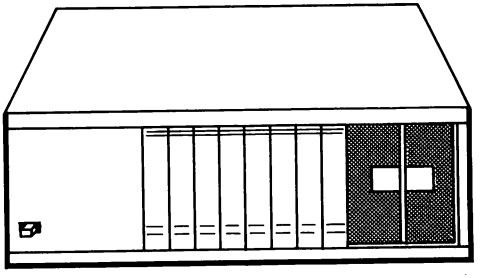


Fig. 5-4. Peripheral expansion box.

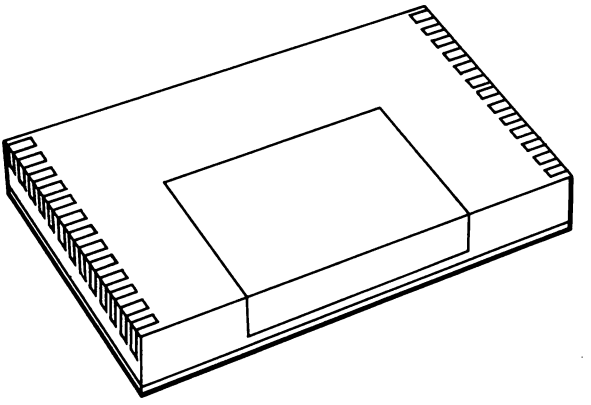


Fig. 5-5. Memory expansion card.

disk and store 180,000 characters per disk. You can read from and write to a disk drive about 30 times faster than to a cassette.

Unfortunately, disk drives are relatively expensive. If you are storing small amounts of data, you can just as well use a cassette. You do face one cassette limit when you start writing large programs—the largest program you can write to a cassette is 12K. There are no such program size limits with a disk.

Some programs are only available on disk. And some applications require a disk. The Editor/Assembler package, for example, is only available on disk.

Disk drives are not as easy to connect to your system as the cassette. You need the *Disk Controller Interface* card for the Expansion Box. The Disk Controller Interface card "talks" to your computer and your disk drive. You can attach up to *three disk drives* to one disk controller card.

You get the *Disk Manager* cartridge with the Disk Controller card. This handy cartridge makes it easy to:

- Format disks
- List the names of the files on a disk
- Rename files
- Delete files
- Copy files
- Make backup copies of a disk

First you get the *Expansion System Disk Drive* that fits into the right side of the Expansion Box.

We must warn you about installing this drive. Someone with adult

sized hands will say very nasty things about disk drives before they finish installing this one. (You could ask a 6 year old to make the connections—that's about the size hands you need.) If you have no talent for things mechanical, get your drive from a store that will install the drive in your Expansion Box.

Once you have your disk system working, you may want to get another drive. Since your Disk Controller will handle up to three drives, each additional drive after the first is less expensive than the first (you don't need to buy another controller card).

You can fit only one disk drive into the Expansion Box. You need external disk drives (like the one shown in Fig. 5-6) for your second and third drives.

You connect the external disk drives to the Disk Controller card tab that sticks out the back of the Expansion Box. Make sure that you get the cables you need to do this.

At this time, several companies are offering TI-compatible disk drives at very attractive prices. If you decide to get one of these drives, make sure that the salesperson guarantees that it will work with your TI 99/4A.

5.4.4 RS-232 Interface Card

The RS-232 Interface card fits into the Expansion Box and has a special tab that fits out the back of the box. This tab has a *16-pin parallel* port and a *25-pin serial* port. Ports are the communications channels between the RS-232 card and an external device, like a

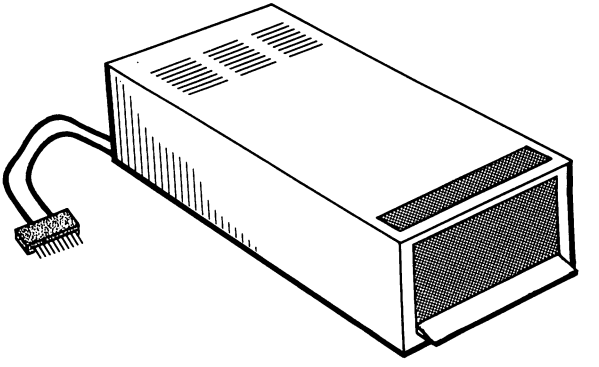


Fig. 5-6. External disk drive.

printer. Some printers use a parallel communications interface and others use a serial interface. Modems use the serial interface.

Because it's an industry-standard interface, the RS-232 Interface card gives you a wide *choice of industry-standard peripherals* (like printers, plotters, and modems). You are not limited to peripherals offered by TI. *You* pick what you like best, and can afford, from the multitude of products available in the larger computer peripherals marketplace.

You can get a "Y" cable for the serial port on your RS-232 Interface card. The Y cable lets you connect *two serial devices* to the single serial port. You can, therefore, connect up to three devices (one parallel, two serial) to a single RS-232 Interface card.

You can put *two* RS-232 Interface cards into the Expansion Box. If you do want to use two RS-232 cards, you must have the second card permanently modified at a Texas Instruments Service Facility or order a modified card.

5.4.5 Printers

Your TI 99/4A *talks to printers* through the RS-232 Interface card. You can use either a "parallel printer" (one that attaches to the RS-232 8-bit, 16-pin parallel port) or a "serial printer" (one that attaches to the RS-232 8-bit, 25-pin serial port). These are standard microcomputer printer connections.

You have an amazing variety of industry-standard printers to choose from. Fig. 5-7 shows you what two different printers look like. Printers are divided into these general categories:

- *Dot-matrix printers* form the characters out of a *bundle of little wires* that make "dots."
- *Fully formed character printers* use a *typing element* on a "daisy wheel" with the letters on the ends of the "petals"; some printers of this type use a "thimble" or a "ball" instead of a "daisy wheel."

The TI printer is a medium priced, dot matrix printer that prints upper and lower case characters and graphics characters. There are a number of similar printers on the market. Look at and choose the one you like best (and can afford).

You will most likely buy a dot matrix printer. Fully formed character printers are relatively slow and very expensive.

Dot matrix printers vary widely in their printing capabilities and the quality of the characters they print. Printers come with an enormous

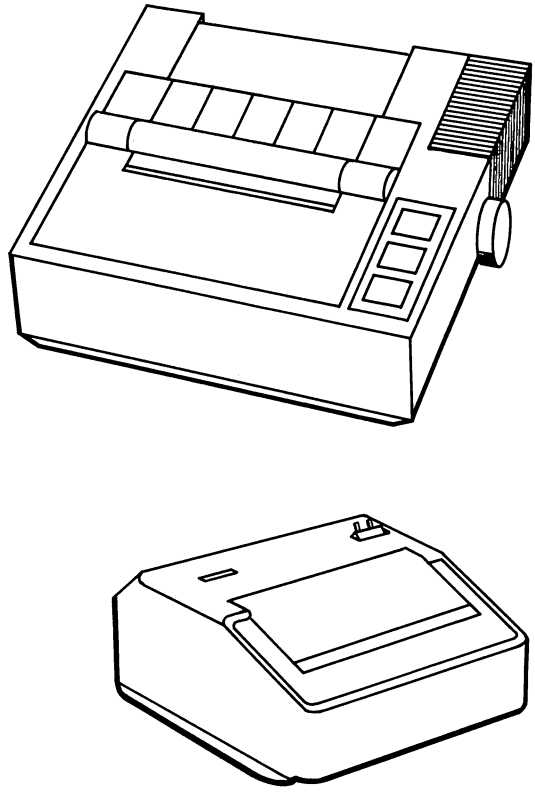


Fig. 5-7. Printers.

mix of features. This mix determines the price of the printer. Some features to look for in a dot matrix printer are:

- The *number of dots* that form a character determines the readability of the printing and the characters that can be printed. Too few dots means only upper case characters, or lower case characters with no *descenders* (the part of letters like p, q, y, and g that extends below the line).
- The *speed* of the printer affects both the quality of the printing and the price of the printer. Many printers operate at more than one speed, with a reduction in print quality as speed increases.
- Some printers use *friction feed,* which is unreliable for printing long listings. Others use *tractor feed* or *pin feed* mechanisms that are more reliable.

- Some printers accept only 8½ inch wide paper. Others use up to 14 inch wide paper. Watch out for printers that require special paper. Special paper can be hard to get and very expensive.
- Graphics, special character sets, variable character size, color printing, and proportional spacing are among the many accessory features available.

You can find dot-matrix printers for as little as $200 or as much as $3000. The price depends on what mix of features and speed you choose.

The choice of printer is very much a choice of what *you want to look at* on the printed output. If you cannot easily read the letters that your printer prints, the printer is useless.

Before you buy a printer, ask to see something that has been printed on the one you like. If you cannot read it or if you don't like the way it looks, check other printers in the same price range.

5.4.6 Modems

A *modem* (*Mo*dulator-*Dem*odulator) is the communication interface between your computer and another computer. Modems take information from your computer, translate it into data that can be sent over telephone lines, and send it. On the other end, a modem does the reverse, taking the information from the telephone line and translating it so that the computer can understand it.

Modems look like small boxes with or without places for you to put your phone. Fig. 5-8 shows you two commonly available styles.

You use a modem when you want your TI 99/4A to act like a terminal to another computer. You need a modem to use services like TEXNET or The Source and you need the Terminal Emulator cartridge to make your TI 99/4A behave like a terminal.

The modem connects to the RS-232 Interface card through a serial port. You cannot use a modem without the RS-232 Interface card.

TI's modem has a place to put the phone handset (the part you hold when you talk on the phone). Other modems connect directly to the *universal connectors* (those little square connectors) on the phone.

Several manufacturers make modems specifically for the TI 99/4A. You can use any standard modem that communicates at the correct speed (up to 300 baud at this time).

You may find a real bargain in modems as more and more people hook their home computers into services like The Source. Manufac-

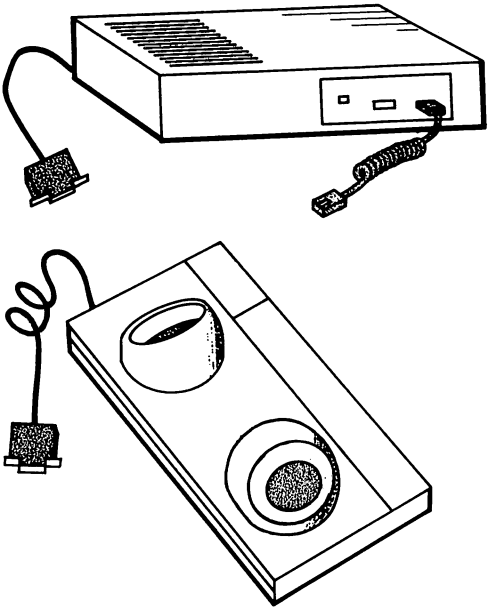


Fig. 5-8. Modems.

turers sometimes offer free connection to such a service when you buy their modem.

5.5 THE HEXBUS PERIPHERALS

TI's Hexbus interface lets you connect relatively low cost Hexbus peripherals to your TI 99/4A. The Hexbus peripherals are small (about 4.5 inches (114 mm) wide, 5.75 inches (146 mm) deep, and 1.5 inches (38 mm) high) and stackable.

Right now, you can get these Hexbus peripherals:

- Hexbus Interface
- Wafertape Drive
- Four Color Printer/Plotter
- RS-232 Interface

TI plans to introduce serveral more Hexbus peripherals including other printers and a modem.

5.5.1 The Hexbus Interface

The *Hexbus Interface* shown in Fig. 5-9 attaches to your TI 99/4A on the right side (where the Speech Synthesizer or Peripheral Expansion Interface attaches).

The Hexbus Interface lets you connect any Hexbus Peripheral to your TI 99/4A.

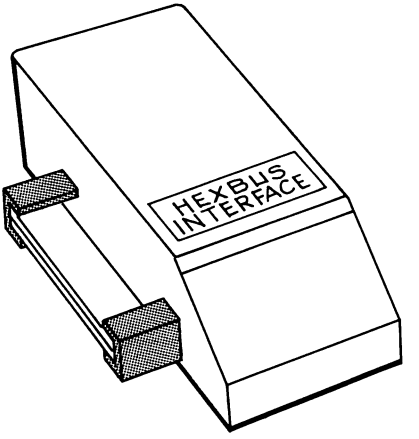


Fig. 5-9. Hexbus interface attachment.

5.5.2 The Wafertape

The *Hexbus Wafertape* digital tape drive and Wafertape tape shown in Fig. 5-10 give you a reasonably priced, sequential random-access storage device.

The Wafertape is a digital recording medium (unlike a cassette tape which is not digital). Reading from and writing to a Wafertape is much faster and more reliable than using an audio cassette. Another important feature of the Wafertape is that you can store and retrieve files by name, just as you can on a disk. You cannot store or retrieve audio cassette files by name.

Wafertapes come in various lengths from 5 to 50 feet (1.5 to 15 m). You can store up to 48K bytes of programs and data files on a single 50-foot (15 m) Wafertape.

5.5.3 The Four-Color Printer/Plotter

The *Hexbus four-color printer/plotter* shown in Fig. 5-11 is a plain-paper printer/plotter. This peripheral gives you an inexpensive way to have hardcopy listings of your programs and data.

You can write anything that you can draw on your screen onto the roll of paper. The printer/plotter can use up to four different pens (usually black, red, blue, and green) when drawing or printing its characters.

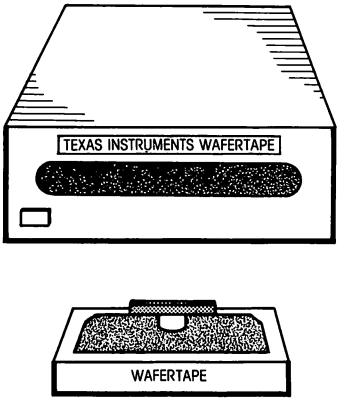


Fig. 5-10. Hexbus wafertape drive and medium.

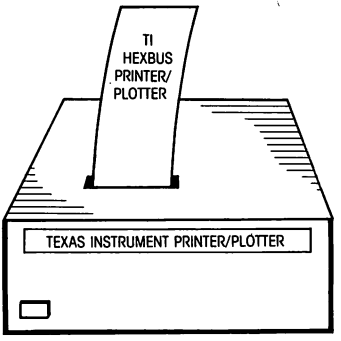


Fig. 5-11. Hexbus four-color printer/plotter.

5.5.4 The RS-232 Interface

The *Hexbus RS-232 Interface* lets you connect peripherals like printers and modems to the Hexbus. The Hexbus RS-232 Interface looks like the other Hexbus peripherals—a small metallic box.

You can also get the Hexbus RS-232 interface with a parallel output port so you can connect parallel printers to the Hexbus. The Hexbus

RS-232 interface lets you connect a modem and/or printer to your TI 99/4A without having to invest in the Expansion Box.

5.6 PLANNING FOR EXPANSION

Everyone who gets a home computer wants to expand it sooner or later. But, not everyone will want, or need, to expand their systems in the same way.

We are going to discuss some common ways to expand your system, using a planned approach to purchasing your hardware and software.

5.6.1 What It Costs

It's often difficult to decide what you want next. There are any number of wonderful peripherals that you *really need.*

To talk about buying equipment, we have to talk prices. Depending on various market factors, you will often find equipment at substantially lower prices than the suggested list prices from Texas Instruments. The equipment prices we quote in this book are *typical discount prices* for TI 99/4A peripherals.

As an example of what things will cost, Table 5-1 shows you some (1983) suggested list prices and some commonly available discount prices.

You can see that there is quite a difference between the suggested list prices and currently available discount prices. You will also find specials in your local stores or through the ads from reputable mail order establishments in the popular home computing magazines and newspapers.

Chart 5-1 lists the components of a well-equipped system. At list price, this equipment would cost about $2500—a real investment. However, at currently available discounted prices, the same equipment would cost about $1780, a savings of $720 (or 29%).

5.6.2 TI Versus Non-TI Peripherals

Your TI 99/4A uses a standard RS-232 interface to talk to its printer and modem. This means you are not locked into buying TI peripherals and you can often find a reasonably priced substitute.

The obvious choices for non-TI peripherals are the printer and modem. But you might also consider *non-TI cards for the Expansion Box.* The easiest to find are substitute Memory Expansion cards, for about half the list price of the TI card.

Table 5-1 Peripherals and Estimated Costs

Item	Suggested List Price	Average Discount Price
Cassette Cables	$ 14.95	$ 12.00
High Resolution 10-inch Monitor	$399.95	$320.00
Peripheral Expansion System	$249.95	$175.00
RS-232 Card	$174.95	$125.00
Disk Controller Card	$249.95	$175.00
TI Expansion System Disk Drive	$399.95	$275.00
TI External Disk Drive	$499.95	$375.00
Non-TI External Disk Drive	—	$250.00
TI Memory Expansion Card (32K RAM)	$299.95	$215.00
Non-TI Memory Expansion Card	—	$150.00
P-Code Card	$249.95	$180.00
Solid State Speech Synthesizer	$149.95	$100.00
TI Telephone Coupler Modem	$224.95	$163.00
Non-TI Modem	—	$100.00
TI Impact Printer	$750.00	$500.00
Non-TI Dot Matrix Printer	—	$200.00 through $1500.00
Non-TI letter quality printer	—	$400.00 through $3000.00
Terminal Emulator II Cartridge	$ 49.95	$ 35.00
Hexbus Interface	$ 59.95	$ 48.00
Hexbus Printer/Plotter	$200.00	$160.00
Hexbus Wafertape	$140.00	$119.00
Wafertapes (50 foot)	$ 8.00	—
Hexbus RS-232 Interface	$100.00	$ 81.00

Chart 5-1
Peripherals in a Well-Equipped System

Expansion Box
Disk Controller Card
Expansion System Disk Drive
Printer
Memory Expansion (32K) Card
RS-232 Interface Card
Telephone Coupler (Modem)
Terminal Emulator II Cartridge
Extended BASIC Cartridge
Cassette Cables
Cassette Recorder

Should you buy Brand X equipment? You are taking some chances if you do, but usually this equipment is reliable and guaranteed. It's almost impossible to go wrong with printers as long as you buy a well known brand.

With modems, you cannot go wrong if you make sure that the modem will work with the TI—TI uses an industry-standard modem. Be careful that you don't get a nonstandard modem specifically made for another home computer.

You will find information on industry-standard peripherals in any of the microcomputer magazines listed in Appendix A.

5.7 BUYING HARDWARE

You can buy hardware in many places—computer stores, discount stores, department stores, toy stores, mail order. And, while you are looking, you will find a wide range of prices for the same item.

But before you go buying anything, ask yourself some questions. Should you go for the cheapest price? Do you really want to order through the mail? Do you need a lot of help putting equipment together? What if there is a problem and the equipment does not work? Who fixes it?

5.7.1 Assess Your Mechanical Talents

The way you buy hardware depends a great deal on the level of experience you have with computers and the mechanical skills you have. Usually, computer users (and not only new home computer users) fit into one of these categories:

- *Mechanically adept, experienced* computer user who can easily put together components and has some experience with computers in general.
- *Mechanically inept, experienced* computer user who has some experience with computers but little, if any, mechanical skill. This user would find it painfully difficult and incredibly boring to do anything more than turn on equipment.
- *Mechanically adept, inexperienced* computer user who has little, if any, experience with computers but has a lot of mechanical ability. This user would typically find it easy and fun putting together components, having no difficulty with connecting peripherals to the TI 99/4A.
- *Mechanically inept, inexperienced* computer user who has little experience with computers and finds putting the components together a horrible experience. This user typically spends hours attaching the rf modulator to the television antenna leads.

Don't worry about which category you fit into. Just be honest. If you are inexperienced, admit it. You will save yourself a lot of trouble.

The amount of experience you have with computers in general determines how much help you need when deciding which parts to buy. Obviously, if you have more experience you are able to make these decisions more easily than someone who is just beginning to use computers and who does not really know what's available.

Your level of mechanical skill makes a great difference in whether you can buy equipment through the mail. If you cannot assemble it yourself or find someone to assemble it for you (a family member or a friend), you will have to buy at retail stores and get the salesperson's help.

5.7.2 Where to Buy Hardware

Discount stores have *great prices* and *little,* if any, *help* after (or even before) the sale. You go into the store, pick up the box with whatever you are buying, pay for it, and walk out. No one will put the pieces together for you. No one will be able to answer any questions about the peripherals.

Computer stores usually sell at *list price.* This is more than you would pay at a discount store but you get some service in a computer store. You will find salespeople who know what the peripherals do and what they are good for. You will get *answers* if you have problems. You might even get *help* putting the pieces together. You pay for this extra service, but it's worth it if you need the help.

If you need a lot of assistance, don't go for the cheapest prices. You are better off paying slightly more and getting the help you need.

Once you have gained some experience (and the ability to diagnose and solve any problems that occur), you can look for bargains through *mass market retailers* and *mail order* firms. When you buy this way, you get equipment in *sealed factory cartons* and *no help.* Reputable firms will exchange damaged items; find out whether the firm you are dealing with has a reasonable exchange policy *before* you buy.

Local and *national user's groups* often offer *special deals* on hardware and software.

5.7.3 Maintenance

There is not much *maintenance* for home computers. Service centers are expensive to maintain, so most service is on a mail-in basis.

Solid state components, like the TI 99/4A console and the Expansion cards, are *very reliable.* The general rule for solid state equipment is: If it does not fail in 90 days, it probably will not.

Mechanical peripherals are more likely to cause problems. Printers are the chief offenders here. Disk drives will sometimes cause problems, though far less often than printers.

If you buy TI peripherals, you get a reasonable warranty and TI will replace or repair the equipment if there is a problem. When you need repairs, TI has the facilities to do them.

What if you buy from someone else? Be sure that they will replace the equipment if it does not work or that they have some way to repair it. Sending your printer to some repair center thousands of miles away and then waiting for months to get it back is not considered good service. Find out what's in store for you *before* you put out your money.

5.8 TYPICAL SYSTEM EXPANSIONS

To help you decide the best way for *you* to expand your system, we have included a number of different *cases,* each designed to meet a specific goal.

We will also look at these expansion plans to give you some ideas how to expand your system:

1. Using your TI to play arcade and adventure-type games
2. Running educational software in cartridges and TI LOGO II
3. Running the PLATO system
4. Using your TI as a terminal to an office computer or an on-line service like TEXNET
5. Creating the ultimate system for a super programmer

Each case will show you:

- A brief description of the reasons for expanding the system
- A list of equipment needed and an approximate cost (at currently available discount prices)
- A suggested expansion plan, showing one way to reach the ultimate system in several small steps.

As we write, TI has announced, but not released, a set of peripherals that were designed for its Compact Computer. You can also use these *Hexbus* peripherals on the TI 99/4A if you buy the Hexbus interface (about $60.00 list).

The two most significant Hexbus peripherals that TI has shown so far are the Wafertape (about $140 list) and the four color printer/ plotter (about $200 list). They also have an RS-232 interface and have promised another printer and a modem for the Hexbus. These peripherals are significantly cheaper than those for the Peripheral Expansion System mainly because they avoid the initial cost of the Expansion Box.

These Hexbus peripherals can serve as low-cost or higher-performance alternatives to standard TI 99/4A printers, cassette tapes, disk drives, and modems.

Case 1: You want to use your TI 99/4A to play both arcade type and adventure type *games.*

You bought your TI 99/4A because you liked its action-based arcade games and the word-based adventure games.

There are some very good games for the TI 99/4A. All the action games from TI can use either keyboard input or joystick control, but most people prefer to use the familiar joysticks. So, one expansion item is a set of joysticks.

Don't think that it's easy to buy joysticks either! TI's set of two joysticks (which TI calls "Wired Remote Controllers") are sturdy, reliable, and rather small. Children find them wonderful. Adults, in the throes of fighting aliens or escaping through a maze, often grip the joysticks too tightly and suffer painful hand cramps.

What to do? Get an adapter and whatever joysticks your hands and wallet like best. Several companies make adapters that let you use any joystick you want. Then you can choose whatever kind of joystick you like best, maybe two different models for the different needs of family members.

Still in the world of game playing, you may find yourself buying cassette cables. Many adventure games have a cartridge that decodes your commands and a tape (or disk) that describes one of several games. The least cost alternative is to use the cassette based software and get the cassette cables. You may have to purchase a cassette recorder if you don't already have one. You can choose from a wide variety of readily available recorders.

A higher performance alternative to the cassette tape is the Hexbus Wafertape. It's much faster than a cassette and you can access your adventure game files or save files by name.

Many companies offer games on disk. Some of the games need either the Mini-Memory Module or the Extended BASIC cartridge to

run. If you want to play any of these games, you need the disk system and cartridges.

So, to play games on your TI 99/4A, you would need this equipment:

- *Joysticks* (about $24.00 discount) *or* an adapter ($18.00 to $25.00) and whatever joysticks you want ($10.00 and up).
- *Cassette cables* (about $12.00 discount) and a cassette recorder ($30.00 and up).
- As an *alternative* to the cassette and disk drive, you can get the Wafertape peripheral (about $140 list) and the Hexbus interface (about $60.00 list).
- *Only* if you get games that need them, *Extended BASIC* (about $80.00 discount) or the *Mini-Memory Module* (about $80.00 discount).
- *Only* if you are going to play games that come on disk or that require more memory, the *Expansion Box* (about $175 discount), a *Disk Controller* card (about $175 discount), an Expansion System *Disk Drive* (about $275 discount), and/or *Memory Expansion* card (about $215 discount).

This is not an extremely expensive expanded system, by any means (unless you get carried away and *need* the disk and memory). However, you might reasonably purchase your equipment in this order:

1. Get the joysticks so you can play the action games.
2. Get the cassette cables when you get your first game that comes on a cassette. Get a cassette recorder now, if you don't have one.
3. Get the Extended BASIC cartridge or the Mini-Memory Module when you get the first game that requires one of these.

If you stop here, you will spend somewhere around $146, including the cost of the cassette recorder.

You can improve the peformance of your system considerably for less than $200 by getting the Hexbus interface and the Wafertape drive.

If you find that you are really interested in playing the more elaborate games, get the Expansion Box, Memory Expansion card, Disk Controller card, and an expansion system Disk Drive for an additional cost of about $840.

This last purchase is definitely *not* required for you to have a great time playing games on your TI 99/4A.

Case 2: You want to run some of the *cartridge-based educational software.*

Suppose you want to take advantage of the superb educational software available for the TI 99/4A and you don't really want to get into much programming of your own.

Most of the cartridge-based software is self-contained so you will not need anything besides the cartridge. However, some software, especially that written for very young children, requires the Speech Synthesizer.

The LOGO II package requires more memory than the 16K that comes with your TI 99/4A. You will have to get the Memory Expansion card, for which you need the Expansion Box.

To run the various educational packages, you will need:

- *Educational cartridges* ($15.00 to $80.00 each, discount)
- The *Speech Synthesizer* (about $100 discount)
- The *Expansion Box* (about $175 discount) and *Memory Expansion* card (about $215 discount)

You can expand your system in a number of ways, depending on what you want to do first. The least expensive way is to start with the cartridge-based software that does not require any support and then move on to getting LOGO II. You could purchase equipment in this order:

1. Some educational cartridges
2. The Speech Synthesizer

If you stop here, you can run most of the cartridge-based educational programs and you will have spent $100 plus the cost of whatever cartridges you buy.

If you want to use LOGO II, you need the Expansion Box and Memory Expansion card for an additional cost of about $460, including the LOGO II cartridge. (You may be able to find a stand-alone 32K memory expansion unit for as little as $160. This is probably a good deal if you don't want to get a disk drive. It will allow you to run LOGO II for about $240.)

When you run LOGO II, you generate programs. It's nice, though not necessary, to save the programs you write. A cassette tape is usually quite satisfactory for this or you could go for the higher performance and ease-of-use of the Hexbus Wafertape drive.

Case 3: You want to run the PLATO educational software (that's written for grades kindergarten through high school).

You can run the wonderful PLATO programs that were originally developed by Control Data Corporation to run on very large computers. TI offers a wide range of course material running under the PLATO system, starting with courses for kindergarten children and going all the way through high school. (Anyone who completes the entire series of high school PLATO courses should be able to qualify for a GED diploma.)

The PLATO software is offered *only on disks,* so you will need:

- *Expansion Box* (about $175 discount)
- *Memory Expansion* card (about $215)
- *Disk Controller* card (about $175 discount)
- Expansion System *Disk Drive* (about $275 discount)
- PLATO *cartridge* (about $50.00)

There is no way to phase this in. If you want to run PLATO, you have to buy the whole thing all at once. Get the Expansion Box, Memory Expansion card, Disk Controller card, Expansion System Disk Drive, and PLATO cartridge.

This expansion plan cannot be reduced and will cost about $890. This does not include the cost of the individual PLATO courses, which run about $40.00 per course disk.

Case 4: You want to use your TI 99/4A as a *terminal* to an office computer and to access on-line services like TEXNET.

You want access to the on-line services like TEXNET, The Source, or Dow Jones. You need:

- *Expansion Box* (about $175 discount)
- RS-232 *Interface* card (about $125 discount)
- Telephone Coupler or *Modem* (about $100 discount)
- *Terminal Emulator II* cartridge (about $35.00 discount)

There is no way to use your TI 99/4A as a terminal without all of the pieces mentioned above. You can, however, look for alternate modems if you want to save some money.

This expansion plan looks like:

1. Get the Peripheral Expansion Box, RS-232 Interface card, some modem, and the Terminal Emulator II cartridge.
2. Sign on to one or more of the on-line services, like TEXNET.

The communications capability will cost about $435, not including the sign-up costs for the services. Several modem manufacturers give free sign-ups to The Source when you buy their modem.

TI has announced a modem for their Hexbus interface. This modem is due for release in late 1983 and may provide a much less expensive path to communications. We estimate the cost of the Hexbus interface and Hexbus modem that you would require to be about $160 total. Look for this alternative when you are ready to buy.

Case 5: You are going to be a *super programmer.*

You get the basic TI 99/4A console and find that you (or your child) are really interested in programming. It's a great hobby and you *really enjoy the act of programming.* You want to expand your system to support ambitious programming.

Some fundamentals about programming. If you want to do anything complex, you *need* a fast storage device (disk) and a printer. More memory is a good idea, too. To meet these requirements you get:

- *Expansion Box* (about $175 discount)
- *Memory Expansion* card (about $215 discount)
- RS-232 *Interface* card (about $125 discount)
- *Printer* (TI's is $500 discount, though you can spend as little as $200 for a cheap dot matrix printer)
- *Disk Controller* card (about $175 discount)
- Expansion System *Disk Drive* (about $275 discount)

This is a *costly* expansion plan. The equipment listed here costs about $965 plus a printer (figure on about $400). You should be certain that you really want to do some significant programming before you spend this much money.

Once you have bought the equipment, you have to decide what languages you want to use. If you want to stay with BASIC, you should get the Extended BASIC cartridge (about $80.00 discount).

You may also want to do some Assembler work. You can do light Assembler programming (small routines for use with TI BASIC or Extended BASIC) with the *Mini-Memory Module* (about $80.00 discount). For more ambitious Assembler work, you will need the *Editor/Assembler* (about $75.00 discount).

5.8.1 Your Own System

These examples are just that—examples. Everyone has different needs when it comes to expanding his or her computer. Only you can decide what you need and when you need to buy it.

We would like to stress that you plan for expansion. It's very easy to get caught up in getting new "toys" and end up with a set of mismatched items—things that are great but not what you need. So, follow these steps when you are going to get more components:

1. *Decide* what you want to do.
2. *Find out* what peripherals you need to *meet that objective.*
3. *Find out* what the various peripherals *cost.*
4. See if you can *"phase in"* your equipment purchase. Maybe you can buy one or two items now and the rest in few weeks or months.
5. Make an equipment purchase *plan* and stick to it. Don't get carried away by the bells and whistles that salespeople show you. Decide what you want. Make revisions to your plans if you see something that really is better. And stick to your own plans.

6

BEYOND BASIC: OTHER PROGRAMMING LANGUAGES

In this chapter, we tell you what languages are available for the TI 99/4A and show you situations where other languages are more appropriate than BASIC. Most of these other programming languages require some system expansion, such as more memory, disks, or the P-Code card.

Our objective is to give you some idea of the structure and suitability of the other languages you can use on your TI 99/4A.

6.1 PROGRAMMING LANGUAGES FOR THE TI 99/4A

The TI 99/4A offers you access to many different programming languages. This is not typical of many home computers. On the TI 99/4A you can use TI BASIC or:

- Extended BASIC
- LOGO II
- 9900 Assembly Language
- UCSD Pascal
- PILOT
- FORTH

But what's wrong with TI BASIC? If you want to use another language, you have to take the time and trouble to learn it, just as you did with TI BASIC. Why not use TI BASIC for everything?

For most common applications TI BASIC (the one that comes in the TI 99/4A console) is perfectly adequate. If your programs are not too large, if they don't need animated graphics, and if they don't depend on fast execution (like arcade games do), TI BASIC is fine for you. You can forget about all these other languages and enjoy yourself with TI BASIC. (Though, you might want to get Programming Aids I to gain fancier control of the screen.)

On the other hand, you may be hitting the limits of TI BASIC. Some problems you may face are:

- *Not enough memory* (despite your best efforts to conserve it) to get your program running.
- *Limited access to* the TI 99/4A *graphics* capabilities, especially Sprites (moving objects on the screen).
- TI BASIC is *too slow* to do what you need.
- BASIC is *not well suited* for younger programmers.
- You want *access to the full power* of your TI 99/4A.

These same obstacles confront users of many of today's home computers. Unfortunately for most of them, there is no way around these obstacles. Fortunately for you, TI 99/4A users have many ways around them.

In the following sections, we will take a look at each of the language offerings for the TI 99/4A. We will give you some idea when and where to use these languages and how much time, talent, and money they require.

6.2 EXTENDED BASIC

If we are talking about Beyond BASIC, then why include Extended BASIC? Because Extended BASIC is such an incredible advance over TI BASIC that it deserves consideration as an alternative to proceeding, for example, to Pascal or Assembly Language.

Chapter 4 contains more information on Extended BASIC. You should read that chapter to find out exactly what extensions are in Extended BASIC. Some highlights are:

- *Multiple statements per line* save memory and contribute to faster execution.
- *Location controlled screen input and output* allow you to do "forms" (sometimes called "full screen") oriented applications like you see in banks and airline reservation systems.

- *Formatted output* allows you to generate professional looking reports.
- *Enhanced error handling* lets you take control of and correct errors that occur during execution.
- *Improved program control structures* (IF-THEN-ELSE and named subroutines with true parameter passing and local variables) make it easier to write complex programs and result in more reliable, tighter code.
- Full *Sprite* control (up to 28 moving objects on the screen) makes it possible to write arcade style games and educational programs for children.
- Easy access to the *Speech Synthesizer.*
- Access to the *Memory Expansion* card so that you can write larger programs.
- Direct and easy *access to Assembly Language routines* means you can write some support routines in Assembler to improve program execution time or do things not possible in BASIC.

With these facilities, Extended BASIC represents a real alternative to the other advanced programming languages available for your TI 99/4A. It's cheaper than Pascal, easier to learn than Assembly Language or FORTH, and faster and more general purpose than LOGO or PILOT.

Its only slight limitation is an inability to access the advanced graphics modes possible on the TMS9918A Video Display Processor (that is the chip that handles the screen display). If you really need high resolution graphics, you will have to use Assembly Language or FORTH.

6.3 LOGO II

LOGO is a delightful, dynamic language designed at MIT specifically for children. LOGO makes heavy use of sprites, graphics, and sound in an easily learned language that even younger children can master.

If you, or someone in your family, is computer-shy, LOGO is the language for you. As it entertains, it teaches problem-solving logic, geometry, spacial relationships, perserverance, and mental discipline. Not bad for a turtle that crawls around on the screen leaving a trail behind it.

LOGO Is easy to use. Consider this example of how you activate a sprite:

TELL SPRITE 10	(activates sprite 10 and causes it to obey the commands that follow)
CARRY :ROCKET	(tells sprite 10 to take the shape of a rocket)
SETCOLOR :WHITE	(makes sprite 10 white)
SETHEADING 90	(prepares the sprite to go to the right)
SETSPEED 20	(starts the sprite moving at speed 20)

Children, especially, love LOGO because it's something they can work at and master. It's highly interactive, very much into graphics, and a lot of fun to play with.

If you have children, you need LOGO. If you buy LOGO, be sure to get LOGO II. LOGO II is much better than the original LOGO and includes a lot of new features. LOGO requires the Memory Expansion card.

6.4 9900 ASSEMBLY LANGUAGE

Assembly Language is the closest you can (or want to) come to pure machine code. Each statement in an Assembly Language program is translated into a single machine instruction. (Not very many programmers think in the ones and zeros that make up real machine code, or the "easier" hexadecimal notation, either.)

Even simple tasks in BASIC—like writing some data to a cassette or adding two numbers—translate into many instructions and a lot of detail work in an Assembly Language program.

So why bother with Assembly Language? Mainly because it's *very powerful and very fast.* Assembly Language lets you use *all* the capabilities of the TI 99/4A, including such features as ultra-high-resolution, Bit-Mapped graphics and interrupt handling.

For some games—Parsec is an example—you have got to write in Assembly Language. Nothing else is fast enough or flexible enough to handle all the action.

Another valuable use for Assembly Language is in *subroutines* that you call from BASIC or Pascal. The Assembly Language routine can carry out tasks impossible to do in the other language. Sometimes it takes too long for BASIC or Pascal to do something you need done

quickly. An Assembly Language subroutine can make short work of the job.

Texas Instruments offers two ways for you to get Assembly Language programs into your TI 99/4A:

- The Mini-Memory cartridge
- The Editor/Assembler package

6.4.1 Mini-Memory Cartridge

The Mini-Memory cartridge includes a line-by-line assembler and is the less expensive alternative. All you need is the cartridge (about $80.00 discount).

You enter the Assembly Language program one line at a time. Each line is translated and stored in memory as it is entered. Obviously, you are not about to develop extensive Assembly Language systems this way. But it's a good way to create those small service subroutines that are so handy when called from BASIC.

The Mini-Memory approach is also an inexpensive way for you to get a feel for Assembly Language programming. If you find you like programming in Assembly Language, proceed to the more expensive Editor/Assembler.

6.4.2 The Editor/Assembler

The Editor/Assembler package is a full 9900 Assembly Language program development system. It includes an *Assembler* that produces relocatable object code and a relocating linking loader that resolves external references between independently assembled programs.

All this probably does not mean much to you unless you are an experienced Assembly Language programmer. It's important to point out, though, that these facilities are extremely sophisticated and entirely unique in a low cost home computer. They promote use of highly modular system designs, thus making it easier to write Assembly Language programs.

The Editor/Assembler is not cheap. It's not so expensive by itself—about $75.00 discount—but it requires the Memory Expansion card and at least one disk drive. If you intend to do anything beyond the trivial, you will also need a printer and the RS-232 card that goes with it.

6.5 UCSD PASCAL

Pascal is one of those elegant languages developed shortly after the philosophy of "structured programming" gained widespread acceptance in professional programming circles. It is widely used in programming courses at colleges and universities because it introduces many concepts not present in less sophisticated languages.

Structured programming methods call for languages containing block-oriented program control statements. These statements determine what other statements will be executed (like the IF. . . THEN or ON. . . GOTO statements in BASIC). Pascal contains a multitude of *block-oriented control statements,* such as:

- BEGIN . . . END
- CASE
- REPEAT . . . UNTIL
- FOR . . . END (like FOR . . . NEXT in BASIC)
- Internal and external procedures (subprograms)

Pascal is also what is called a "heavily typed" language. No, that does not mean you have to hit the keys harder. What it does mean is that all of your variables must be *explicitly declared as a certain type* (for example, a REAL number, or a fixed length character string). Interaction between types is then carefully controlled.

Pascal also includes *advanced data types* like arrays, varying length strings, sets, and RECORDS that make it easier to solve many commonly encountered programming problems.

These features allow you to create clear, elegant, and more easily debugged programs than is possible with BASIC programs.

Of course, all this is not free. Pascal is a complex language that is less intuitive than BASIC and hence harder to learn.

To run Pascal on your TI 99/4A you need the P-Code card and the Memory Expansion card for your Expansion Box. This earns you the right to *execute existing Pascal programs* that have been compiled for the p-System.

If you actually want to *write Pascal programs,* you also need at least one disk drive and some disk-based software. The total cost of all this, not counting the Expansion Box, Memory Expansion card, or disk drive, is about $350 (discount).

Pascal programs compiled under the UCSD p-System are not converted to machine language. They are, instead, converted to an intermediate language, called p-code, that can be executed by a p-code

interpreter running on any computer. The p-code is always the same, it's only the interpreter that changes from computer to computer.

Since the p-code interpreter is relatively cheap and easy to write, the UCSD p-System is now running on a large number of mainframes, minicomputers, and microcomputers. This means a UCSD Pascal program developed on a DEC VAX 11/780 (a very powerful minicomputer) can probably be run on your TI 99/4A with no change.

It also means that a UCSD Pascal program that you develop on your TI 99/4A can be easily transferred to any computer that supports the UCSD p-System. This is what is known as *program portability.*

Even if a particular system does not support, or for some reason can't run, the UCSD p-System, many systems do support a Pascal compiler of their own. Going to one of these compilers could require some rewriting, but Pascal is pretty well standardized so any rewrites should not be terribly difficult.

Although the cost to implement a full UCSD Pascal development system on the TI 99/4A is high, you should consider the Pascal system if you are planning to develop programs for distribution on other systems.

You might also consider Pascal if you are a serious amateur or if you have a precocious child with a strong interest in computer programming. Just make sure that you choose the language most appropriate to the use.

6.6 PILOT

PILOT is a special purpose language that allows you to easily develop *Computer Aided Instruction (CAI)* lessons. TI PILOT includes special instructions that let you use the TI 99/4A's sprites, sound, and powerful graphics.

TI PILOT is obviously intended for teachers interested in developing powerful and creative courseware. It is not a general purpose language.

TI PILOT requires the Memory Expansion card, the P-Code card, and at least one disk drive.

6.7 FORTH

FORTH is a programming language that only a professional programmer could love. It is a higher level language (higher than Assembly Language) intended for serious TI 99/4A system development.

FORTH is a dense, but very flexible and expandable, language that results in fast-executing machine code (as opposed to BASIC or Pascal which must be interpreted).

FORTH is for the professional systems developer or the serious (and talented) amateur. It provides you with access to all of the power of the 9900 microprocessor, including interrupt handling, and allows full use of the TMS9918A Video Display Processor (that controls your screen).

FORTH is definitely an alternative to Assembly Language.

6.8 CONCLUSIONS

No one programming language will ever be suitable for all the possible programs that you want to write. Your best approach is to become comfortable with programming and then pick the languages most suitable to your needs and talent.

7

GRAPHICS

In this chapter, we show you how to do some simple graphics using TI BASIC and discuss the other types of graphics available on your TI 99/4A.

The graphics processor in your TI 99/4A is capable of much more than TI BASIC can make it do. We tell you what software you will need to make full use of dynamic, high resolution graphics on your TI 99/4A.

7.1 SCREEN CONTROL

Running a screen display is a full time job for any processor. For this reason, the screen display on your TI 99/4A is *not* controlled and maintained by its TMS9900 CPU.

Computer manufacturers include in their products *dedicated special purpose chips* whose sole function is to maintain the image you see on the screen. In the TI 99/4A, that dedicated processor is the TMS9918A *Video Display Processor.*

It is, in fact, the "A" in this name that puts the "A" in TI 99/4A. The older TI 99/4 is equipped with the TMS9918 Video Display Processor. The newer "A" processor has more features than the older processor.

The Video Display Processor determines:

- The quality of the display (sometimes called *resolution*)
- The number of colors that can appear
- The choice of display modes
- The support for sprites (moving objects)

No single one of these features determines the overall power and performance of the Video Display Processor. It's the ability to combine and control powerful features that determines how good a Video Display Processor really is. The TMS9918A is one of the best.

7.2 DISPLAY RESOLUTION

Resolution is a fancy term that represents, in some numbers, the potential *quality of the image*. The higher the resolution, the higher the quality of the resulting image.

Think of resolution as the number of small squares on the screen that you color in to make a picture. If you had, say, 20 squares across by 10 down you could not make nearly as fine a picture as you could if you had 200 squares across by 100 down.

Resolution is expressed in *pixels* (*pic*ture *el*ements), the little colored dots that make up a complete image on the screen. The maximum resolution of the TMS9918A chip in your TI 99/4A is:

256 pixels across
by
192 pixels down

Thus, the image on your screen is composed of 49,152 separate dots. This resolution is very near the maximum that can be displayed on an ordinary television screen.

With high resolution, you can create smooth curved lines that don't have that "stepped" look you frequently see in lower resolution images. Many games, educational programs, and business or personal graphics programs require this high resolution.

7.3 COLORS

The TI 99/4A can display up to 16 *colors* on the screen at the same time (see Table 7-1 for a list of these colors). This is typical of most home computers and more than adequate for nearly all applications.

The level of control you have over which colors will appear at what locations on your screen is determined by the *mode* in which you are operating.

Color control is just as important as the number of colors you can display. We will talk more about coloring the screen when we discuss the various TMS9918A modes later in this chapter.

Table 7-1 TI 99/4A Color Codes

Color	Code Number
Transparent	1
Black	2
Medium green	3
Light green	4
Dark blue	5
Light blue	6
Dark red	7
Cyan (bluish green)	8
Medium red	9
Light red	10
Dark yellow	11
Light yellow	12
Dark green	13
Magenta (purplish red)	14
Gray	15
White	16

7.4 DISPLAY MODES

The TI 99/4A can operate in any one of four different *display modes:*

- Graphics Mode (used by BASIC)
- Text Mode
- Multicolor Mode
- Bit-Mapped Mode

Each *mode* offers a different level of control over the image on the screen. These levels vary from the simplest text-only with very limited color control (Text Mode) to the high resolution mode (Bit-Mapped Mode) that lets you assign *two colors* to every *eight pixels* on the screen. (Remember: there are 49,152 pixels on the screen.)

You can use all four modes from Assembler, but you can use only Graphics Mode from BASIC.

In the following sections, we will give you an idea of what can be done in each display mode. Actually doing it, though, is often very complicated, detailed work. If you need to know how to do graphics beyond what can be done in TI BASIC or Extended BASIC, you will have to get the TI Editor/Assembler package.

7.4.1 Graphics Mode

Graphics Mode, the mode available from BASIC, gives you a good deal of color control and lets you define your own 8-by-8 dot characters.

In Graphics Mode, the screen is divided into:

32 columns
by
24 rows
of 8-by-8 dot characters
(768 characters)

Graphics Mode is the standard display mode for TI BASIC and Extended BASIC. Why Graphics Mode? Why not Text Mode? Because Graphics Mode both displays characters (letters and numbers) *and* offers some support for doing screen graphics.

(If Text Mode was the standard mode for BASIC, you would only be able to write letters and numbers on your screen. You would not be able to do any screen graphics at all—not even simple things like drawing colored stripes on the screen.)

Each character position on the screen is an 8-by-8 dot, two-color image. A *table in memory* determines which *two colors* will appear in the character. Each dot can be turned "on" or "off" to determine which color to use for the dot. "On" dots (those set to 1) become one color and "off" dots (those that are 0) the other.

Fig. 7-1 shows you how to draw a letter "A" and a heart using on and off dots in an 8-by-8 grid. This is how character images are formed on the screen.

You may know that the screen characters used in the TI 99/4A have coded values, called *ASCII codes,* assigned to them. These codes are simply numbers in the range 0 to 255. ASCII codes are sometimes expressed as *hexadecimal* (base 16) values, especially in Assembler.

For example, the letter P (capital P) has the value 80 decimal (50 hexadecimal). The letter Q has the value 81 decimal (51 hexadecimal). There is nothing magic about these values. They were chosen for convenience. Isn't it convenient that the letter Q has a value one more than the letter P—just like in the alphabet? This makes it easy to arrange things in alphabetical order; when you compare a "P" to a "Q", as in this BASIC statement:

```
IF "P" > "Q" THEN 500
```

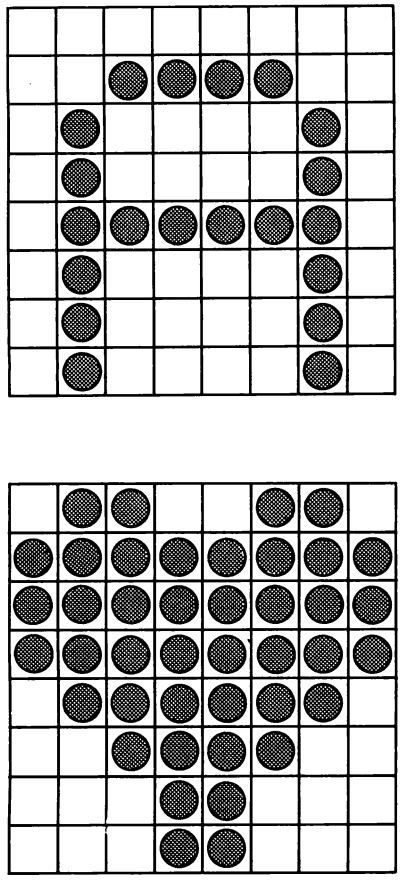


Fig. 7-1. 8×8 graphics mode character definition diagrams.

You are really comparing 80 to 81, as in this statement:

```
IF 80 > 81 THEN 500
```

When these ASCII codes (representing a character) are written to the screen as in:

```
PRINT "P"
```

or

```
PRINT CHR$(80)
```

they determine which of the 256 (0 to 255) possible 8-by-8 dot patterns appears. If you write an 80 decimal (PRINT CHAR$(80)), you

get a pattern that looks like a capital P. Write an 81 instead, and you get the pattern that looks like a capital Q.

Changing Patterns—One nice feature in Graphics Mode is you can *change the patterns* that correspond to the ASCII values. You use TI BASIC's CHAR subprogram to change the patterns.

Suppose you don't like the letters P and Q. You think the original designer of the alphabet made a mistake that you are going to correct. You are going to switch the two characters. In your personal alphabet P is going to look like the Q of everyone else's alphabet, and vice versa. To do that, enter the code shown in Listing 7-1.

As you can see, all the P's on your screen suddenly become Q's, and vice versa. But their ASCII values remain the same—it's only the *image* of the character that changes. If you compare a P to a Q (as in IF "P" > "Q" THEN 500) you get the same result as you did before you exchanged their images.

Listing 7-1. Example program—Ps and Qs

```
100 REM THIS PROGRAM EXCHANGES THE IMAGES OF CHARACTERS
    P AND Q
110 REM
120 REM THE P$ AND Q$ VARIABLES HOLD THE PATTERNS FOR
    THE LETTERS
130 REM
140 P$="0078444478404040"
150 Q$="0038444444544834"
160 CALL CLEAR
170 PRINT "P Q P Q P Q P Q"
180 PRINT
190 INPUT "HIT ENTER AND WATCH THE P'S AND Q'S":X$
200 PRINT
210 REM
220 REM CALL TO CHANGE THE  CHARACTER PATTERNS
230 REM
240 CALL CHAR(80,Q$)
250 CALL CHAR(81,P$)
260 PRINT "P Q P Q P Q P Q"
270 PRINT
280 INPUT "HIT ENTER AND WATCH THE P'S AND Q'S":X$
290 PRINT
300 REM
310 REM NOW CHANGE THEM BACK
320 REM
330 CALL CHAR(80,P$)
340 CALL CHAR(81,Q$)
350 PRINT "P Q P Q P Q P Q"
360 PRINT
370 INPUT "HIT ENTER TO END.":X$
```

You can use this *character definition capability* to create larger images on the screen built from individually tailored 8-by-8 characters. This works very much like putting together a jigsaw puzzle but first you have to build the pieces. The Tic-Tac-Toe program in Appendix C shows what you can do from TI BASIC with a few simple Graphics Mode commands.

7.4.2 Text Mode

Text Mode is a monochrome (one screen color, one character color), characters-only display mode. Its advantage over Graphics Mode is that it creates a screen image that is:

40 columns
by
24 rows
of 6-by-8 dot characters
(960 characters)

In Text Mode, your color control is limited to:

- *one background* (screen) color
- *one dot color* for all characters

Text Mode is handy for displaying text data (for example, a word processing application). In Graphics Mode, characters are written in an 8-by-8 dot character square (see Fig. 7-1). In Text Mode, the characters are formed in a 6-by-8 dot matrix, as you can see in Fig. 7-2.

This 6-by-8 matrix makes for slightly less well-formed, but still very readable, upper and lower case characters. Fig. 7-2 shows you the same letter "A" as in Fig. 7-1, but with the lower resolution of Text Mode. You can define your own characters in Text Mode, just as you can in Graphics Mode.

Unlike Graphics Mode, Text Mode allows only *one* combination of background/dot colors that applies to *all* the characters on the screen. This is not really a limitation if you are doing genuine text work, but it is the reason BASIC operates in Graphics Mode.

Text Mode is not available from BASIC, but is used by the Terminal Emulator II (communications) and Editor/Assembler cartridges.

7.4.3 Multicolor Mode

Multicolor Mode is a medium-resolution graphics mode in which the screen is divided into:

64 columns
by
48 rows
of 4-by-4 dot boxes
(3072 boxes)

You can individually assign each little 4-by-4 dot box its own color. In Multicolor Mode you can "draw" a very fine picture.

You cannot easily write normal characters to the screen as you can in Graphics and Text Mode. If you want to write a message on the screen, you have to "draw" the characters like you draw anything else you put on the screen in Multicolor Mode.

To use Multicolor Mode, you will have to go to a language other than BASIC. Assembler provides access to Multicolor Mode.

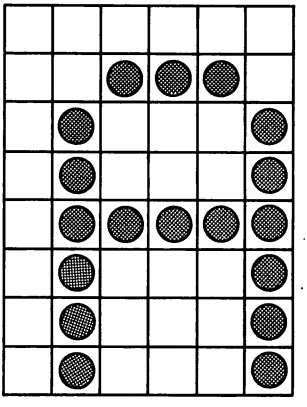


Fig. 7-2. 6×8 text mode character definition diagram.

7.4.4 Bit-Mapped Mode

Bit-Mapped Mode provides the highest resolution, most finely detailed graphics possible on your TI 99/4A. In Bit-Mapped Mode, the screen is divided into:

32 columns
by
192 rows
of 8-dot strips
(6144 strips)

Each dot in an 8-dot strip can be *one of the two colors* assigned specifically to that strip. The "on" dots in the strip appear as one of

the colors assigned to it, while the "off" dots appear as the other color.

This coloring scheme allows you to draw an extremely detailed scene.

Bit-Mapped Mode is not available from BASIC. If you need Bit-Mapped Mode, you will have to work in a lower level language like Assembler.

7.5 SPRITES

Sprites are those moving objects, like the little guys in Munch Man or the ships in Parsec, that are independent of whatever else is on the screen. The TI 99/4A can place as many as 32 sprites on the screen at one time. *Extended BASIC* supports sprites, but can handle only 28 sprites.

Sprites come in four flavors:

1. *Normal sprites* are the size of a character (8-by-8 dots).
2. *Magnified sprites* are defined like normal sprites but occupy four character positions (in a 2-by-2 square) instead of one (each dot becomes a 4-dot square).
3. *Double sprites* are the size of four characters and are defined in a 16-by-16 dot square.
4. *Magnified double sprites* are defined like double sprites but are enlarged to occupy a 32-by-32 dot square (again, each dot becomes a 4-dot square).

It's somewhat difficult to visualize all these options so we have included a *Sprite Editor* program in Appendix C. In order to run this program, you will need Extended BASIC. We encourage you to enter this program and experience first hand the options you have when creating Sprites.

Sprites also have some other fine qualities:

- *Each* sprite can be *any* of the 16 colors.
- They can be made to *move automatically* in any direction over a wide *range of speeds*. Your program does not have to continually change the position of a sprite on the screen, it changes position automatically in the direction and at the speed you set for it.

Unfortunately, sprites are not available from TI BASIC. You can make full use of sprites from other languages, including Extended BASIC. LOGO is one of the best languages to use to gain access to

sprites—it is especially good for children and very inexperienced programmers.

You can use sprites in every display mode except Text Mode. In Bit-Mapped Mode, you can use sprites, but you cannot use their automatic motion feature.

Many of the games you can get for your TI 99/4A could not exist were it not for sprites. Parsec, for example, would not be possible without automatic motion sprites.

Sprites are a very powerful feature. If you expect to write games or other programs that rely on dynamic graphics, learn to use sprites.

7.6 CONCLUSIONS

Table 7-2 contains a summary of the TI 99/4A display modes. As you can see, the TI 99/4A has powerful and dynamic graphics capability.

Table 7-2 Display Mode Comparison Chart

Mode	Element Size (dots)	Color Control	No. of Screen Elements (rows by cols)
Graphics	8 by 8	16 sets of 2 colors each	32 by 24
Text	6 by 8	1 set of 2 colors	40 by 24
Multicolor	4 by 4	1 color per element	64 by 48
Bit-Mapped	8 by 1	2 colors per element	32 by 192

Note: Sprites are not allowed in Text Mode. Sprites are allowed in Bit-Mapped Mode, but their automatic motion feature cannot be used.

Some of this capability is accessible from TI BASIC, and more from Extended BASIC. If you really want to make full use of the TI 99/4A's graphics features, you will have to use Assembler.

What does this mean to those of you not interested in doing your own graphics work? It means professional software writers have a sufficiently powerful machine in the TI 99/4A to produce programs that are interesting, complex, and visually exciting.

8

SOUND ON YOUR TI

You can use your TI 99/4A to generate tones, sound effects, music, and even speech.

In this chapter, we tell you about the kinds of sounds you can get from your TI and what hardware and software you need to make your TI speak to you. We also give you a few BASIC programs that generate sounds.

8.1 THE SOUNDS OF YOUR TI

Producing sound is a specialized job that requires a dedicated, special purpose sound chip. The sound chip included in your TI 99/4A console is the TMS9919 *Sound Generator Controller chip.*

Producing speech is an even more demanding task. It requires a very sophisticated speech synthesis chip and a series of complex codes to represent words. The Solid State Speech Synthesizer peripheral for the TI 99/4A contains the TMS5200 *Voice Synthesis Processor* chip and a stored vocabulary of more than 300 words. You can also build your own words and have them spoken by the speech synthesizer.

You can use BASIC or Extended BASIC (or other languages like Assembler) to program your computer to produce these distinct sounds:

- *Music*—your computer generates a *tone* that it sustains for a specified time at a selected volume. You can generate up to three tones at one time. You can make just about any musical tone with your TI 99/4A.

- *Noise*—your computer generates one of eight possible noises, four are called "periodic noises" and four are called "white noises." You can mix one noise with up to three tones at the same time.
- *Speech*—your computer talks to you through the Speech Synthesizer peripheral. You need Extended BASIC or one of the cartridges that generate speech to use the Speech Synthesizer.

8.2 MAKING SOUNDS

The TMS9919 *Sound Generator Controller* chip in your TI 99/4A generates sounds that are:

- In the *frequency range* 110 to 55,938 hertz (Hz, or cycles per second)
- Of a *duration* ranging from 1 to 4250 milliseconds (0.001 through 4.25 seconds)
- On a *volume* scale of 0 to 30 (0 is the loudest, 30 the quietest)

8.2.1 Music and Tones

Tones are sounds of a specific frequency (110 to 55,938 Hz) that can be generated at a selected volume (0 to 30) for a specified amount of time (1 to 4250 milliseconds). You use tones to make music or sound effects. BASIC can generate tones only up to 44,733 Hz. This is not a real limitation because few people can hear frequencies higher than 11,000 Hz.

Your computer continues executing statements while it's generating sounds. That means that you can write programs that interact with the user while your computer is playing music or making appropriate game sounds.

Don't worry about learning some special language to generate these sounds. You can control up to *three tones* and *one noise* at the same time from BASIC through the SOUND subprogram. Table 8-1 shows you the frequency values for some of the musical notes.

You make tones by using the SOUND subprogram in BASIC, like this:

CALL SOUND *(duration, tone, volume)*

where:

Duration is the time in milliseconds (1 to 4250).
Tone is the frequency in hertz (110 to 44,733).
Volume is the volume control (0 = loudest to 30 = quietest).

It's not at all difficult to get your computer to play notes. For example, to get the note middle C (frequency 262) played at middle volume (15) for 2 seconds (2000 milliseconds), you:

```
CALL SOUND(2000,262,15)
```

Table 8-1 Frequencies for Musical Notes

Frequency	Note	Frequency	Note
110.00	A	698.46	F
116.54	A flat, B sharp	739.99	F sharp, G flat
123.47	B	783.99	G
130.81	C (low C)	830.61	G sharp, A flat
138.59	C sharp, D flat	880.00	A (Above high C)
146.83	D	923.33	A sharp, B flat
155.56	D sharp, E flat	987.77	B
164.81	E	1046.50	C
174.61	F	1108.73	C sharp, D flat
185.00	F sharp, G flat	1174.66	D
196.00	G	1244.51	D sharp, E flat
207.65	G sharp, A flat	1318.51	E
220.00	A (below middle C)	1396.91	F
233.08	A sharp, B flat	1479.98	F sharp, G flat
246.94	B	1567.98	G
261.63	C (middle C)	1661.22	G sharp, A flat
277.18	C sharp, D flat	1760.00	A
293.66	D	1864.66	A sharp, B flat
311.13	D sharp, E flat	1975.53	B
329.63	E	2093.00	C
349.23	F	2217.46	C sharp, D flat
369.99	F sharp, G flat	2349.32	D
392.00	G	2489.02	D sharp, E flat
415.30	G sharp, A flat	2637.02	E
440.00	A (above middle C)	2793.83	F
466.16	A sharp, B flat	2959.96	F sharp, G flat
493.88	B	3135.96	G
523.25	C (high C)	3322.44	G sharp, A flat
554.37	C sharp, D flat	3520.00	A
587.33	D	3729.31	A sharp, B flat
622.25	D sharp, E flat	3591.07	B
659.26	E	4186.01	C
		4434.92	C sharp, D flat

You can control volumes for up to three tones and one noise with a single call to SOUND, like this:

```
CALL SOUND (duration,tone1,vol1,tone2,vol2,tone3,vol3)
```

Suppose you want to play these three tones at the same time:

- Middle C for 2.5 seconds at middle volume

- A below middle C at low volume
- A above middle C at low volume

You do this in BASIC by:

```
CALL SOUND(2500,262,15,220,25,440,25)
```

You can tell your computer to wait until all previous sound(s) are finished before playing its new sound, or you can tell it to stop playing its current sound(s) and begin the new sound(s) immediately. To make the new sound start immediately, you supply a *negative duration* (−1 to −4250).

Suppose you want to play the notes above, execute some other BASIC statements, and, when the sounds are finished, play a middle C at full volume for one second. In BASIC, you would use:

```
CALL SOUND(2500,262,15,220,25,440,25)
          .
          .
          .
some other BASIC statements
          .
          .
          .
CALL SOUND(1000,262,0)
```

If you want to stop the previous sounds when the BASIC statements are done executing and play a high C for 4 seconds at full volume, you would use:

```
CALL SOUND(2500,262,15,220,25,440,25)
          .
          .
          .
some other BASIC statements
          .
          .
          .
CALL SOUND(−4000,523,0)
```

To help you get used to using the SOUND features of your TI 99/4A we have given you some programs. The program in Listing 8-1 asks you for the frequency, duration, and volume of the note you want to play. The program in Listing 8-2 lets you play up to three tones.

Listing 8-1. Single tone generator example program.

```
100 REM THIS PROGRAM SHOWS YOU HOW TO MAKE A SINGLE
    TONE
120 REM
140 REM  IT ASKS FOR THE DURATION, FREQUENCY, AND
    VOLUME
160 REM AND CHECKS THAT THE VALUES ARE IN THE RIGHT
    RANGES
180 REM
200 REM FIRST, TELL WHAT'S HAPPENING
220 REM
240 CALL CLEAR
260 GOSUB 1000
280 REM
300 REM
320 REM NOW, GET THE DURATION
340 REM
360 GOSUB 2000
380 REM
400 REM NOW, GET THE FREQUENCY
420 GOSUB 3000
440 REM
460 REM FINALLY, GET THE VOLUME
480 GOSUB 4000
500 REM
520 REM AT LAST, PLAY THE TONE
540 REM
560 CALL SOUND(DURATION,FREQUENCY,VOLUME)
580 REM
600 REM SEE IF MORE
620 REM
640 INPUT "TRY AGAIN? (Y/N) ":YESNO$
660 IF YESNO$="Y" THEN 360
680 STOP
1000 REM
1020 REM TELL WHAT TO ENTER
1040 REM
1060 PRINT :"YOU CAN GET A TONE FOR 1 TO 4250
     MILLISECONDS"
1080 PRINT "(.001 TO 4.25 SECONDS)"
1100 PRINT
1120 PRINT "IF YOU ENTER A POSITIVE     VALUE (1 TO
     4250), THE TONE WAITS UNTIL THE CURRENT TONE ENDS"
1140 PRINT :"A NEGATIVE VALUE (-1 TO     -4250) MEANS
     THE TONE STARTS IMMEDIATEL Y"
1160 PRINT :"YOU CAN GET A FREQUENCY FROM 110 TO 44733"
1180 PRINT "PEOPLE CAN'T HEAR MUCH ABOVE 11000"
1200 PRINT
1220 PRINT "YOU CAN GET A VOLUME OF 0 TO 30 (0 IS
     LOUDEST)"
1240 PRINT
1260 RETURN
```

Listing 8-1. Cont—Single tone generator example program.

```
2000 REM
2020 REM GET THE DURATION
2040 REM
2060 INPUT "ENTER THE TIME (-4250 TO 4250) ":DURATION
2080 REM
2100 REM CHECK FOR CORRECT VALUES
2120 REM
2140 IF DURATION<-4250 THEN 2280
2160 IF DURATION>4250 THEN 2280
2180 REM
2200 REM GOOD VALUE
2220 REM
2240 RETURN
2260 REM
2280 REM BAD VALUE
2300 REM
2320 PRINT
2340 PRINT "TIMES MUST BE BETWEEN -4250 AND 4250"
2360 PRINT
2380 GOTO 2060
3000 REM
3020 REM NOW, GET THE FREQUENCY
3040 REM
3060 INPUT "ENTER FREQUENCY (110 TO 44733) ":FREQUENCY
3080 REM
3100 REM CHECK FOR CORRECT VALUE
3120 REM
3140 IF FREQUENCY<110 THEN 3260
3160 IF FREQUENCY>44733 THEN 3260
3180 REM
3200 REM GOOD VALUE
3220 REM
3240 RETURN
3260 REM BAD VALUE
3280 PRINT
3300 REM
3320 PRINT "FREQUENCY MUST BE BETWEEN 110 AND 44733"
3340 PRINT
3360 GOTO 3060
4000 REM
4020 REM GET THE VOLUME
4040 REM
4060 INPUT "ENTER THE VOLUME (0 TO 30) ":VOLUME
4080 REM
4100 REM CHECK FOR GOOD VALUE
4120 REM
4140 IF VOLUME<0 THEN 4260
4160 IF VOLUME>30 THEN 4260
4180 REM
4200 REM GOOD VALUE
4220 REM
4240 RETURN
```

Listing 8-1. Cont—Single tone generator example program.

```
4260 REM
4280 REM BAD VALUE
4300 REM
4320 PRINT
4340 PRINT "VOLUME MUST BE BETWEEN 0 AND 30":"TRY
     AGAIN"
4360 PRINT
4380 GOTO 4060
4400 END
```

To get started playing real music, try the programs in Appendix C called *Twinkle* and *Hot Cross Buns.*

8.2.2 Noises

You also use the SOUND statement to make these noises by using a negative *tone* value (−1 to −8):

- Three "periodic noises" (−1 to −3)
- One periodic noise that varies with the frequency of the third tone in the SOUND statement (−4)
- Three "white noises" (−5 to −7)
- One white noise that varies with the frequency of the third tone in the SOUND statement (−8)

Each set of four noises starts with the highest pitch in the set and gets progressively lower. That means that noise with a tone −3 is lower in pitch than the noise with a tone −1. Similarly, −7 tone noise sounds lower than −5.

It's impossible to describe what a noise is. You have to hear it to know what it is. You use the noises to make sound effects. The white noises are especially good to make explosion sounds. Listing 8-3 gives you a small program that lets you try out the various noise effects.

You can mix noise and tones by using a noise value (−1 to −8) for the frequency or *tone* variable in the CALL SOUND statement. You can also control up to three tones and one noise in a single CALL SOUND statement like this:

CALL SOUND *(duration,tone1,vol1,tone2,vol2,tone3,vol3,noise,vol)*

Listing 8-2. Multiple tone example program.

```
 100 REM THIS PROGRAM SHOWS YOU HOW TO MAKE TONES
 120 REM
 140 REM  IT ASKS FOR THE DURATION, FREQUENCY, AND
     VOLUME
 160 REM AND CHECKS THAT THE VALUES ARE IN THE RIGHT
     RANGES
 180 REM YOU CAN ENTER UP TO THREE TONES AND VOLUMES
 200 REM
 220 REM FIRST, TELL WHAT'S HAPPENING
 240 REM
 260 CALL CLEAR
 280 GOSUB 1000
 300 REM HOW MAY TONES
 320 INPUT "HOW MANY TONES DO YOU WANT (1 TO 3)!"
     :NUMTONES
 340 IF (NUMTONES<1)+(NUMTONES>3)THEN 380
 360 GOTO 460
 380 PRINT "PLEASE ANSWER CORRECTLY."
 400 GOTO 320
 420 REM
 440 REM
 460 REM NOW, GET THE DURATION
 480 REM
 500 GOSUB 2000
 520 REM
 540 REM NOW, GET THE FREQUENCIES AND VOLUMES
 560 ON NUMTONES GOSUB 5000,6000,7000
 580 REM
 600 REM AT LAST, PLAY THE TONE
 620 REM
 640 ON NUMTONES GOTO 660,700,740
 660 CALL SOUND(DURATION,FREQ1,VOL1)
 680 GOTO 760
 700 CALL SOUND(DURATION,FREQ1,VOL1,FREQ2,VOL2)
 720 GOTO 760
 740 CALL SOUND(DURATION,FREQ1,VOL1,FREQ2,VOL2,FREQ3,
     VOL3)
 760 REM
 780 REM SEE IF MORE
 800 REM
 820 INPUT "TRY AGAIN? (Y/N) ":YESNO$
 840 IF YESNO$="Y" THEN 320
 860 STOP
1000 REM
1020 REM TELL WHAT TO ENTER
1040 REM
1060 PRINT "YOU CAN PLAY UP TO THREE TONES AT THE SAME
     TIME"
1080 PRINT :"TONES CAN PLAY FOR 1 TO 4250 MILLISECONDS
     (.001 TO 4.25 SECONDS)"
```

Listing 8-2. Cont—Multiple tone example program.

```
1100 PRINT "IF YOU ENTER A POSITIVE     VALUE (1 TO
     4250), THE TONE WAITS UNTIL THE CURRENT TONE ENDS"
1120 PRINT "A NEGATIVE VALUE (-1 TO     -4250) MEANS
     THE TONE STARTS IMMEDIATELY "
1140 PRINT :"YOU CAN GET A FREQUENCY FROM 110 TO 44733
     -- BUT PEOPLE CAN'T HEAR MUCH ABOVE 11000"
1160 PRINT :"YOU CAN GET A VOLUME OF 0 TO 30 (0 IS
     LOUDEST)"
1180 PRINT
1200 RETURN
2000 REM
2020 REM GET THE DURATION
2040 REM
2060 INPUT "ENTER THE TIME (-4250 TO 4250) ":DURATION
2080 REM
2100 REM CHECK FOR CORRECT VALUES
2120 REM
2140 IF DURATION<-4250 THEN 2280
2160 IF DURATION>4250 THEN 2280
2180 REM
2200 REM GOOD VALUE
2220 REM
2240 RETURN
2260 REM
2280 REM BAD VALUE
2300 REM
2320 PRINT
2340 PRINT "TIMES MUST BE BETWEEN -4250 AND 4250"
2360 PRINT
2380 GOTO 2060
3000 REM
3020 REM NOW, GET THE FREQUENCY
3040 REM
3060 INPUT "ENTER FREQUENCY (110 TO 44733) ":FREQUENCY
3080 REM
3100 REM CHECK FOR CORRECT VALUE
3120 REM
3140 IF FREQUENCY<110 THEN 3260
3160 IF FREQUENCY>44733 THEN 3260
3180 REM
3200 REM GOOD VALUE
3220 REM
3240 RETURN
3260 REM BAD VALUE
3280 PRINT
3300 REM
3320 PRINT "FREQUENCY MUST BE BETWEEN 110 AND 44733"
3340 PRINT
3360 GOTO 3060
4000 REM
```

Listing 8-2. Cont—Multiple tone example program.

```
4020 REM GET THE VOLUME
4040 REM
4060 INPUT "ENTER THE VOLUME (0 TO 30) ":VOLUME
4080 REM
4100 REM CHECK FOR GOOD VALUE
4120 REM
4140 IF VOLUME<0 THEN 4260
4160 IF VOLUME>30 THEN 4260
4180 REM
4200 REM GOOD VALUE
4220 REM
4240 RETURN
4260 REM
4280 REM BAD VALUE
4300 REM
4320 PRINT
4340 PRINT "VOLUME MUST BE BETWEEN 0 AND 30":"TRY
     AGAIN"
4360 PRINT
4380 GOTO 4060
4400 END
5000 REM
5020 REM GET ONLY ONE TONE AND VOLUME
5040 REM
5060 GOSUB 3000
5080 FREQ1=FREQUENCY
5100 GOSUB 4000
5120 VOL1=VOLUME
5140 RETURN
6000 REM
6020 REM GET TWO TONES AND VOLUMES
6040 REM
6060 PRINT "FOR THE FIRST TONE"
6080 GOSUB 3000
6100 FREQ1=FREQUENCY
6120 GOSUB 4000
6140 VOL1=VOLUME
6160 PRINT "FOR THE SECOND TONE"
6180 GOSUB 3000
6200 FREQ2=FREQUENCY
6220 GOSUB 4000
6240 VOL2=VOLUME
6260 RETURN
7000 REM
7020 REM GET THREE TONES AND VOLUMES
7040 REM
7060 PRINT "FOR THE FIRST TONE AND VOLUME"
7080 GOSUB 3000
7100 FREQ1=FREQUENCY
7120 GOSUB 4000
7140 VOL1=VOLUME
7160 PRINT "FOR THE SECOND TONE AND VOLUME"
```

Listing 8-2. Cont—Noise generator example program.

```
7180 GOSUB 3000
7200 FREQ2=FREQUENCY
7220 GOSUB 4000
7240 VOL2=VOLUME
7260 PRINT "FOR THE THIRD TONE AND VOLUME"
7280 GOSUB 3000
7300 FREQ3=FREQUENCY
7320 GOSUB 4000
7340 VOL3=VOLUME
7360 RETURN
```

8.3 SPEECH

Not only can you generate music and arcade type sounds on your TI 99/4A, but, with the optional Speech Synthesizer Peripheral, you can make your computer talk!

Many educational cartridges, and more and more game cartridges, are using the Speech Synthesizer. You just plug in the appropriate cartridge and listen to your computer talk. It's that easy. Or, you can write programs to make your computer say what you want it to.

You control speech on your TI 99/4A by attaching the Speech Synthesizer Peripheral and:

- Using one of the cartridges that use the Speech Synthesizer
- Using the Speech Editor cartridge
- Programming in Extended BASIC
- Programming in a lower level language like Assembler

You cannot get your computer to talk to you using TI BASIC unless you have the Speech Editor cartridge.

If you get the Extended BASIC cartridge, it's very easy to get your TI 99/4A to talk. Extended BASIC has two speech-related subprograms:

- SAY makes your Speech Synthesizer "talk." Extended BASIC supplies 373 numbers, words, letters, and phrases.
- SPGET gets the codes needed to produce speech on the Speech Synthesizer. You can make up your own words this way.

Listing 8-3. Noise generator example program.

```
100 REM THIS PROGRAM SHOWS YOU HOW TO MAKE ONE NOISE
120 REM
140 REM  IT ASKS FOR THE TIME, FREQUENCY, AND VOLUME
160 REM AND CHECKS THAT THE VALUES ARE IN THE RIGHT
    RANGES
180 REM
200 REM FIRST, TELL WHAT'S HAPPENING
220 REM
240 CALL CLEAR
260 GOSUB 1000
280 REM
300 REM NOW, GET THE TIME
320 REM
340 GOSUB 2000
360 REM
380 REM NOW, GET THE TYPE OF NOISE
400 GOSUB 3000
420 REM
440 REM FINALLY, GET THE VOLUME
460 GOSUB 4000
480 REM
500 REM AT LAST, PLAY THE TONE
520 REM
540 CALL SOUND(DURATION,NOISE,VOLUME)
560 REM
580 REM  SEE IF ANY MORE
600 REM
620 INPUT "TRY AGAIN? (Y/N) ":YESNO$
640 IF YESNO$="Y" THEN 340
660 STOP
1000 REM
1020 REM TELL WHAT TO ENTER
1040 REM
1060 PRINT :"YOU CAN GET A SOUND FOR 1 TO 4250
     MILLISECONDS"
1080 PRINT "(.001 TO 4.25 SECONDS)"
1100 PRINT
1120 PRINT "IF YOU ENTER A POSITIVE     VALUE (1 TO
     4250), THE SOUND WAITS UNTIL THE CURRENT ONE ENDS"
1140 PRINT :"A NEGATIVE VALUE (-1 TO     -4250) MEANS
     THE SOUND STARTS IMMEDIATELY"
1160 PRINT :"YOU CAN GET ANY OF 8":" BACKGROUND NOISES
     WHERE"
1180 PRINT "-1 TO -4 ARE PERIODIC NOISES AND -5 TO -8
     ARE WHITE NOISES"
1200 PRINT
1220 PRINT "YOU CAN GET A VOLUME OF 0 TO 30 (0 IS
     LOUDEST)"
1240 PRINT
1260 RETURN
```

Listing 8-3. Cont—Noise generator example program.

```
2000 REM
2020 REM GET THE DURATION
2040 REM
2060 INPUT "ENTER THE TIME (-4250 TO 4250) ":DURATION
2080 REM
2100 REM CHECK FOR CORRECT VALUES
2120 REM
2140 IF DURATION<-4250 THEN 2280
2160 IF DURATION>4250 THEN 2280
2180 REM
2200 REM GOOD VALUE
2220 REM
2240 RETURN
2260 REM
2280 REM BAD VALUE
2300 REM
2320 PRINT
2340 PRINT "TIMES MUST BE BETWEEN -4250 AND 4250"
2360 PRINT
2380 GOTO 2060
3000 REM
3020 REM NOW, GET THE TYPE OF NOISE
3040 REM
3060 INPUT "ENTER NOISE TYPE(-1 TO -8) ":NOISE
3080 REM
3100 REM CHECK FOR CORRECT VALUE
3120 REM
3140 IF NOISE<-8 THEN 3260
3160 IF NOISE>-1 THEN 3260
3180 REM
3200 REM GOOD VALUE
3220 REM
3240 RETURN
3260 REM BAD VALUE
3280 PRINT
3300 REM
3320 PRINT :"NOISE VALUE MUST BE BETWEEN -1 AND -8"
3340 PRINT
3360 GOTO 3060
4000 REM
4020 REM GET THE VOLUME
4040 REM
4060 INPUT "ENTER THE VOLUME (0 TO 30) ":VOLUME
4080 REM
4100 REM CHECK FOR GOOD VALUE
4120 REM
4140 IF VOLUME<0 THEN 4260
4160 IF VOLUME>30 THEN 4260
4180 REM
4200 REM GOOD VALUE
4220 REM
4240 RETURN
```

Listing 8-3 Cont—Noise generator example program.

```
4260 REM
4280 REM BAD VALUE
4300 REM
4320 PRINT
4340 PRINT "VOLUME MUST BE BETWEEN 0 AND 30":"TRY
     AGAIN"
4360 PRINT
4380 GOTO 4060
4400 END
```

9

GOOD PROGRAMMING PRACTICES

Some people consider programming to be an art while others consider it a science. In practice, it's a mixture of both. Programming can be easy and fun if you follow some simple, reasonable rules.

In this chapter, we tell you about some programming practices that make it easier to write and maintain programs.

9.1 GOOD PROGRAMMERS

Programming requires a number of skills, many of them very practical. For example, it's a lot easier to enter your programs if you can type since you interact with your computer through its keyboard.

Being a good programmer means that you can write a program that tells the computer *exactly* what you want it to do. And, more importantly, you can *later* make changes to correct problems or make the program even better.

Some good programming rules are:

- Use meaningful variable names.
- Include remarks in your programs.
- Be user friendly.
- Keep track of your programs and data files.
- Back up files.

9.2 MEANINGFUL NAMES

Data that gets manipulated in a program is stored in *variables.* Variables in programs have *names* and you are the one who chooses what every variable is called. Using meaningful variable names makes it a whole lot easier for you to know what is going on in your programs.

You can use up to 15 *characters* for your variable names. Unlike the BASICs on many other home computers, TI BASIC and Extended BASIC have few restrictions on the names that you choose for your variables. You cannot, of course, choose a name for a variable that is assigned as the name of one of TI BASIC's or Extended BASIC's commands, statements, or functions.

It's a lot easier to know what is going on in a program if you name your variables so that they have *some meaning.* Choose *useful names* like:

RATIO, NAME$, PERCENT, YEAR%

instead of *meaningless names* like:

ZZ, X%, Y2, R$

Maintenance is that wonderful process of making a program better. Sometimes it involves adding new features. Other times, it's needed to correct problems. No matter how wonderful you think your program is today, you will probably want to change it sooner or later.

When you are making changes later, you will really appreciate having taken the time to name your variables with some type of reasonable names in the beginning. Anyone who has programmed for any time has seen programs that have statements like this:

```
A = B + C
```

When the programmer wrote that statement, he or she knew what A, B, and C represented. Later, he/she wants to make a change to this wonderful program. What was it A, B, and C stood for? It would be easier to tell what was going on if the statement had been:

```
PAYMENT = PRINCIPAL + INTEREST
```

If you find that you are *running out of space,* you will have to shorten some names. You may want to make a "dictionary" in REMark statements at the beginning of your program telling you what

the shorter names stand for. The statement above could be shorter, and still easily understood:

```
PMT = PRIN + INT
```

9.3 BE USER FRIENDLY

A *user friendly program* tells you what is going on and helps you run the program. Whatever you enter as input is explained. Errors are "trapped" and you are told what happened.

In general, a user friendly program is written so that you don't feel lost when something unusual happens. This takes some care in the programming and you will find yourself becoming more user friendly as you gain experience. You will see what errors occur most often and learn how to trap them.

For example, take a relatively simple program like the one in Listing 9-1. This program is supposed to help you balance your checkbook.

Do you have any idea what the program in Listing 9–1 is doing? What are you entering as data? How do you end it? What is it writing out on the screen? Suppose, instead, that the program looked like the one in Listing 9-2.

The program in Listing 9-2 has a lot more statements which means more typing on your part. But, when you want to make changes, you can read the remarks and see what the program is doing, what type of input you need, and what type of output you should expect. And, when you run the program, it tells you what it expects you to enter.

The program in Listing 9-2 also contains some error checking code.

Listing 9-1. Unfriendly checkbook program example.

```
100 INPUT "BEG VAL ":BEG
110 NEWV=BEG
120 INPUT "POS VAL ":VL
130 IF VL=0 THEN 170
140 NEWV=NEWV+VL
150 PRINT NEWV
160 GO TO 120
170 INPUT "NEG VAL ":VL
180 IF VL=0 THEN 220
190 NEWV=NEWV-VL
200 PRINT NEWV
210 GOTO 170
220 PRINT "DONE"
230 END
```

Listing 9-2. Friendly checkbook program example.

```
100 REM THIS PROGRAM BALANCES A CHECKBOOK
110 REM FIRST, ENTER THE INITIAL BALANCE
120 REM NEXT, ENTER DEPOSITS, ONE AT A TIME, 0 WHEN
    DONE
130 REM NOW, ENTER CHECKS, ONE AT A TIME, 0 WHEN DONE
140 REM BAL = INITIAL + DEPOSITS - CHECKS
150 PRINT "THIS PROGRAM":" BALANCES YOUR CHECKBOOK.": :
160 INPUT "DO YOU WANT INSTRUCTIONS?   (Y/N)":YESNO$
170 IF YESNO$="N" THEN 230
180 PRINT :"1 ENTER THE INITIAL BALANCE."
190 PRINT :"2 ENTER YOUR DEPOSITS,":"  ONE AT A TIME":"
    ENTER A 0 WHEN YOU'RE DONE"
200 PRINT :"3 ENTER YOUR CHECKS,":"  ONE AT A TIME":"
    ENTER A 0 WHEN YOU'RE DONE"
210 PRINT
220 REM  GET THE INFO YOU NEED
230 INPUT "INITIAL BALANCE --> ":INITIAL
240 BALANCE=INITIAL
250 REM NOW, GET THE DEPOSITS
260 PRINT "ENTER YOUR DEPOSITS":"  0 WHEN DONE"
270 INPUT "DEPOSIT --> ":DEPOSIT
280 IF DEPOSIT=0 THEN 390
290 IF DEPOSIT>0 THEN 350
300 PRINT "YOUR DEPOSIT WAS NEGATIVE"
310 INPUT "IS THIS CORRECT? (Y/N)":YESNO$
320 IF YESNO$="Y" THEN 350
330 PRINT "PLEASE RE-ENTER."
340 GOTO 270
350 BALANCE=BALANCE+DEPOSIT
360 PRINT "CURRENT BALANCE: ";BALANCE
370 GOTO 270
380 REM NOW PROCESS THE CHECKS
390 PRINT "ENTER YOUR CHECKS.":"  0 WHEN DONE"
400 INPUT "CHECK AMOUNT --> ":CHECK
410 IF CHECK=0 THEN 520
420 IF CHECK>0 THEN 480
430 PRINT "YOUR CHECK WAS NEGATIVE."
440 INPUT "IS THIS CORRECT? (Y/N)":YESNO$
450 IF YESNO$="Y" THEN 480
460 PRINT "PLEASE RE-ENTER."
470 GOTO 400
480 BALANCE=BALANCE-CHECK
490 PRINT "CURRENT BALANCE: ";BALANCE
500 GOTO 400
510 REM DONE, PRINT FINAL BALANCE
520 PRINT "DONE PROCESSING.":"FINAL BALANCE IS ";
    BALANCE
530 END
```

You cannot enter a negative amount for a deposit or check without it's being caught as a possible error. You know how to tell the program when you are done entering deposits and want to enter checks. And when you are done entering checks and want a final balance.

9.4 INCLUDE REMARKS

Remarks have a place in *every* program. TI BASIC's REM statement lets you put whatever you want into your program. BASIC ignores REM statements when it processes your program but you will not.

You should use REM statements to tell you:

- What the program is supposed to be doing
- What the input is
- What to expect as output
- Any special information
- What the variables mean (especially if there is something special going on)

REMarks take room in your computer's memory. Each character in a remark takes up one character of your computer's memory. For relatively small programs like the ones shown in Listings 9-1 and 9-2, you will not have to worry about running out of memory.

When you start writing larger, more complicated programs, you may have to make your remarks a bit more cryptic. But put them in anyway, cryptic as they may look. You will find it easier to read one strange looking statement than to try to figure out what many lines of code are doing.

Shortened remarks may not be as understandable, but you can still get some idea of what is going on:

```
10 REM BALANCES CHECKBOOK, ENTER INIT BAL, DEPS,
   CHECKS
20 REM 0 FOR END OF DEPS, CHKS
```

9.5 TAKING CARE OF YOUR PROGRAMS AND DATA FILES

You store your programs and data files on cassettes or disks, called *media. Treat your storage media carefully.*

- *Never* put your tapes and disks near a *magnet.*
- Keep the tapes and disks in a *cool, dry* place.
- *Write* gently on disk labels with a *marker.*

- *Never* use a *pencil* to write on a *disk label*—the graphite can get on the disk surface and your computer may not be able to read the disk.
- *Never* use a *ball point pen* to write on a *disk label*—you will make grooves in the recording surface and your computer may not read the data.
- *Always* put some *identification* on a *tape* or *disk.*
- Keep a *log* of what is where.

All tapes and disks look alike on the outside. Only your computer can read what is on them. Make up some naming conventions for your tapes and/or disks.

It's easy to keep a log of what you stored where. Some people use a notebook and add or remove pages as necessary. Others get a binder with dividers and keep really good records. Take your choice. But do something.

Label your tape or disk with a number, a letter, a name, or something. Then, in your notebook, record what is on the tape or disk. Chart 9-1 shows you what information you might want to keep in your log.

Chart 9-1 Data/Program Logbook Example

Tape Number **PGMS01**
Date Written **11/20/82**

File#	Tape Counter	Description
1.	15	a program that balances a checkbook
2.	95	a game program that plays checkers (pretty bad)
3.	159	my check data from 10/82

Disk Number **DSK15**
Date Written **12/25/82**

File Name	Description
CHECKPGM	an updated checkbook program
CHECK1282	check data for 12/82
CHECKERS	a checkers game that I got from a magazine
GRAPHS	a graphics program from a textbook
CHECK1182	check data for 11/82

9.6 BACK UP YOUR FILES

When you have a tape or disk that contains important data or programs, you should make a backup (or copy) of the tape or disk. This

ensures that you will have a copy when you accidentally write over your program or your dog eats your disk.

9.6.1 Backing up Disk Files

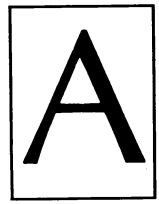
It's very easy to back up a disk file. You don't even have to copy an entire disk, though you can if you want to. You can copy a single file. You might want to create a series of archive disks where you keep your important programs and data.

Just follow the instructions that the Disk Manager Module puts on your screen. You must be careful to remember which disk is being copied from and which one you want to write to.

Once you have copied the programs, update your log for the archive disk.

9.6.2 Backing up Cassette Files

You should also make copies of important cassette files. If you have only one cassette recorder, you load the BASIC program into your computer and write it to the tape. Follow the instructions about writing to a tape. Remember to update your tape log.



WHERE TO LOOK FOR MORE INFORMATION

This appendix tells you *some* places you can find more information for your TI 99/4A. Several sources are specific to the TI 99/4A. Others cover home computers in general and often have articles about the TI 99/4A.

A.1 INTERNATIONAL 99/4 USER'S GROUP

This is a wonderful organization for you to join if you own a TI 99/4A or a TI 99/4. They have a newsletter, software exchange, information and referral service, new product bulletins, and discounts on TI products. Not bad for $12.00 per year. The address is:

International 99/4 User's Group
P.O. Box 67
Bethany, OK 73008

telephone (405) 787-8521

A.2 99'ER MAGAZINE

99'er Magazine bills itself as "Covering the TI 99/4A and other Texas Instruments Personal Computer Systems." And it does a very good job of it. You may be able to find 99'er at a newsstand.

For a $25.00 per year subscription fee, you get a very well written, monthly magazine covering the TI 99/4A in depth. There are articles for beginners and experienced TI 99/4A users, covering entertainment, education, business, professional, and home applications.

There are special "mini-magazines" in every issue covering computer

gaming (both arcade and adventure), LOGO, and portable computing. They also have in-depth reviews of hardware, software, and books.

Every issue has several programs that you can enter. They offer the programs in an issue on cassette for $10.00 to $12.00 if you are a subscriber. This means that you won't have to enter (and debug your errors in) the programs in the magazine.

99'er Magazine's address is:

99'er Magazine
P.O. Box 5537
Eugene, OR 97405

telephone (503) 485-8796

A.3 OTHER HOME COMPUTER MAGAZINES

There are a wealth of magazines written for the home computer user. Some magazines in this category are written for small business or professional users. We are only giving you the names of those magazines which cover home computer use—not small business use and not extremely technical topics.

These magazines can be found at many newsstands. Look at several and see which ones are most comfortable for you to read. You will often find articles on the TI 99/4A in these magazines and several are beginning to have monthly columns dedicated to the TI 99/4A. You will see a lot of ads for equipment in these magazines.

Even if you don't see articles specifically for the TI 99/4A, you will often find BASIC programs in these magazines. You can enter the programs (making whatever changes are needed to translate to TI BASIC) and get essentially free software. In the process, you will learn more about programming and about BASIC.

This is by no means a complete list of home computer magazines. The descriptions are those of the magazines themselves. Look for others at newsstands near you.

CompuKids Magazine

description:	"A computer magazine for beginners"
address:	P.O. Box 874 Sedalia, MO 65301
telephone:	1-800-822-KIDS
subscription:	$17.00 per year
issues:	monthly

COMPUTE!

description:	"The Leading Magazine of Home, Educational, and Recreational Computing"

address: P.O. Box 914
Farmingdale, NY 11737
telephone: 1-800-334-0868
subscription: $20.00
issues: monthly

creative computing
description: "the #1 magazine of computer applications and software"
address: P.O. Box 5214
Boulder, CO 80321
subscription: $19.97
issues: monthly

BYTE
description: "the small systems journal"
address: Subscription Department
P.O. Box 590
Martinsville, NJ 08836
subscription: $21.00
issues: monthly

Popular Computing
address: Subscription Department
P.O. Box 307
Martinsville, NJ 08836
subscription: $11.97
issues: monthly

Personal Computing
address: P.O. Box 2941
Boulder, CO 80321
subscription: $11.97
issues: monthly

Microcomputing
address: Subscription Department
P.O. Box 997
Farmingdale, NY 11737
subscription: $24.97
issues: monthly

B

PROTECTING YOUR INVESTMENT

The TI 99/4A and the Peripheral Expansion System are remarkably sturdy. Texas Instruments is a large industrial company that builds things to industrial quality standards. Part of the reason for the hardiness of the TI 99/4A and its peripherals is that it is intended for use in schools. The care it receives in the school room is likely to be less considerate than the care you are likely to give it in your own home.

There are a few things, though, that you should watch out for.

B.1 STATIC ELECTRICITY

Static electricity is the biggest hazard to your TI 99/4A, indeed, to all computers. It's fast and strikes without warning.

The results can be devastating. Static electricity literally blasts the sensitive chips, burning them out or weakening them so they fail sooner.

The TI 99/4A is most vulnerable to static when you are inserting a *cartridge* into the console.

The Peripheral Expansion System is most vulnerable when you are inserting a peripheral card into its slot.

Note: *Touch something metal to ground yourself before you insert a cartridge or peripheral card into your TI 99/4A!*

B.2 WATER

Your TI 99/4A and its Expansion Box are electrically powered.

Spill a cup of coffee, can of soda, or glass of milk into it and it's all over.

This is a particular hazard when younger children are using your computer (running the wonderful LOGO). A friend of ours has a policy (applied equally

to adults and children) of not allowing liquids on the same table as the computer. If you want a drink, you have to walk away from the TI 99/4A and you cannot bring the drink back with you.

If you spill a liquid into your TI 99/4A while it's turned off, you should be all right. If it's sticky, wipe it off with a damp cloth. *Let it dry before you turn it on!*

B.3 MAGNETS

Tapes and disks are magnetic storage media. That means the information is written on them as little areas of orderly magnetism.

Keep tapes and disks away from all magnetic fields!

Don't store them on your TI 99/4A console, the Expansion Box, or on top of your television set. All these items have power supplies in them that produce strong magnetic fields. They will quickly destroy your stored programs and data.

B.4 HEAT

Semiconductor components are sensitive to heat. They will not fail immediately unless you expose them to extreme temperatures. But, they do weaken with repeated exposure to high temperatures.

Obviously, you should not place your TI 99/4A on a radiator or store it in your oven.

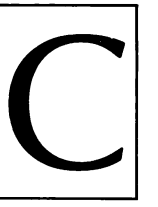
Keep your computer out of direct sunlight! This also applies to peripherals, cartridges, cassette tapes, and disks. Lengthy exposure to direct sunlight is the most common and least appreciated way to shorten the life of your investment.

B.5 EXPANSION BOX TABS

Some of the peripheral cards have a tab on them that sticks out the back of the Expansion Box. You could, if you tried, break this off. If you succeed, you can throw away the card.

These tabs are often connected by wires to other pieces of equipment (printers, modems, disk drives). If you pull hard enough on these wires, you can break off the tab. *So be careful when you move this external equipment!* The best thing to do is to disconnect the wires from the tabs before you move any equipment.

You can also trip over a wire and do the same damage. *Keep wires off the floor and out of the way.*



SOME BASIC PROGRAMS

In this Appendix, we give you the source code for some BASIC programs. You can enter the code and have working programs that show you something about the sound and graphics capabilities of your TI 99/4A.

The Tic-Tac-Toe program actually plays the game. If you don't want to use hearts and bows for the markers, then design a different marker and put it into the program. Make changes to reflect your own likes.

The Sprite Editor program makes it easy for you to design sprites and new characters. We used the Sprite Editor to design the heart and bow markers you see in the Tic-Tac-Toe program. You must have the Extended BASIC cartridge to run the Sprite Editor.

The Twinkle program plays the familiar childhood song for you. This shows you how to enter notes and times to play a song. The Hot Cross Buns program plays another song for you.

C.1 TIC-TAC-TOE PROGRAM

The Tic-Tac-Toe program (Listing C-1) is a two-player game. Player one gets the heart marker. Player two gets the bow.

The program can tell whether there is a winner or if the game is a draw. You get a chance to play again.

C.2 SPRITE EDITOR PROGRAM

This is an interactive character and sprite editing program that requires Extended BASIC (Listing C-2).

This program includes the following features:

- Full control over screen, character, and sprite color
- Single (8 by 8) or double (16 by 16) sized sprites
- Normal or magnified sprites
- Full editing control

Listing C-1. Tic-tac-toe program.

```
 100 REM  PLAY TIC TAC TOE
 120 REM  USE GRAPHICS CHARACTERS FOR PIECES
 140 REM  THIS IS A TWO PLAYER GAME
 160 REM
 180 CALL CLEAR
 200 REM ** INITIALIZE GAME
 220 DIM ROWPAT$(11)
 240 REM P$ ARRAY SHOWS PLAYER TOKENS
 260 P$(1)=CHR$(128)
 280 P$(2)=CHR$(136)
 300 GOSUB 4000
 320 REM
 340 REM ** INITIALIZE BOARD
 360 FOR I=1 TO 9
 380 BOARD(I)=0
 400 NEXT I
 420 GOSUB 3000
 440 REM
 460 REM GET FIRST TWO MOVES
 480 REM NO ONE CAN WIN IN TWO MOVES
 500 FOR K=1 TO 2
 520 PLAYER=1
 540 GOSUB 1000
 560 PLAYER=2
 580 GOSUB 1000
 600 NEXT K
 620 REM CHECK FOR WIN ON REST OF MOVES
 640 REM
 660 PLAYER=2
 680 FOR M=5 TO 9
 700 PLAYER=3-PLAYER
 720 GOSUB 1000
 740 GOSUB 2000
 760 IF WIN<>0 THEN 840
 780 NEXT M
 800 PRINT "THE GAME IS A DRAW"
 820 GOTO 860
 840 PRINT :"GOOD PLAYING "&P$(PLAYER)
 860 INPUT "WANT TO PLAY AGAIN (Y/N) ":Y$
 880 IF Y$="Y" THEN 340
 900 STOP
1000 REM GET MOVE
1020 INPUT "YOUR MOVE(1-9) "&P$(PLAYER):MOVE
1040 IF (MOVE<1)+(MOVE>9)THEN 1140
1060 IF BOARD(MOVE)<>0 THEN 1220
1080 BOARD(MOVE)=(2*PLAYER)-3
1100 GOSUB 3000
1120 RETURN
1140 PRINT "MOVES MUST BE 1-9"
1160 FOR I=1 TO 250
1180 NEXT I
```

Listing C-1. Cont—Tic-tac-toe program.

```
1200 GOTO 1020
1220 PRINT "SQUARE ALREADY PLAYED!!"
1240 FOR I=1 TO 250
1260 NEXT I
1280 GOTO 1020
2000 REM  CHECK FOR WIN
2020 REM IF BOARD=PLAYER THEN PLAYER WINS
2040 WIN=PLAYER
2060 IF ABS(BOARD(1)+BOARD(5)+BOARD(9))=3 THEN 2260
2080 IF ABS(BOARD(4)+BOARD(5)+BOARD(6))=3 THEN 2260
2100 IF ABS(BOARD(2)+BOARD(5)+BOARD(8))=3 THEN 2260
2120 IF ABS(BOARD(3)+BOARD(5)+BOARD(7))=3 THEN 2260
2140 IF ABS(BOARD(1)+BOARD(2)+BOARD(3))=3 THEN 2260
2160 IF ABS(BOARD(1)+BOARD(4)+BOARD(7))=3 THEN 2260
2180 IF ABS(BOARD(3)+BOARD(6)+BOARD(9))=3 THEN 2260
2200 IF ABS(BOARD(7)+BOARD(8)+BOARD(9))=3 THEN 2260
2220 REM NO WINNER
2240 WIN=0
2260 RETURN
3000 REM WRITE BOARD
3020 CALL CLEAR
3040 FOR ROW=1 TO 11
3060 PRINT "   ";ROWPAT$(ROW)
3080 NEXT ROW
3100 PRINT : :
3120 REM FILL IN BOARD
3140 FOR I=1 TO 9
3160 IF BOARD(I)=0 THEN 3360
3180 ROW=4*(INT(I/3.5)+1)+8
3200 FOR COL=I TO 1 STEP -3
3220 IF COL<4 THEN 3260
3240 NEXT COL
3260 COL=7+((COL-1)*4)
3280 IF BOARD(I)=1 THEN 3340
3300 CALL HCHAR(ROW,COL,128)
3320 GOTO 3360
3340 CALL HCHAR(ROW,COL,136)
3360 NEXT I
3380 RETURN
4000 REM INITIALIZE
4020 CALL SCREEN(14)
4040 FOR I=1 TO 14
4060 CALL COLOR(I,16,1)
4080 NEXT I
4100 CALL COLOR(3,2,16)
4120 CALL COLOR(4,2,16)
4140 CALL COLOR(13,7,16)
4160 CALL COLOR(14,5,16)
4180 CALL CHAR(59,"0000000000000000")
4200 CALL CHAR(60,"1818181818181818")
4220 CALL CHAR(61,"181818FFFF181818")
```

Listing C-1. Cont—Tic-tac-toe program.

```
4240 CALL CHAR(62,"000000FFFF000000")
4260 CALL CHAR(128,"66FFFFFF7E3C1818")
4280 CALL CHAR(136,"81C3E7FFFFE7C381")
4300 ROWPAT$(1)="1"&CHR$(59)&CHR$(59)&CHR$(60)&"2"&CHR$
     (59)&CHR$(59)&CHR$(60)&"3 "&CHR$(59)&CHR$(59)
4320 ROWPAT$(5)="4"&CHR$(59)&CHR$(59)&CHR$(60)&"5"&CHR$
     (59)&CHR$(59)&CHR$(60)&"6 "&CHR$(59)&CHR$(59)
4340 ROWPAT$(9)="7"&CHR$(59)&CHR$(59)&CHR$(60)&"8"&CHR$
     (59)&CHR$(59)&CHR$(60)&"9 "&CHR$(59)&CHR$(59)
4360 ROWA$=CHR$(59)&CHR$(59)&CHR$(59)&CHR$(60)&CHR$(59)
     &CHR$(59)&CHR$(59)&CHR$(60)&CHR$(59)&CHR$(59)&CHR$
     (59)
4380 ROWB$=CHR$(62)&CHR$(62)&CHR$(62)&CHR$(61)&CHR$(62)
     &CHR$(62)&CHR$(62)&CHR$(61)&CHR$(62)&CHR$(62)&CHR$
     (62)
4400 ROWPAT$(2)=ROWA$
4420 ROWPAT$(3)=ROWA$
4440 ROWPAT$(4)=ROWB$
4460 ROWPAT$(6)=ROWA$
4480 ROWPAT$(7)=ROWA$
4500 ROWPAT$(8)=ROWB$
4520 ROWPAT$(10)=ROWA$
4540 ROWPAT$(11)=ROWA$
4560 RETURN
```

The Sprite Editor puts a grid on the screen in which you place "." and "X." Where there is an "X," the sprite is colored. Where there is a ".", the screen color shows through.

The Sprite Editor commands are as follows:

- C lets you change the colors of the screen, characters, and sprite.
- U causes the sprite image in the upper right corner of the screen to be *updated* to show the changes you have made to the sprite editing grid.
- BACK (FCTN 8) ends editing of the current sprite.
- . represents an "off" dot in the sprite image. These will be the same color as the screen.
- X represents an "on" dot in the sprite image. These will be the color you specified for the sprite.
- A is a toggle that turns the *automove* feature on/off during sprite editing. Automove is the automatic movement of the cursor after you press the "." or "X" keys. Movement is in the direction of the last entered direction key.
- M is also a toggle that alters the magnification of the sprite image.
- **FCTN** SEDX are the standard FCTN key shifted direction *arrows on* the

Listing C-2. Sprite editor program.

```
 100 OPTION BASE 1 !SET ARRAYS TO START AT CELL ONE
 120 ON WARNING NEXT
 140 REM
 160 REM THIS PROGRAM CREATES A SPRITE IMAGE
     INTERACTIVELY
 180 REM
 200 REM ITS MAIN ROUTINE IS IN LINES 100-1999 (1000 IS
     THE INITIALIZATION ROUTIN E)
 220 REM
 240 REM THE SPRITE DISPLAY ROUTINE IS FROM LINE 2000-
     2999
 260 REM
 280 REM LINES 3000-3999 CONVERT THE SPRITE TABLE TO A
     HEXADECIMAL STRING FOR THE  CAL TO THE CHAR ROUTINE
 300 REM
 320 REM KEYBOARD INPUT IS IN LINES 4000-4999
 340 REM
 360 REM THE 5000 SERIES ROUTINE SETS THE COLORS
 380 REM
 400 REM THE 6000 SERIES ROUTINE PRINTS THE SPRITE
     DEFINITION STRING TO THE SCREEN SO THAT YOU CAN
     INCLUDE IT IN YOUR PROGRAMS
 420 REM
 440 REM
 460 REM
 480 REM SPAR IS THE SPRITE DEFINITION ARRAY
 500 REM
 520 DIM SPAR(16,16)
 540 CALL CLEAR
 560 GOSUB 1000 !INITIALIZE SYSTEM
 580 CALL CLEAR
 600 CALL DISPRITE(SPAR(,),CHCOLOR,SPCOLOR,SCRCOLOR,
     SPSIZE,SPDEF$)
 620 PRINT "DO YOU WANT TO SEE"
 640 INPUT "    THE SPRITE DATA?":Y$
 660 IF Y$="N" THEN 720
 680 IF Y$<>"Y" THEN 620
 700 CALL DUMPSPR(SPDEF$)
 720 INPUT "CREATE ANOTHER SPRITE?":Y$
 740 IF Y$="N" THEN STOP
 760 CALL DELSPRITE(#1)
 780 IF Y$="Y" THEN 540
 800 GOTO 720
 820 REM INITIALIZE SPRITE VARIABLES
1000 INPUT "USE DOUBLE SIZE SPRITES (16x16)?":Y$
1020 SPSIZE=1
1040 IF Y$="Y" THEN SPSIZE=3 :: GOTO 1100
1060 IF Y$="N" THEN 1100
1080 GOTO 1000
```

Listing C-2. Cont—Sprite editor program.

```
1100 INPUT "DO YOU WANT MAGNIFIED SPRITES?":Y$
1120 IF Y$="Y" THEN SPSIZE=SPSIZE+1 :: GOTO 1180
1140 IF Y$="N" THEN 1180
1160 GOTO 1100
1180 CALL PATTERN(#1,128)
1200 CALL MAGNIFY(SPSIZE)
1220 CALL CLEAR
1240 FOR I=1 TO 16
1260 FOR J=1 TO 16 :: SPAR(I,J)=0 :: NEXT J
1280 NEXT I
1300 SCRCOLOR=14 :: CALL SCREEN(SCRCOLOR)
1320 SPCOLOR=2
1340 CHCOLOR=16
1360 FOR C=0 TO 12 :: CALL COLOR(C,CHCOLOR,1):: NEXT C
1380 CALL SETCOLOR(CHCOLOR,SPCOLOR,SCRCOLOR)
1400 CALL CLEAR
1420 CALL CHAR(128,RPT$("0",64))
1440 CALL SPRITE(#1,128,SPCOLOR,10,176)
1460 RETURN
2000 SUB DISPRITE(SPAR(,),CHCOLOR,SPCOLOR,SCRCOLOR,
     SPSIZE,SPDEF$)! DISPLAY SPRIT E ROUTINE
2020 C$(1)="." :: C$(2)="X"
2040 MAXI=16 !MAX SPRITE SIZE
2060 IF SPSIZE<3 THEN MAXI=8
2080 CALL CLEAR
2100 FOR I=1 TO MAXI
2120 FOR J=1 TO MAXI :: DISPLAY AT(I+2,J+1):C$(SPAR(I,
     J)+1):: NEXT J
2140 NEXT I
2160 DISPLAY AT(7,19)SIZE(11):"COMMANDS" :: DISPLAY AT
     (8,19)SIZE(11):"C=COLOR" :: DISPLAY AT(9,19)SIZE
     (11):"U=UPDATE"
2180 DISPLAY AT(10,19)SIZE(11):"BACK=DONE" :: DISPLAY
     AT(11,19)SIZE(11):".=DOT OFF" :: DISPLAY AT(12,19)
     SIZE(11):"X=DOT ON" 2200 DISPLAY AT(13,19)SIZE (11)
     :"A=AUTOMOVE" :: DISPLAY AT(14,19)SIZE(11): "M=CHG
     SIZE"
2220 DISPLAY AT(16,19)SIZE(11):"FCTN SEDX=" :: DISPLAY
     AT(17,19)SIZE(11):" DIRECTION"
2240 CROW=3 :: CCOL=2
2260 CALL BITOHEX(SPAR(,),SPSIZE,SPDEF$)
2280 CALL RDKEY(SPAR(,),RTVAL,CROW,CCOL,SPSIZE,CHCOLOR,
     SPCOLOR,SCRCOLOR)
2300 IF RTVAL=99 THEN SUBEXIT
2320 GOTO 2260
2340 SUBEND
3000 SUB BITOHEX(SPAR(,),SPSIZE,SPDEF$)!CREATE THE
     SPRITE DEF STRING FROM SPAR
3020 SPDEF$=""
```

Listing C-2. Cont—Sprite editor program.

```
3040 STRTROW=1 :: ENDROW=8
3060 STRTCOL=1
3080 GOSUB 3300
3100 IF SPSIZE<3 THEN CALL CHAR(128,SPDEF$):: SUBEXIT
3120 STRTROW=9 :: ENDROW=16
3140 GOSUB 3300
3160 STRTROW=1 :: ENDROW=8
3180 STRTCOL=9
3200 GOSUB 3300
3220 STRTROW=9 :: ENDROW=16
3240 GOSUB 3300
3260 CALL CHAR(128,SPDEF$)
3280 SUBEXIT
3300 FOR I=STRTROW TO ENDROW
3320 HEXVAL=0
3340 FOR J=0 TO 3
3360 IF SPAR(I,STRTCOL+J)=1 THEN HEXVAL=HEXVAL+2^ABS
     (J-3)
3380 NEXT J
3400 SPDEF$=SPDEF$&SEG$("0123456789ABCDEF",HEXVAL+1,1)
3420 HEXVAL=0
3440 FOR J=4 TO 7
3460 IF SPAR(I,STRTCOL+J)=1 THEN HEXVAL=HEXVAL+2^ABS
     (J-7)
3480 NEXT J
3500 SPDEF$=SPDEF$&SEG$("0123456789ABCDEF",HEXVAL+1,1)
3520 NEXT I
3540 RETURN
3560 SUBEND
4000 SUB RDKEY(SPAR(,),RTVAL,CROW,CCOL,SPSIZE,CHCOLOR,
     SPCOLOR,SCRCOLOR)!READ THE KEYS AND ACT ON
     COMMANDS
4020 MAXI=17
4040 IF SPSIZE<3 THEN MAXI=9
4060 RTVAL=0 !INIT RETURN VALUE
4080 D=9 !INIT AUTOMOVE DIRECTION
4100 CALL KEY(0,K,S):: DISPLAY AT(CROW,CCOL)SIZE(1):
     CHR$(30)
4120 IF SPAR(CROW-2,CCOL-1)=0 THEN DISPLAY AT(CROW,
     CCOL)SIZE(1):"." ELSE DISPLAY AT(CROW,CCOL)SIZE(1)
     :"X"
4140 IF K=-1 THEN 4100
4160 DISPLAY AT(22,1)SIZE(1)BEEP:""
4180 IF K=ASC(".")THEN SPAR(CROW-2,CCOL-1)=0 :: DISPLAY
     AT(CROW,CCOL)SIZE(1):"." :: GOTO 4360
4200 IF K=ASC("X")THEN SPAR(CROW-2,CCOL-1)=1 :: DISPLAY
     AT(CROW,CCOL)SIZE(1):"X":: GOTO 4360
4220 IF K=ASC("U")THEN SUBEXIT
4240 IF K=ASC("A")THEN AUTOM=1-AUTOM :: GOTO 4100
```

Listing C-2. Cont—Sprite editor program.

```
4260 IF K=ASC("C")THEN CALL SETCOLOR(CHCOLOR,SPCOLOR,
     SCRCOLOR):: GOTO 4100
4280 IF K=15 THEN RTVAL=99 :: SUBEXIT
4300 IF K=ASC("M")THEN GOSUB 4700 :: GOTO 4100
4320 IF (K<8)+(K>11)THEN 4620
4340 D=K :: GOTO 4380
4360 ON AUTOM+1 GOTO 4100,4380
4380 IF D<>8 THEN 4440
4400 CCOL=CCOL-1 :: IF CCOL<2 THEN CCOL=2
4420 GOTO 4100
4440 IF D<>9 THEN 4500
4460 CCOL=CCOL+1 :: IF CCOL>MAXI THEN CCOL=MAXI
4480 GOTO 4100
4500 IF D<>11 THEN 4560
4520 CROW=CROW-1 :: IF CROW<3 THEN CROW=3
4540 GOTO 4100
4560 IF D<>10 THEN 4620
4580 CROW=CROW+1 :: IF CROW>MAXI+1 THEN CROW=MAXI+1
4600 GOTO 4100
4620 DISPLAY AT(22,1)SIZE(1)BEEP:""
4640 DISPLAY AT(22,1)SIZE(31)BEEP:"INVALID COMMAND!"
4660 FOR S=1 TO 220 :: NEXT S
4680 DISPLAY AT(22,1)SIZE(31):"" :: GOTO 4100
4700 IF SPSIZE<3 THEN SPSIZE=3-SPSIZE ELSE SPSIZE=7
     -SPSIZE
4720 CALL MAGNIFY(SPSIZE)
4740 RETURN
4760 SUBEND
5000 SUB SETCOLOR(CHCOLOR,SPCOLOR,SCRCOLOR)
5020 DISPLAY AT(23,1):"CHANGE SCREEN COLOR?"
5040 ACCEPT AT(23,22)VALIDATE("YN"):Y$
5060 IF Y$="N" THEN 5100
5080 C=SCRCOLOR :: GOSUB 5320 :: CALL SCREEN(C)::
     SCRCOLOR=C
5100 DISPLAY AT(23,1)SIZE(31):"CHANGE SPRITE COLOR?"
5120 ACCEPT AT(23,22)VALIDATE("YN"):Y$
5140 IF Y$="N" THEN DISPLAY AT(23,1)SIZE(31):"" :: GOTO
     5180
5160 C=SPCOLOR :: GOSUB 5320 :: CALL COLOR(#1,C)::
     SPCOLOR=C
5180 DISPLAY AT(23,1)SIZE(31):"CHANGE CHARACTER COLOR?"
5200 ACCEPT AT(23,25)VALIDATE("YN"):Y$
5220 IF Y$="N" THEN DISPLAY AT(23,1)SIZE(31):"" ::
     SUBEXIT
5240 C=CHCOLOR :: GOSUB 5320
5260 CHCOLOR=C
5280 FOR C=0 TO 12 :: CALL COLOR(C,CHCOLOR,1):: NEXT C
5300 SUBEXIT
5320 DISPLAY AT(22,1)SIZE(31):"CURRENT COLOR IS";C
5340 DISPLAY AT(23,1)SIZE(31):"ENTER NEW COLOR."
```

Listing C-2 Cont—Sprite editor program.

```
5360 ACCEPT AT(23,18)VALIDATE(DIGIT):C
5380 DISPLAY AT(22,1)SIZE(31):"" :: DISPLAY AT(23,1)
     SIZE(31):""
5400 IF C<1 THEN GOSUB 5460 :: GOTO 5340
5420 IF C>16 THEN GOSUB 5460 :: GOTO 5340
5440 RETURN
5460 DISPLAY AT(23,1)SIZE(31)BEEP:"COLORS RANGE FROM 1
     TO 16." :: FOR C=1 TO 300 :: NEXT C :: RETURN
5480 SUBEND
6000 SUB DUMPSPR(SPDEF$)!PRINT THE SPRITE DATA
6020 PRINT : :"DATA FOR SPRITE "
6040 PRINT "IN TWO CHARACTER PIECES ->"
6060 PRINT
6080 COUNT=1
6100 FOR I=1 TO LEN(SPDEF$)STEP 2
6120 PRINT SEG$(SPDEF$,I,2)&" ";
6140 COUNT=COUNT+1 :: IF COUNT>8 THEN COUNT=1 :: PRINT
6160 NEXT I
6180 PRINT
6200 SUBEXIT
6220 SUBEND
6240 END
```

TI 99/4A keyboard. You move the cursor using the arrow keys, just like editing a BASIC program.

The Sprite Editor program is an example of the power of Extended BASIC. Much of the quality of interaction is possible only because of features included in Extended BASIC.

The program is structured as a series of *named subprograms*. The main routine runs from line 100 to 1460. The routine that begins at line 1000 is an initialization routine.

The DISPRITE (display sprite) named subprogram begins at line 2000. This routine sets up the initial sprite editor display, then CALLs the routines that interact with the user and update the sprite image.

The BITOHEX (binary to hexadecimal) named subprogram that begins at line 3000 converts the zeros and ones in the array variable SPAR (sprite array) into the hexadecimal character string (SPDEF$) needed by the CHAR subprogram.

The GOSUB routine that begins at line 3300 may be a bit confusing. This routine looks at two 4-element series in array SPAR, interpreting each 1 value as an "on" bit in the sprite image and each 0 value as an "off" bit in the image. The two 4-element series give the two hexadecimal digits that define the pattern of a single row in one byte of the sprite image.

The hexadecimal digit is determined by computing a value for each nibble (4-bit half byte). This value can range from 0 (all "bits" off) to 15 (all "bits" on). This corresponds to the hexadecimal digits 0 through F.

To compute this value, you need to know that bit positions in a byte are numbered from 7 (on the left) to 0 (on the right). It turns out, the decimal value of an "on" bit in any position is equal to 2 raised to the position of the bit. That is, an "on" bit in position 3 represents a decimal value of 2^3 (2 raised to the third power) or 8.

Once we have added together the values of all of the "on" bits, represented by ones in SPAR, we have an *index* into a string (statement 3400) of hexadecimal digits. (We have to add one to the HEXVAL because strings don't start at zero.)

Got all that? Good!

The RDKEY subprogram at line 4000 takes keyboard input from the TI 99/4A console and does as you command. Notice that you hear a beep each time a command is accepted and two beeps when you enter an invalid command. This is to let you know what the program is doing.

The SETCOLOR subprogram at line 5000 does just that. It lets you set the color of the screen, characters, and sprite image.

The DUMPSPR (dump sprite) subprogram at line 6000 prints the sprite definition string to the screen. This lets you use the string in other programs. For example, we got the definition strings for the bow and heart used in the Tic-Tac-Toe program from the Sprite Editor.

C.2.1 Room for Improvement

There are always ways to improve programs. Look at the Sprite Editor to see if you can find some.

To give you an idea what to look for, we will give you some suggested improvements that you can do yourself.

1. The SETCOLOR subprogram should give you a list of the colors and their codes. It can do this only in the lower part of part of the screen so as not to interfere with the sprite grid.
2. The DUMPSPR routine should be able to save the sprite data to a disk or tape.
3. You should be able to provide an initial sprite definition string so that you don't have to start from scratch each time. (This involves writing a hexadecimal to bit routine.)

C.3 TWINKLE PROGRAM

Twinkle (Listing C-3) plays the familiar song. This program is easy to experiment with.

Try making the song play over and over by using the appropriate GO TO at the end (GOTO 500).

Listing C-3. Twinkle program.

```
100 REM TWINKLE
110 DIM NOTE(50),TIME(50)
120 CALL CLEAR
130 REM
140 REM BELOW ARE THE NOTES
150 REM
160 DATA 262,262,392,392,440,440,392
170 DATA 349,349,330,330,294,294,262
180 DATA 392,392,349,349,330,330,294
190 DATA 392,392,349,349,330,330,294
200 DATA 262,262,392,392,440,440,392
210 DATA 349,349,330,330,294,294,262
220 REM
230 REM HERE ARE THE TIMES
240 REM
250 DATA 250,250,250,250,250,250,500
260 DATA 250,250,250,250,250,250,500
270 DATA 250,250,250,250,250,250,500
280 DATA 250,250,250,250,250,250,500
290 DATA 250,250,250,250,250,250,500
300 DATA 250,250,250,250,250,250,500
310 REM
320 REM LOAD THE NOTES
330 REM
340 FOR I=1 TO 42
350 READ NOTE(I)
360 NOTE(I)=NOTE(I)
370 NEXT I
380 REM
390 REM LOAD THE TIME FOR THE NOTES
400 REM
410 FOR I=1 TO 42
420 READ TIME(I)
430 TIME(I)=TIME(I)
440 NEXT I
450 REM
460 REM
470 REM PLAY THE SONG
480 REM
490 REM
500 FOR I=1 TO 42
510 CALL SOUND(TIME(I),NOTE(I),0)
520 NEXT I
530 END
```

Change the pitch by multiplying the NOTE in statement 360. If you multiply by a number less than one (.7), you will get each note played lower. If you multiply by a number higher than one (2 or 3.6), you will get the song played higher. Remember, if you multiply by a number that has values after its decimal point (like 3.5), you will change the notes, as well as the pitch for the note.

If you use a factor in the TIME in statement 430, you will change the length of time each note is played.

C.4 HOT CROSS BUNS PROGRAM

The Hot Cross Buns program (Listing C-4) plays the first 17 notes of the tune with that name. Use this base program to learn more about how your TI 99/4A plays music.

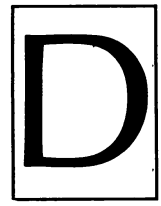
If you adjust the factor in statement 250 to a higher number, you will get the tune played with higher notes. If you adjust it lower, you will get lower notes. If you use a factor like 3.5, you will get the tune played with other than the correct notes. You will be surprised how different notes sound with the same timing.

Listing C-4. Hot cross buns program.

```
100 REM PLAYS HOT CROSS BUNS
110 REM
120 REM   PLAY ONLY 17 NOTES
130 REM
140 DIM NOTE(17),TIME(17)
150 DATA 165,147,131,165,147,131,131
160 DATA 131,131,131,147,147,147,147
170 DATA 165,147,131
180 DATA 250,250,500,250,250,500,125
190 DATA 125,125,125,125,125,125,125
200 DATA 250,250,500
210 REM ** READ THE NOTES **
220 REM ADJUST THE PITCH BY A FACTOR
230 FOR I=1 TO 17
240 READ NOTE(I)
250 NOTE(I)=NOTE(I)*2
260 NEXT I
270 REM **.READ TIME FOR EACH NOTE **
280 REM MAKE THE TONES PLAY LONGER OR SHORTER TIMES
290 FOR I=1 TO 17
300 READ TIME(I)
310 TIME(I)=TIME(I)*2
320 NEXT I
330 FOR I=1 TO 17
340 CALL SOUND(TIME(I),NOTE(I),0)
350 NEXT I
360 END
```

If you adjust the factor in statement 310, the tones will play for longer (larger factor) or shorter (smaller factor) times. Adjusting this factor to something like 6.33 will make the music sound really strange.

This is only the beginning of a song. Try the base program with tones and timing for your favorite song. Just be sure to adjust the NOTE and TIME arrays to the appropriate values (one for each note/time pair) and put the data in the correct DATA statements. DATA statements 150 to 170 are the tones. Statements 180 to 200 are the times, one for each note. If you use more than 17 notes, put the extra notes after statement 170 and before 180. Extra times go after statement 200. And, don't forget to adjust the FOR loops to reflect the new number of notes and times.



GLOSSARY OF COMPUTER TERMS

ADDRESS—A unique number assigned to each memory location (byte). The TI 99/4A generates addresses between 0 and 65,535—a 64K range.

ASCII—The *A*merican *S*tandard Code for *I*nformation *I*nterchange is a set of one-byte codes used by many computer systems to represent letters, digits, punctuation marks, and special control codes.

ASSEMBLER—The *program* that converts Assembly Language source statements into machine language for direct execution by the TMS9900 microprocessor. Also used to refer to Assembly Language, as in "programming in Assembler." Compare to a compiler and an interpreter.

ASSEMBLY LANGUAGE—A low-level programming language in which each statement translates into a *single* machine language instruction. This is the most powerful language available for the TI 99/4A and also the most difficult to work in.

BACKUP—A secure second copy of important information. You should make backups of all tape and disk resident data and programs that you cannot, or don't want to, recreate. The backup can be another tape or disk, or, less desirable, a paper listing of the program or data.

BASIC—*B*eginners *A*ll *P*urpose *S*ymbolic *I*nstruction Code is the most widely used programming language on microcomputers today. TI BASIC conforms to the American National Standard.

BINARY—The base 2 number system computers use to count. The binary system has only two digits (0 and 1) that correspond to the "on" and "off" bits in a computer memory.

BIT—A single *B*inary Dig*it*. A bit can be either "on" (value 1) or "off" (value 0) and is the fundamental unit of computer memory. For most purposes, bits are usually arranged in groups of eight to form a byte.

BYTE—A unit of memory sufficient to store one character. A byte contains eight bits and is the unit most commonly used to measure memory size. The TI 99/4A console contains 16,384 bytes of user memory.

CARTRIDGE—Contains a program and plugs into the slot on the top right side of the TI 99/4A console. Cartridges contain programs stored in ROM and are a convenient way to buy software for the TI 99/4A.

CHARACTER—A letter, number, space, or punctuation mark. Characters in the TI 99/4A are stored as ASCII codes in one byte of memory.

CHIP—A small piece of silicon containing finely etched circuitry. This is a slang expression for an integrated circuit. There is a microprocessor chip, a video display chip, and many memory chips in the TI 99/4A.

COMPILER—A program that takes the source language statements of a higher-level language (higher than Assembly Language) and converts them into machine language for direct execution by the TMS9900 microprocessor. Each statement of a higher-level language, like FORTH, is translated into more than one machine language instruction. Compare this to an assembler and an interpreter.

CPU—The Central *P*rocessing *U*nit is the brain of the computer. The 16-bit 9900 microprocessor is the CPU in the TI 99/4A.

DATA—Values that are manipulated by programs. There are two major types of data in BASIC, string data and numeric data.

DISK—A round magnetic storage medium. A disk (sometimes called a diskette) works in a disk drive to store programs and data. Single sided disks store 90,000 characters of data; double sided disks store 180,000 characters.

DISK DRIVE—A fast mass storage peripheral that provides immediate access to programs and data. A disk drive is more than 30 times faster than a cassette recorder.

EXPANSION BOX—The first component of the TI Peripheral Expansion System. It contains a power supply for the Peripheral Cards and internal disk drive that fit into it.

EXPANSION CARDS—Circuit boards that fit into the Expansion Box. There are expansion cards to add memory, access disks, and communicate with other computers.

GRAPHICS—Drawing charts, graphs, or pictures on the screen. The TI 99/4A supports sophisticated color graphics.

HARDWARE—The physical components of your TI 99/4A. This includes the console, cassette cables, joysticks, disk drive—anything you can put your hands on.

HERTZ—A frequency measurement equal to one cycle per second.

HEXADECIMAL—A base 16 numbering system commonly used in computer systems. Hexadecimal is convenient to use because two hexadecimal digits (0 to 9 and A to F) can represent any one byte value.

INTERPRETER—A program that interprets and executes the statements in

another program. BASIC is an interpreted language: the BASIC interpreter reads a BASIC statement, analyzes it, and does what it says. Compare this to a compiler or assembler.

MACHINE LANGUAGE—The *native language* of the TMS9900 microprocessor. Human beings do not generally program in machine language. The closest approximation to machine language that is useful to people is Assembly Language, where each statement corresponds to a single machine language instruction.

MODEM—A *Mo*dulator-*Dem*odulator used in computer to computer telephone communications. To use a modem on your TI 99/4A, you need an RS-232 Interface card in the Expansion Box.

MONITOR—A high quality television set. You don't need a monitor to use your TI 99/4A.

OBJECT CODE—The output from the Assembler or from a compiler. Object code is a complete set of machine language instructions ready—or nearly ready—to be loaded and directly executed by the TMS9900 microprocessor.

OVERLAY—The plastic strips that fit into the slot above the number keys on the TI 99/4A console. The words on the overlay indicate the action of FCTN and CTRL shifted number keys.

PACKAGE—A program or programs that perform some task. Software is sold in units called packages.

PERIPHERAL—A piece of hardware external to the TI 99/4A console.

PROGRAM—A set of detailed instructions that make the computer perform a desired task. You can write programs in many languages on the TI 99/4A, but BASIC is the most common language.

RAM—*R*andom *A*ccess *M*emory is the memory that is available for your use. You can write to and read from RAM, but RAM forgets what you put in it after you turn off the TI 99/4A. The TI 99/4A console contains 16K of RAM and the system can expand to accommodate 52K of RAM.

READ—The moving of data from an external source into Random Access Memory. The TI 99/4A reads data from such places as the keyboard, cassette tapes, Wafertapes, the RS-232 interface, disk drives, and a telephone modem.

ROM—*R*ead *O*nly *M*emory has a program permanently stored in it. You cannot use ROM to store your programs. The TI 99/4A console contains 26K of ROM most of which contains the TI BASIC interpreter. The cartridges you plug into the console all contain varying amounts of ROM.

RS-232—An industry-standard hardware and software communications protocol. This is a set of rules defining the way computers talk to modems, printers, or other computers.

SOFTWARE—The instructions that make the computer do what you want. There is a great deal of software available for the TI 99/4A.

TMS5200—The speech synthesis chip in the Solid State Speech Synthesis

Peripheral. This chip allows your TI 99/4A to speak to you. It's very useful in early learning software and in games.

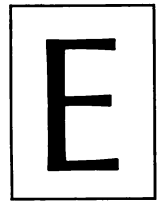
TMS9900—The 16-bit microprocessor that serves as the central processing unit in the TI 99/4A.

TMS9918A—The video display controller in the TI 99/4A. This sophisticated video chip maintains the image you see on the screen. It allows you to change screen colors, draw fine color pictures, and create independent moving objects called sprites.

TMS9919—The sound generator chip in the TI 99/4A. This chip can create three tones and eight noises.

TRAP—The code you write to detect and report an error, sometimes *before* it occurs. The BASIC interpreter traps many errors before they cause your program to go crazy. In other languages, especially Assembly Language, you must write your own error trapping code.

WRITE—The moving of data from Random Access Memory to an external device. The TI 99/4A can write data to such places as the screen, a cassette tape, a Wafertape, disks, the RS-232 interface, a printer, and the telephone modem.



TI BASIC AND EXTENDED BASIC COMMANDS, STATEMENTS, AND FUNCTIONS

Notation:

words in **BOLDFACE AND CAPITALS** are *keywords* that you enter exactly as they appear
num-exp means any *numeric expression,* like A+B, 42.34
num-var means any *numeric variable,* like X, INTEREST
str-exp means any *string expression,* like A$, "XYZ", FIRST$&MIDDLE$&LAST$
str-var means any *string variable,* like Y$, NAME$
variable means *any* variable, string or numeric, like YES$, PAYMENT
brackets ([ ]) mean whatever is between the [ ] is *optional* and you don't have to use it if you don't want to
ellipsis (,. . .) means that the preceding thing can be repeated as many times as you want
device-filename means the device for cassette files (like CS1) and, for disk files, the name of the file on the disk as well as the device name (like DSK1.MYFILE)

ABS(*num-exp*)

Type: Function
Description: **ABS** returns the absolute value (positive value) of *num-exp*.
Example: **PRINT ABS(-199.34)**

This prints the value 199.34

ACCEPT [[**AT**(*row,col*)] [**VALIDATE** (*data*[,. . .])] [**BEEP**] [**ERASE ALL**] [**SIZE**(*expression*)] :] *variable*

Type: Extended BASIC Statement or Command
Description: **ACCEPT** positions the cursor at *row* and *col* and waits for data to be entered from the keyboard. It works somewhat like an **INPUT** statement.

Example:

```
 50 CALL CLEAR
100 PRINT "ENTER A Y OR N"
200 ACCEPT AT (22,10) VALIDATE ("YN") : Y$
300 IF Y$ = "Y" THEN PRINT "ANSWER IS YES" :: GOTO 100
400 PRINT "ANSWER IS NO" :: GOTO 100
```

This **ACCEPT** statement positions the cursor at row 22, column 10 and reads an answer into the string variable Y$. The **VALIDATE** keyword means that the answer can be only "Y" or "N". If the answer is Y, "ANSWER IS YES" is printed. If the answer is N, "ANSWER IS NO" is printed. Then the screen is cleared and the program starts over. (Press FCTN **CLEAR** to end it.)

ASC(*str-exp*)

Type: Function
Description: **ASC** returns the ASCII value of the first character of *str-exp*.
Example: **PRINT ASC("ABCDEF")**

This prints 65.

ATN(*num-exp*)

Type: Function
Description: **ATN** returns the arctangent of *num-exp*. An arctangent is a trigonometric function.
Example: **PRINT ATN(123.45)**

This prints 1.562696058

BREAK [*line-num-list*]

Type: Command or Statement
Description: **BREAK** makes the program stop until you enter a **CONTINUE** command. If you use **BREAK** with a list of line numbers (*line-num-list*), the program will stop when it

reaches any of the lines included in *line-num-list*. **UNBREAK** makes all **BREAK** commands inactive.

Example:

```
100 BREAK 150,200
125 PRINT "HELLO THERE"
150 PRINT "HERE I AM AT STMT 150"
175 PRINT "BACK AGAIN"
200 PRINT "AND NOW I'M AT STMT 200"
225 PRINT "AND I'M RUNNING AGAIN"
250 PRINT "NOW I'M DONE"
275 END
```

This program stops at statements 150 and 200. You make it continue by entering **CON** or **CONTINUE.**

BYE

Type: Command
Description: **BYE** closes all open files and leaves BASIC.
Example: **BYE**

Close any open files, leave BASIC, and return to the main screen.

CALL *subprog-name* [(*parameters*)]

Type: Extended BASIC Statement
Description: In Extended BASIC, you can write subprograms with names (*subprog-name*) and call them with a **CALL** statement. (You cannot write named subprograms in TI BASIC.) You pass values to your named subprogram through *parameters*. The **CALL** statement transfers control to subprogram *subprog-name* and passes the *parameters*.
Example: 100 CALL MYSUB (123.45,"XYZ")

This **CALL** statement transfers control to the subprogram with the name MYSUB and passes it the values 123.45 and "XYZ". You need **SUB** and **SUBEND** statements to create a named subprogram.

CALL CHAR(*ASCII-code,pattern-string*)
or
only in Extended BASIC
CALL CHAR(*ASCII-code,pattern-string*[,. . .])

Type: Command or Statement
Description: **CHAR** redefines the pattern or image associated with the character represented by *ASCII-code*. The new pattern is given in *pattern-string*.

Example: **CALL CHAR(128,"66FFFFFF7E3C1818")**

This redefines the image associated with ASCII code 128 to the dot pattern which prints a heart (hexademical string 66FFFFFF7E3C1818).

CALL CHARPAT(*ASCII-code,str-var*[,. . .])

Type: Extended BASIC Statement or Command
Description: **CHARPAT** gets the pattern code (a string of hexadecimal digits that represent the image) for the character associated with *ASCII-code* and puts the pattern into *str-var*.
Example: **CALL CHARPAT(90,PAT$)**

This puts the pattern code for character Z (ASCII code 90) into the string variable PAT$ as a series of hexadecimal digits (007C04081020407C).

CALL CHARSET

Type: Extended BASIC Statement or Command
Description: **CHARSET** restores the standard character patterns and colors for the standard character set (ASCII codes 32 through 95).
Example: **CALL CHARSET**

This resets the standard character set (upper and lower case letters, numbers, and special symbols).

CHR$(*num-exp*)

Type: Function
Description: **CHR$** gives a one character string whose ASCII value is *num-exp*.
Example: **PRINT CHR$(90)**

This prints the letter "Z" on your screen.

CALL CLEAR

Type: Statement or Command
Description: **CLEAR** "clears the screen" to all blank characters.
Example:

```
100 CALL CLEAR
200 FOR I = 1 to 23
300 PRINT "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ"
400 NEXT I
500 INPUT "PRESS ENTER TO CONTINUE"  :  Y$
600 GOTO 100
```

This program first clears the screen and then fills the screen with Zs. When you press ENTER, it clears the screen again and refills the screen with Zs. Press FCTN CLEAR to stop it.

CLOSE # *file-number* [**:DELETE**]

Type: Statement or Command

Description: **CLOSE** closes file *file-number* and, if you say **DELETE,** removes the file from the device. You can only delete files from a disk. If you say **DELETE** with a cassette file, the file is closed but not removed from the tape.

Example:

```
100  CLOSE # 12
200  CLOSE # 15  :  DELETE
```

This program closes file #12. Then it closes file #15 and deletes it.

CALL COINC(*#sprite-num1,#sprite-num2,tolerance,num-var*)

or

CALL COINC(**ALL,***num-var*)

or

CALL COINC(*#sprite-num,dot-row,dot-col,tolerance,num-var*)

Type: Extended BASIC Statement or Command

Description: **COINC** detects when a sprite is at a specific position on the screen (*dot-row,dot-col*) or when two sprites coincide (are at the same *dot-row* and *dot-col*). *tolerance* tells how many dots apart the sprite and position or other sprite can be for coincidence. *num-var* is 0 for no coincidence and −1 for coincidence.

Example:

```
100  CALL COINC (#1,#2,10,COLLIDE)
200  IF COLLIDE = 0 THEN 500
300  PRINT "SPRITES COLLIDED"
400  GOTO 999
500  PRINT "SPRITES DIDN'T COLLIDE"
999  STOP
```

This tells you if sprite 1 and sprite 2 are within 10 dots of each other.

CALL COLOR (*char-set, foreground-color, background-color*)

or

only in Extended BASIC

CALL COLOR(*char-set,foreground-color,background-color*[,. . .])

or

CALL COLOR(*#sprite-num,foreground-color*[,. . .])

Type: Statement or Command
Description: **COLOR** sets the foreground and background colors for the characters in *char-set.*
In Extended BASIC, **COLOR** also sets the color for specified sprites.
Example: **CALL COLOR(5,1,5)**

This **COLOR** command sets the colors for the characters A through G as transparent on a light blue background. The other letters and numbers are not changed.

CONTINUE
or
CON

Type: Command
Description: The **CONTINUE** or **CON** command resumes executing a program after the program has executed a **BREAK** statement/command, had an error occur, or you press FCTN CLEAR. You cannot CONTINUE a program after you have edited it.
Example: **CON**

Use the **CON** command with the program described under **BREAK.**

COS(*num-exp*)

Type: Function
Description: **COS** returns the trigonometric cosine of the value *num-exp* where *num-exp* is expressed in radians.
Example: **PRINT COS(2.34)**

This prints −.6955633265

DATA *data-list*

Type: Statement
Description: A **DATA** statement stores numeric and/or string data in a program. You use a **READ** statement to put the values in the **DATA** statement *data-list* into variables in your program.
Example:

```
100  DATA 1,2,3,ABC,DEF
200  READ A, B
300  PRINT A
400  READ C
500  READ ST1$, ST2$
```

```
600  PRINT B, C, ST2$, ST1$
700  END
```

This program stores three numeric values (1, 2, 3) and two string values (ABC, DEF). The **READ** statements at 200, 400, and 500 put the values into variables.

DEF *fctn-name*[(*parameter*)] = *expression*

Type: Statement

Description: **DEF** defines a numeric or string function with the name *fctn-name*. You can pass a value to the function with *parameter*. The value returned by the function is defined by *expression*. You use a function instead of rewriting *expression* every place you need it. This saves space in your program and makes it easy to change *expression*.

Example:

```
100  DEF ACUBED (A) =A*A*A
200  A = 100
300  PRINT ACUBED(A)
400  X = 15
500  PRINT ACUBED(X)
600  PRINT ACUBED(2)
700  END
```

This shows how you define a function that is the cube of the parameter. Statement 300 prints 1,000,000. Statement 500 prints 3,375. Statement 600 prints 8.

DELETE *"device-filename"*

Type: Command or Statement

Description: **DELETE** deletes file *filename* from *device*. You cannot use DELETE with a cassette.

Example: DELETE "DSK1.OLDFILE"

This deletes file "OLDFILE" from disk DSK1.

CALL DELSPRITE (ALL)

or

CALL DELSPRITE (*#sprite-num*[,. . .])

Type: Extended BASIC Statement or Command

Description: **DELSPRITE** removes sprites from the screen. You can remove sprites by number (*sprite-num*) or remove all sprites.

Example: **CALL DELSPRITE (#4,#9)**

This removes sprites 4 and 9 from the screen. Any other sprites stay on the screen.

DIM *array-name*(*bound1*[,*bound2*,. . .]) [,. . .]

Type: Statement or Command
Description: **DIM** allocates (dimensions) space for the arrays. Each array (*array-name*) gets the number of elements (*bound1*. . .) allocated. You can have up to 3 dimensions in TI BASIC and up to 7 dimensions in Extended BASIC.
Example: **DIM A(100), B(20,20), ST$(5,5,5)**

This shows how you allocate 101 elements (0 to 100) for array A; 441 (21 × 21) for B; and 216 (6 × 6 × 6) for ST$. Remember: arrays start with element 0 unless you say **OPTION BASE 1.**

DISPLAY [*list*]

Type: Statement or Command
Description: **DISPLAY** writes data in *list* to the screen like a **PRINT** statement.
Example: **DISPLAY "HERE I AM"**

This writes HERE I AM on your screen.

DISPLAY [[**AT** (*row,col*)] [**BEEP**] [**ERASE ALL**]
[**SIZE**(*num-exp*)] :] *variable-list*

Type: Extended BASIC Statement or Command
Description: This form of the **DISPLAY** statement writes at *row,col* on your screen. You can make your computer **BEEP** or **ERASE** the screen before printing.
Example: **DISPLAY AT(15,10) BEEP ERASE ALL : "HELLO THERE!"**

This erases the screen, beeps, and prints HELLO THERE! at row 15 and column 10.

DISPLAY [*options*] **USING** *str-exp* [: *variable-list*]
or
DISPLAY [*options*] **USING** *line-num* [: *variable-list*]

Type: Extended BASIC Statement or Command
Description: This form of the **DISPLAY** statement writes data to the screen using a format. You can use an **IMAGE** statement for the format (*line-num*) or put the format (*str-exp*) in the **DISPLAY** statement *options* are any options for the **DISPLAY AT** statement.
Example:

```
100 DOLLARS = 234.5
200 DISPLAY AT(10,10) BEEP USING $###.## :  DOLLARS
```

This beeps and prints $234.50 at row 10, column 10.

CALL DISTANCE (*#sprite-num1,#sprite-num2,num-var*)
or
CALL DISTANCE (*#sprite-num1,dot-row,dot-col,num-var*)

Type: Extended BASIC Statement or Command
Description: **DISTANCE** tells you the square of the distance between two sprites (*sprite-num1* and *sprite-num2*) or between a sprite and a position on the screen (*dot-row,dot-col*). The squared distance is placed in variable *num-var*.
Example: **CALL DISTANCE (#1, #9, DIST)**

This puts the square of the distance between sprites 1 and 9 in the variable DIST.

EDIT[*line-num*]

Type: Command
Description: **EDIT** allows you to edit line *line-num*.
Example: **EDIT 200**

This displays line 200 of the BASIC program in memory and allows you to make changes to it.

END

Type: Command or Statement
Description: **END** ends your program and stops its execution. You can only use one **END** statement per program, as the last line in your program.
Example:

```
100 PRINT "HI!"
200 END
```

This prints HI! on your screen and then stops.

EOF [(*file-num*)]

Type: Function
Description: The **EOF** function tells you if you are at the end of the file *file-num*. If you are not, you get a 0. If you are, you get a 1. If there is no more room on the disk, you get a −1. The EOF function does not work with cassette files.
Example: `100 IF EOF(2) = 0 THEN 900`

This branches to statement 900 if there is still room left in file number 2.

CALL ERR (*err-code,err-type* [*,severity,line-num*])

Type: Extended BASIC Statement or Command
Description: **ERR** tells you the most recent error code (*err-code*) and error type (*err-type*). You can also get the line number where the error occurred. *Severity* is always 9.
Example: **CALL ERR(CODE,TYPE)**

This returns the most recent error code in variable CODE and the error type in variable TYPE.

EXP(*num-exp*)

Type: Function
Description: **EXP** returns the exponential value of *num-exp*. This is e^x where e =2.718281828.
Example: **PRINT EXP(1.23)**

This prints 3.421229536

FOR *control* = *init-val* **TO** *end-val* [**STEP** *incr*]

Type: Statement or Command
Description: **FOR-TO-STEP** repeatedly executes the statements between the **FOR** and **NEXT** statements. The control variable *control* starts at *init-val*. Each time the associated **NEXT** statement is executed, *control* is incremented by *incr* or by one (if you don't use **STEP**). If *control* is less than *end-val*, the statements between **FOR** and **NEXT** are executed again.

Example:

```
100  FOR I=0 to 100 STEP 10
200  PRINT I
300  NEXT I
400  END
```

This prints by tens from 0 to 100.

CALL GCHAR (*row,col,num-var*)

Type: Statement or Command
Description: **GCHAR** puts the ASCII code for the character at position *row* and *col* into *num-var*.
Example: **CALL GCHAR(10,15,CHVAR)**

This puts the ASCII code for the character at row 10, column 15 into the variable CHVAR.

GOSUB *line-num*
or
GO SUB *line-num*

Type: Statement
Description: **GOSUB** transfers control to the subprogram at line *line-num.*

Example:

```
100  GOSUB 1000
200  PRINT "AT STATEMENT 200"
300  PRINT "DONE"
400  STOP
1000 PRINT "AT STATEMENT 1000"
1100 PRINT "RETURNING"
1200 RETURN
1300 END
```

This shows how you use a subprogram (the statements between 1000 and 1200). The **GOSUB** statement at 100 "calls" the subprogram at line 1000.

GOTO *line-num*
or
GO TO *line-num*

Type: Statement
Description: **GOTO** unconditionally transfers control to the statement at line *line-num.*

Example:

```
100 PRINT "AT STATEMENT 100"
200 GOTO 500
300 PRINT "AT STATEMENT 300"
400 STOP
500 PRINT "AT STATEMENT 500"
600 GOTO 100
```

The **GOTO** statement at line 200 transfers control to statement 500. Statement 300 will never be executed unless you have a **GOTO 300** statement somewhere else in your program (like **550 GOTO 300**).

CALL HCHAR (*row,col,ASCII-code* [*,repetitions*])

Type: Statement or Command
Definition: **HCHAR** writes the character with value *ASCII-code* at row *row* and column *col.* If you use a value for *repetitions,* you will get that many characters written across the screen beginning at *row,col.*
Example: **CALL HCHAR(10,15,63,10)**

This starts at row 10, column 15 and writes 10 ? (question marks) across the screen.

IF *condition* **THEN** *line-num1* [**ELSE** *line-num2*]
or
only in Extended BASIC
IF *condition* **THEN** *clause1* [**ELSE** *clause2*]

Type: Statement
Description: **IF-THEN-ELSE** determines if *condition* is true or false and transfers control to line number *line-num1* when the expression is true or line number *line-num2* when the expression is false.
In Extended BASIC *only,* you can use a statement or group of statements instead of a line number after **THEN** or **ELSE.**
Example: **IF A > B THEN 100 ELSE 200**

If the value of A is greater than the value of B, statement 100 is executed next. Otherwise, statement 200 is executed next.

IMAGE *format-string*

Type: Extended BASIC Statement
Description: **IMAGE** specifies a format in *format-string* for a **PRINT USING** or **DISPLAY USING** statement.
Example:

```
100  IMAGE $###.##
200  DOLLS = 123.77
300  PRINT USING 100  :  DOLLS
400  END
```

This prints the value $123.77.

CALL INIT

Type: Extended BASIC Statement or Command
Description: **INIT** prepares the system to load and run an Assembly Language subprogram. You use **INIT** with **LINK, LOAD,** and **PEEK** statements.
Example: **CALL INIT**

This sets up the Extended BASIC program to load and run an Assembly Language subprogram.

INPUT [*prompt*:] *variable-list*

Type: Statement
Description: **INPUT** writes a message (*prompt*) to the screen and reads data into the variables in *variable-list.*
Example: **INPUT "ENTER YOUR NAME " : NAME$**

This prints the message ENTER YOUR NAME on the screen and puts whatever you enter into NAME$.

INPUT# *file-num* : *variable-list*

Type: Statement
Description: **INPUT#** reads data from file *file-num* into the variables in *variable-list.*
Example: **100 INPUT# 5 : NAME$, AMOUNT**

This reads from file 5 and puts the first value into the string variable NAME$ and the second into AMOUNT.

INT(*num-exp*)

Type: Function
Description: **INT** returns the integer value that is the largest integer less than or equal to *num-exp.*
Example: **PRINT INT(123.456+234.777)**

This prints 358.

CALL JOYST(*key-unit,x-return,y-return*)

Type: Statement or Command
Description: **JOYST** tells you the position of either joystick (*key-unit* = 1 or 2).
Example: **CALL JOYST(1,XPOS,YPOS)**

This tells you the position of joystick one.

CALL KEY (*key-unit,return-var,status-var*)

Type: Statement or Command
Description: **KEY** returns the ASCII value of the key pressed in the *key-unit.*
Example: **CALL KEY(2,KEYV,STAT)**

This looks at the key pressed from *key-unit* 2 (the right side of the keyboard). The key value is placed in KEYV. The status in STAT tells whether any key was pressed or whether the same key was pressed as the last time KEY was called.

LEN (*str-exp*)

Type: Function
Description: **LEN** returns the number of characters in *str-exp.*
Example: **PRINT LEN("ABCDEFGHIJKLMNOP")**

This prints 16, the number of letters in ABCDEFGHIJKLMNOP.

[**LET**] *variable* = *expression*

Type: Statement or Command
Description: **LET** assigns the value of *expression* to *variable*. The keyword **LET** is an optional part of an assignment statement.
Example: **LET ABC=1234**

This places the value 1234 into variable ABC. You could also write this statement as:

ABC=1234

CALL LINK("*subprog-name*"[,*arguments*])

Type: Extended BASIC Statement or Command
Description: **LINK** passes control to the Assembly Language subprogram *subprog-name* and passes it the arguments you specify.
Example: **CALL LINK("MYASM")**

This calls the Assembly Language subprogram MYASM.

LINPUT [[*#file-num*] [,**REC** *rec-num*] :] *str-var*
or
LINPUT [*prompt* :] *str-var*

Type: Extended BASIC Statement
Description: **LINPUT** writes the optional *prompt* message and reads the data into *str-var*. The data must end with a carriage return. *#file-num* means that **LINPUT** reads from file *file-num* instead of the keyboard. The **REC** option allows you to read records randomly (by number) from a disk file.
Example: **LINPUT# 7 : DATALINE$**

This reads a line from file 7 into DATALINE$.

LIST [[*start-line*] [– [*end-line*]]]

Type: Command
Description: **LIST** lists the lines from the BASIC program in memory, beginning with line *start-line* and ending with line *end-line*. If you don't use *start-line*, the first line in the program is used. If you don't use *end-line*, the last line in the program is used.
Example: **LIST 100-200**

This lists lines 100 through 200 from the BASIC program in memory.

CALL LOAD (*address,byte1*[,*byte2, . . .*]
[,'''',*address,byte*[,*byte, . . .*]])
or
CALL LOAD (''*device-filename*''[, . . .])

Type: Extended BASIC Statement or Command
Description: **LOAD** is used with **INIT**, **LINK**, and **PEEK**. **LOAD** brings an Assembly Language program into the memory expansion memory. You can also use **LOAD** to enter values (*byte1, byte2,* etc.) into specific locations in the expansion memory beginning at location *address.*
Example: **CALL LOAD (12000,15,16,17)**

This loads three bytes into the expansion memory at locations 12000 through 12002. Location 12000 becomes 15; location 12001 becomes 16; location 12002 becomes 17.

CALL LOCATE(*#sprite-num, dot-row,dot-col*[,. . .])

Type: Extended BASIC Statement or Command
Description: **LOCATE** moves sprite *sprite-num* to location *dot-row* and *dot-col.*
Example: **CALL LOCATE(#1,100,100,#4,150,150)**

This moves sprite 1 to 100,100 and sprite 2 to 150,150. *dot-row* can be 1 through 192. *dot-col* can be 1 through 256.

LOG(*num-exp*)

Type: Function
Description: **LOG** returns the natural logarithm of the expression *num-exp*
Example: **Y = LOG(125)**

This puts the value 4.828343737 into variable Y.

CALL MAGNIFY (*magnification-factor*)

Type: Extended BASIC Statement or Command
Description: **MAGNIFY** changes the size and magnification of all sprites. *magnification-factor* is 1 for single sized sprites; 2 for magnified single sized sprites; 3 for double sized sprites; and 4 for double sized magnified sprites.
Example: **CALL MAGNIFY(2)**

This makes all sprites single sized and magnified.

MAX (num-exp1,num-exp2)

Type: Extended BASIC Function

Description: **MAX** returns the larger of the two numeric expressions *num-exp1* and *num-exp2*.
Example: **BIGGER=MAX(A,B)**

This puts the larger of the two variables A and B into the variable BIGGER.

MERGE *device.filename*

Type: Extended BASIC Command
Description: **MERGE** brings lines from file *filename* on device *device* and merges these lines (puts them in line number order) into the program lines already in the computer's memory.
Example: **MERGE DSK1.SUB20**

This reads the BASIC program DSK1.SUB20 and puts its lines into the program already in memory.

MIN *(num-exp1,num-exp2)*

Type: Extended BASIC Function
Description: **MIN** returns the smaller of the two numeric expressions *num-exp1* and *num-exp2*.
Example: **SMALLER = MIN(X*2, X*Y*6.5)**

This evaluates the expressions X*2 and Y*Y*6.5 and puts the smaller value into variable SMALLER.

CALL MOTION (*#sprite-num,row-vel,col-vel*[, . . .])

Type: Extended BASIC Statement or Command
Description: **MOTION** changes the motion of the sprite(s).
Example: **CALL MOTION (#2,20,40)**

This moves sprite 2 down and to the right at a slow speed.

NEW

Type: Command
Description: **NEW** clears the computer's memory, clears the screen, and gets ready to accept a new program.
Example: **NEW**

The BASIC program in memory is erased and you can enter a new program.

NEXT [*control*]

Type: Statement or Command

Description: **NEXT** goes with a **FOR-TO-STEP** statement and increments the *control* value in a **FOR** statement.
Example: see the **FOR-TO-STEP** statement.

NUMBER [*start-line*[,*increment*]]
or
NUM [*start-line*[,*increment*]]

Type: Command
Description: **NUMBER** or **NUM** generates sequenced line numbers for entering a BASIC program. *start-line* is 100 if you don't say otherwise. *increment* is 10 if you don't specify a value.
Example: **NUM 150,25**

This prints line numbers for entering a BASIC program, starting with line number 150 and incrementing the line numbers by 25.

OLD *device*[.*program-name*]

Type: Command
Description: **OLD** loads the BASIC program from device *device*. If you are loading a program from a disk, you need to use *program-name*.
Example: **OLD CS1**

This loads a program from the cassette.

OLD DSK1.MYPROG

This loads the program in file MYPROG from disk DSK1.

ON BREAK STOP
or
ON BREAK NEXT

Type: Extended BASIC Statement
Description: **ON BREAK** determines the action taken when a breakpoint occurs in a program. Breakpoints may be caused by program action (**BREAK** statement) or by keyboard action (FCTN CLEAR).
Example: **ON BREAK STOP**

This makes the program stop when you press (FCTN CLEAR).

ON ERROR STOP
or
ON ERROR *line-num*

Type: Extended BASIC Statement
Description: **ON ERROR** determines the action taken when an error condition occurs.
Example: **ON ERROR 9900**

When an error occurs, the program branches to line 9900 (where you wrote error handling code).

ON *num-exp* **GOSUB** *line-num* [, . . .]

Type: Statement
Description **ON-GOSUB** transfers control to the subprogram at the line number corresponding to *num-exp.*
Example: **ON X GOSUB 100,200,5200**

This executes the subprogram at line 100 when X = 1, at 200 when X = 2, and at 5200 when X = 3.

On *num-exp* **GOTO** *line-num* [, . . .]

Type: Statement
Description: **ON-GOTO** unconditionally transfers control to the line number in the position corresponding to *num-exp.*
Example: **ON (X*Y)/2 GOTO 1000,9500,4000**

This evaluates the expression (X*Y)/2 and branches to statement 1000 when the value is 1, to statement 9500 when the value is 2, and to 4000 when the value is 3.

ON WARNING PRINT
or
ON WARNING STOP
or
ON WARNING NEXT

Type: Extended BASIC Statement
Description: **ON WARNING** determines the action taken when a warning condition occurs.
Example: **ON WARNING STOP**

The program stops when a warning condition occurs.

OPEN *#file-num:device-filename* [*,file-org*][*file-type*] [*,open-mode*] [*,record-type*]

Type: Statement or Command
Description: **OPEN** associates the specified file with number *file-num* and enables the program to read/write data from/to the file.
Example: **OPEN #4:DSK1.DATA,FIXED,OUTPUT**

This opens disk file DATA, file number 4, as a fixed length output file.

OPTION BASE 0
or
OPTION BASE 1

Type: Statement
Description: **OPTION BASE** sets the lowest subscript for all arrays to zero or one.
Example: **OPTION BASE 1**

This makes all arrays start with element 1 (otherwise, they start at 0).

CALL PATTERN *(#sprite-num,value*[, . . .])

Type: Extended BASIC Statement or Command
Description: **PATTERN** changes the pattern for the sprite(s).
Example: **CALL PATTERN (#1,SPRPAT1$)**

This makes sprite 1 have the pattern defined by string SPRPAT1$.

CALL PEEK *(address,num-var-list)*

Type: Extended BASIC Statement or Command
Description: **PEEK** places the value(s) in the *address*(es) into the variables in *num-var-list.*
Example: **CALL PEEK (8000,VAR1,VAR2)**

This places the value of address 8000 into VAR1 and the value of 8001 into VAR2.

PI

Type: Extended BASIC Function
Description: **PI** returns the value of pi as 3.14159265359.
Example: **AREA = PI * RAD * RAD**

This calculates the area as pi times the radius squared.

POS *(string1,string2,num-exp)*

Type: Function

Description: **POS** returns the position of the first occurrence of *string2* in *string1* starting at character *num-exp* in *string1*.
Example: **WHERE = POS ("ABCDEFG","D",1)**

This puts the value of 4 in WHERE.

CALL POSITION (*#sprite-num,dot-row,dot-col* [, . . .])

Type: Extended BASIC Statement or Command
Description: **POSITION** returns the positions of the sprite(s).
Example: **CALL POSITION (#5,YPOS,XPOS)**

This returns the position of sprite 5. Its row value is in YPOS and its column value is in XPOS.

PRINT [*#file-num* [,**REC** *rec-num*] :] [*list*]

Type: Statement or Command
Description: **PRINT** writes the data in *list* to the screen or to the file *file-num*.
Example: **PRINT " 5 + 6 = ";5+6**

This writes " 5 + 6 = 11" on your screen.

PRINT [*#file-num* [,**REC** *num-exp*]] **USING** *str-exp:list*
or
PRINT [*#file-num* [,**REC** *num-exp*]] **USING** *line-num:list*

Type: Extended BASIC Statement or Command
Description: **PRINT USING** writes the data in *list* to the screen or the file *file-num* using the format specified in the **USING** clause. (See the **DISPLAY USING** statement.)
Example: **PRINT #9 USING $###,### : 123456**

This writes the value $123,456 to file number 9.

RANDOMIZE [*num-exp*]

Type: Statement or Command
Description: **RANDOMIZE** resets the random number generator.
Example: **RANDOMIZE**

READ *variable-list*

Type: Statement or Command
Description: **READ** assigns values from a **DATA** statement to the variables in *variable-list*. (See the **DATA** statement.)
Example:

```
100 DATA 1,2.238,MY STRING DATA
```

```
200 READ XVAL, YVAL
300 READ STRDATA$
400 PRINT YVAL, XVAL, STRDATA$
500 END
```

This reads the value 1 into variable XVAL, 2.238 into YVAL, and MY STRING DATA into STRDATA$ and then prints the values.

REC [*file-num*]

Type: Extended BASIC Function
Description: **REC** returns the current record position in file *file-num*.
Example: **PRINT REC(6)**

This prints the number of the record in file 6 that will be processed with the next **PRINT, INPUT,** or **LINPUT** statement.

REM *string*

Type: Statement or Command
Description: **REM** lets you include remarks (nonexecutable statements) in a BASIC program. You use REMarks to tell what your program is doing, how it operates, and what your variables are.

Example:

```
200 REM THIS PROGRAM WAS WRITTEN 2/22/82
210 REM BY JOHN SMITH
220 REM THE PROGRAM CALCULATES MONTHLY PAYMENTS
```

This includes nonprogram information in a program.

RESEQUENCE [*initial*] [,*increment*]
or
RES [*initial*] [,*increment*]

Type: Command
Description: **RES** or **RESEQUENCE** renumbers the lines in the BASIC program currently in memory. Line numbers start at *initial* (or 100) and increase by *increment* (or 10).
Example: **RES 500,50**

This renumbers the lines in the BASIC program currently in memory, beginning the new line numbers at 500 and increasing them by 50.

RESTORE [*line-num*]
or
RESTORE #*file-num* [,**REC** *rec-num*]

Type: Statement or Command

Description: **RESTORE** resets the line number for **DATA** statement used in the next **READ** statement. **RESTORE#** resets the current record number for file *file-num*.
Example: **RESTORE 200**

This sets line 200 as the next **DATA** statement used by a **READ** statement.

RETURN

Type: Statement
Description: **RETURN** transfers program control from a subprogram to the statement following the **GOSUB** or **ON GOSUB** statement that called the subprogram.
Example:

```
100  GOSUB 500
150  PRINT "BACK FROM SUBPROGRAM AT 500"
200  STOP
500  PRINT "HERE I AM IN THE SUBPROGRAM"
550  PRINT "I'M GOING TO RETURN NOW"
600  RETURN
650  END
```

The **GOSUB** at line 100 "calls" the subprogram starting at line 500. The **RETURN** at line 600 ends the subprogram and returns control to statement 150.

RETURN [*line-num*]
or
RETURN [NEXT]

Type: Extended BASIC Statement
Description: This form of the **RETURN** statement is used with the **ON ERROR** statement. It transfers program control after an error occurrs.
Example: **9875 RETURN NEXT**

When an error occurs and is handled with an **ON ERROR** statement, this **RETURN** statement transfers control to the statement after the statement causing the error.

RND

Type: Function
Description: **RND** returns a random number between 0 and 1.
Example: **NEWVAL = OLDVAL * RND**

This uses the **RND** function to get a number between 0 and 1 and then multiplies the value by OLDVAL.

RPT$ *(str-exp,num-exp)*

Type: Extended BASIC Function
Description: **RPT$** repeats the characters in *str-exp, num-exp* times.
Example: **STRDATA$=RPT$ ("ABCDEF",4)**

This sets the string STRDATA$ to ABCDEFABCDEFABCDEFABCDEF.

RUN [*line-num*]
or
only in Extended BASIC
RUN ["*device-filename*"]

Type: Command or Statement
Description: **RUN** loads and executes the program in *device-filename* or begins executing the BASIC program currently in memory, starting at *line-num*. If you just use **RUN**, the program in memory begins executing at its first line.
Example: **RUN "DSK1.NEWPROG"**

This loads the BASIC program called NEWPROG from disk DSK1 and begins executing it.

SAVE *device-filename*
or
only in Extended BASIC
SAVE *device-filename* [,**PROTECTED**]
or
SAVE *device-filename* [,**MERGE**]

Type: Command
Description: **SAVE** writes the BASIC program currently in memory to *device-filename*. If you are using Extended BASIC, you can protect the program or make it a merged file (which you use with a **MERGE** command).
Example: **SAVE CS1**

This writes the BASIC program currently in memory to the cassette recorder CS1.

CALL SAY *(word-string*[,*direct-string*] [, . . .])

Type: Extended BASIC Statement or Command
Description: **SAY** makes the speech synthesizer say the word *word-string* or a *direct-string* returned by **SPGET**.
Example: **CALL SAY ("HELLO")**

This makes your speech synthesizer say the word HELLO.

CALL SCREEN *(color-code)*

Type: Statement or Command
Description: **SCREEN** changes the screen color to that given by *color-code.*
Example: **CALL SCREEN(5)**

This changes the screen color to dark blue.

SEG$ *(str-exp,position,length)*

Type: Function
Description: **SEG$** returns a substring of *str-exp.* The returned string is *length* characters long and begins at character *position* in *str-exp.*
Example: **SUBST$=SEG$ ("HELLO HI THERE",7,8)**

This puts the string HI THERE into SUBST$.

SGN *(num-exp)*

Type: Function
Description: **SGN** returns a one if *num-exp* is positive, a zero if *num-exp* is zero, and a minus one if *num-exp* is negative.

Example:

```
100 MYDATA = -1 * 400
200 PRINT MYDATA, SGN(MYDATA)
300 END
```

This prints the values −400 and −1 (because MYDATA is negative).

SIN *(num-exp)*

Type: Function
Description: **SIN** returns the sine of *num-exp* where *num-exp* is in radians.
Example: **PRINT SIN(1.25)**

This prints .9489846194

SIZE

Type: Extended BASIC Command
Description: **SIZE** prints the number of bytes of unused memory.
Example: **SIZE**

This tells you how much memory is not used by the program and data currently in memory. If you use this right after a **NEW** command, you will see how much memory you can use for your BASIC programs. If you use **SIZE**

after a program has finished, you can see how much memory the program used.

CALL SOUND *(duration,freq1,vol1[, . . . ,freq4,vol4])*

Type: Statement or Command
Description: **SOUND** controls the tone and noise generator. You get a tone of frequency *freq1* at *vol1* for *duration* milliseconds. You can get up to four simultaneous tones (all for the same *duration*).
Example: **CALL SOUND (4000,262,0)**

This plays a middle C for 4 seconds (4000 milliseconds) at loudest volume.

CALL SPGET *(word-string,return-string)*

Type: Extended BASIC Statement or Command
Description: **SPGET** gets the speech bit pattern in *return-string* for *word-string*.
Example: **CALL SPGET ("HELLO",DATA1$)**

This puts the bit pattern for the word HELLO into string variable DATA1$. You use DATA1$ with the **SAY** statement.

CALL SPRITE *(#sprite-num,char,spr-color,dot-row,dot-col [,row-vel,col-vel] [, . . .])*

Type: Extended BASIC Statement or Command
Description: **SPRITE** activates one or more sprites. *char* is an ASCII value for a character whose definition is set by **PATTERN** or **CHAR**; *spr-color* is the color of sprite; *dot-row* and *dot-col* are the location of the sprite on the screen; and *row-vel* and *col-vel* set the speed and direction of motion of the sprite.
Example:

```
100 CALL CHAR (128, "66FFFFFF7E3C1818")
200 CALL SPRITE (#15,128,6,75,75,20,10)
300 END
```

This makes a sprite that looks like a purple heart and starts it moving down and to the right.

SQR *(num-exp)*

Type: Function
Description: **SQR** returns the square root of *num-exp*.
Example: **PRINT SQR(100+525)**

This prints 25.

STOP

Type: Statement or Command

Description: **STOP** terminates program execution. You can use **STOP** statements anywhere in your program except after subprograms.

Example:

```
100 PRINT "HELLO THERE"
200 INPUT "DO YOU WANT TO SEE ME STOP? (Y/N) " : Y$
300 IF Y$ <> "Y" THEN 600
400 PRINT "OK"
500 STOP
600 PRINT "I'LL KEEP GOING UNTIL YOU SAY TO STOP."
700 GOTO 200
800 END
```

This shows you how to put **STOP** statements in the middle of a program.

STR$ *(num-exp)*

Type: Function

Description: **STR$** converts *num-exp* into its string form. **VAL** works the other way, changing a string to a numeric form.

Example: **STRVAL$=STR$ (VALUE)**

This converts the number in VALUE to character string format and puts the string into STRVAL$.

SUB *subprog-name* [*(parameter-list)*]

Type: Extended BASIC Statement

Description: **SUB** is the first statement in a named subprogram in Extended BASIC. You use a **CALL** statement to transfer control to the subprogram. You can pass values to *subprog-name* through the optional *parameter-list.*

Example: **SUB MYSUB**

This is the first statement for the named subprogram MYSUB. When you want to use MYSUB, you **CALL MYSUB.**

SUBEND

Type: Extended BASIC Statement

Description: **SUBEND** marks the end of a named subprogram (one that starts with a **SUB** statement).

Example:

```
100 CALL PRNT
200 STOP
300 SUB PRNT
```

```
400  PRINT "HERE I AM IN THE SUBPROGRAM"
500  SUBEND
600  END
```

This shows you how to use a named subprogram PRNT.

SUBEXIT

Type: Extended BASIC Statement
Description: **SUBEXIT** transfers control from a named subprogram to the statement following the **CALL** statement for the subprogram.

Example:

```
100  CALL PRNT
200  STOP
300  SUB PRNT
400  PRINT "HERE I AM IN THE SUBPROGRAM"
500  INPUT "WANT ME TO CONTINUE? (Y/N)"  :  Y$
600  IF Y$ <> "Y" THEN SUBEXIT ELSE 400
700  SUBEND
800  END
```

This shows you how to use a SUBEXIT statement to return from a subprogram.

TAB *(num-exp)*

Type: Function
Description: **TAB** positions **PRINT** or **DISPLAY** statements at column *num-exp*.
Example: **PRINT TAB(10);"ABC"**

This prints the letters ABC beginning at column 10.

TAN *(num-exp)*

Type: Function
Description: **TAN** returns the tangent of *num-exp* where *num-exp* is expressed in radians.
Example: **PRINT TAN(1.5)**

This prints 14.10141995

TRACE

Type: Command or Statement
Description: **TRACE** lists the line numbers of statements before they are executed.

Example: **TRACE**

This prints the line number of each BASIC statement before the statement is executed.

UNBREAK [*line-num-list*]

Type: Command or Statement
Description: **UNBREAK** removes the breakpoints for the lines in *line-num-list* or all breakpoints (if you don't use *line-num-list*). You set breakpoints with a **BREAK** command.
Example: **UNBREAK 200,350**

This removes the breakpoints set by a previous **BREAK** statement for lines 200 and 350. Any other breakpoints are still effective.

UNTRACE

Type: Command or Statement
Description: **UNTRACE** cancels a **TRACE** command. Line numbers are no longer printed before the statements are executed.
Example: **UNTRACE**

This makes the BASIC program run as usual, without printing line numbers before executing its statements.

VAL (*str-exp*)

Type: Function
Description: **VAL** converts *str-exp* to a numeric form.
Example: **NUMVAL = VAL("12.345")**

This translates the string "12.345" into numeric format and puts the value into the numeric variable NUMVAL.

CALL VCHAR (*row,col,ASCII-code*[,*repetitions*])

Type: Statement or Command
Description: **VCHAR** writes the character with the ASCII value *ASCII-code* at row *row* and column *col*. If you use a value for *repetitions*, you will get that many characters written down the screen beginning at *row,col*.
Example: **CALL VCHAR(10,15,63,10)**

This starts at row 10, column 15 and writes 10 ? (question marks) down the screen.

CALL VERSION *(num-var)*

Type: Extended BASIC Statement or Command

Description: **VERSION** returns the value of the Extended BASIC version currently being used.

Example: **CALL VERSION (BASVER)**

This puts the value of the current BASIC version into variable BASVER.

INDEX

TO THE READER

Sams Computer books cover Fundamentals — Programming — Interfacing — Technology written to meet the needs of computer engineers, professionals, scientists, technicians, students, educators, business owners, personal computerists and home hobbyists.

Our Tradition is to meet your needs and in so doing we invite you to tell us what your needs and interests are by completing the following:

1. I need books on the following topics:

_____

_____

_____

_____

_____

_____

2. I have the following Sams titles:

_____

_____

_____

_____

_____

_____

3. My occupation is:

_____ Scientist, Engineer
_____ Personal computerist
_____ Technician, Serviceman
_____ Educator
_____ Student
_____ D P Professional
_____ Business owner
_____ Computer store owner
_____ Home hobbyist
Other _____

_____

Name (print) _____

Address _____

City _____ State _____ Zip _____

Mail to: **Howard W. Sams & Co., Inc.**
Marketing Dept. #CBS1/80
4300 W. 62nd St., P.O. Box 7092
Indianapolis, Indiana 46206

22071

THE TI 99/4A USER'S GUIDE

- Covers the common problems that TI 99/4A owners have with their first computer.
- Will introduce you to the many options available for improving and expanding your TI 99/4A.
- Discusses the options you have for finding software and getting it up and running.
- Will tell you what peripherals are available for your TI 99/4A, explain what they do, and give you some suggestions on how you can expand your system.
- Shows how you can use your TI 99/4A to generate tones, sound effects, music, and speech.
- Provides information on TI BASIC, Extended BASIC, LOGO II, 9900 Assembly Language, UCSD Pascal, PILOT, and FORTH.

This book is for anyone who has a TI 99/4A or who may be planning to buy one.

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, Indianapolis, Indiana 46268 USA

$11.95/22071

ISBN: 0-672-22071-7