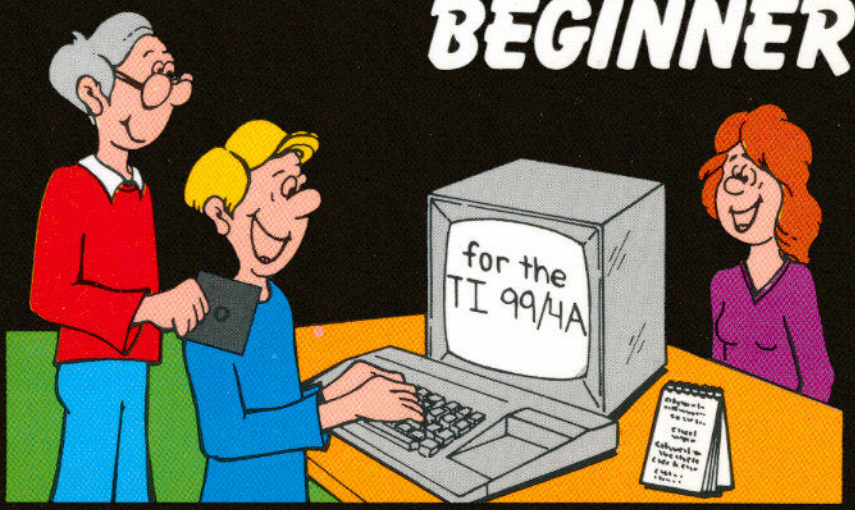
99/4A
99/4A
TI 99/4A®
99/4A
99/4A

FOR THE BEGINNING BEGINNER



ANOTHER GOOD IDEA BOOK FROM ENRICH
THE GOOD IDEA PEOPLE
INSTRUCTIONAL MATERIALS DIVISION OF OHAUS®

BASIC STATEMENTS & COMMANDS

CALL-Calls a built-in subprogram.

END-Marks the end of a program.

FOR-TO-STEP-NEXT-Marks beginning and end of a program loop.

GOTO-Causes the computer to jump to another place in the program.

IF-THEN-ELSE-Tests a conditional expression.

INPUT-Inputs information from the keyboard.

LET-Assigns a value to a variable.

LIST-Shows program in memory.

NEW-Erases program from memory.

OLD-Loads a stored program from tape or diskette.

PRINT-Prints an item or list of items on the display.

REM-A REMark inserted in a program listing.

RUN-Starts execution of the program in memory.

SAVE-Stores current program on cassette tape or diskette.

SUBPROGRAMS
(used with CALL)

CHAR-Allows the user to define special graphics shapes.

CLEAR-Clears the TV screen.

COLOR-Sets the colors of the next characters to be printed. The color of the characters (foreground) and of their background are both set.

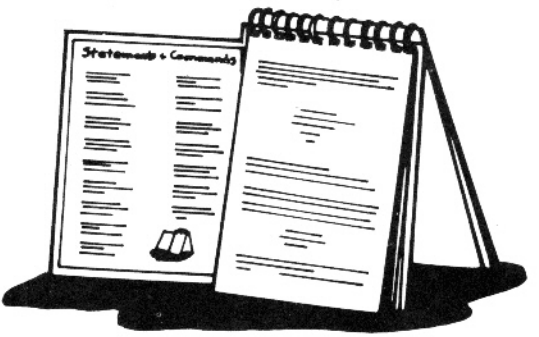
HCHAR-Places a character in any position on the screen, and repeats it horizontally.

JOYST-Reads the positions of the joysticks.

KEY-Reads a single keypress from the keyboard. Also reads the joysticks' firing buttons.

SCREEN-Sets the color of the TV screen.

VCHAR-Places a character in any position on the screen, and repeats it vertically.



This book's unique binding enables the user to prop it up by the computer when in use. Note illustration.

THE TI 99/4A
FOR THE BEGINNING BEGINNER

An easy and helpful introduction to computers and programming

Les Cowan

ENRICH/OHAUS
San Jose, California

Graphic Design by Kaye Graphics

Cover Design by Kaye Quinn

Edited by Matt Foley, John Deubert, Jim Haugaard, and Eldon Kerr

Typography by KGN Graphics

Published by
ENRICH/OHAUS
2325 Paragon Drive
San Jose, CA 95131

ISBN: 0-86582-132-1
Catalog No. EN79225
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ABOUT THE AUTHOR

Les Cowan is a well-known author of books and articles in the computer field. He has been writing about technical subjects since the seventies. As a journalist (Masters of Journalism, U.C. Berkeley) he had a natural interest in all communications media, which involved him first in a number of video projects and then, via his first word processor, the world of computers. Since then he has written technical documents for a number of Silicon Valley corporations and has pursued an active parallel career in the trade and general press.

Mr. Cowan is also one of the few hardy souls who have forayed successfully into the world of software engineering without formal training. Having taught himself BASIC, Mr. Cowan accepted a position with a leading producer of educational software, for whom he designed and implemented a number of programs for teaching computer literacy at the junior high and high school level.

Mr. Cowan is presently a contributing editor of *Air-Cal Magazine,* for which he writes a monthly column on developments in high technology. He also is a regular contributor to *PC World, Popular Computing, Home Electronics and Entertainment* and has written occasionally for other computer-related periodicals. His last book for ENRICH/OHAUS was *"The Illustrated Computer Dictionary and Handbook."*

TABLE OF CONTENTS

INTRODUCTION

If you are reading this, you are probably interested in learning more about the Texas Instruments home computer, the TI 99/4A. You may already own one, or you may be thinking about getting one. If you have already looked at other books or manuals on "computer literacy", you may be a little bit confused by the swarm of new words and unfamiliar ideas. You are not alone.

A computer cannot do anything unless someone tells it what to do. To tell a computer what to do you must use a "language" that the computer understands. Like any language, computer language has certain rules. These rules may seem overwhelming at first, but once you learn them, talking to the computer will be easy.

This book will guide you through everything you need to know, to start telling your computer what to do. The first chapter helps you get started by taking you step by step through the process of getting your TI 99/4A properly connected and turned on. Chapter two gives you a quick rundown of the history of computers and of how they work. This information is organized from a practical standpoint. If you understand how a computer works, then the things you must do to communicate with it will make more sense. If something makes more sense, it is easier to learn.

Some readers may not wish to spend time on the first two chapters. Some of us like to dive right in. If you are among this group you could start at chapter three. Here we begin learning the building blocks of a computer language called BASIC. We start with the easiest blocks and gradually build up, one step at a time. You are encouraged to try everything out on the keyboard. Experiment; learn by doing!

In chapters four and five we learn the tricks of programming,

and we finish by putting together a long program that demonstrates all the important abilities of the TI 99/4A. If you follow each step along the way, you will soon be ready to start designing your own programs with confidence.

This book was written for the beginner who knows nothing about computers. It uses simple language, and avoids jargon and buzzwords. In this book you will learn gradually, starting at the simplest possible level, and proceeding at whatever pace is most comfortable for you. We have done everything possible to make this book an effective learning tool that is easy to use and, most importantly, fun.

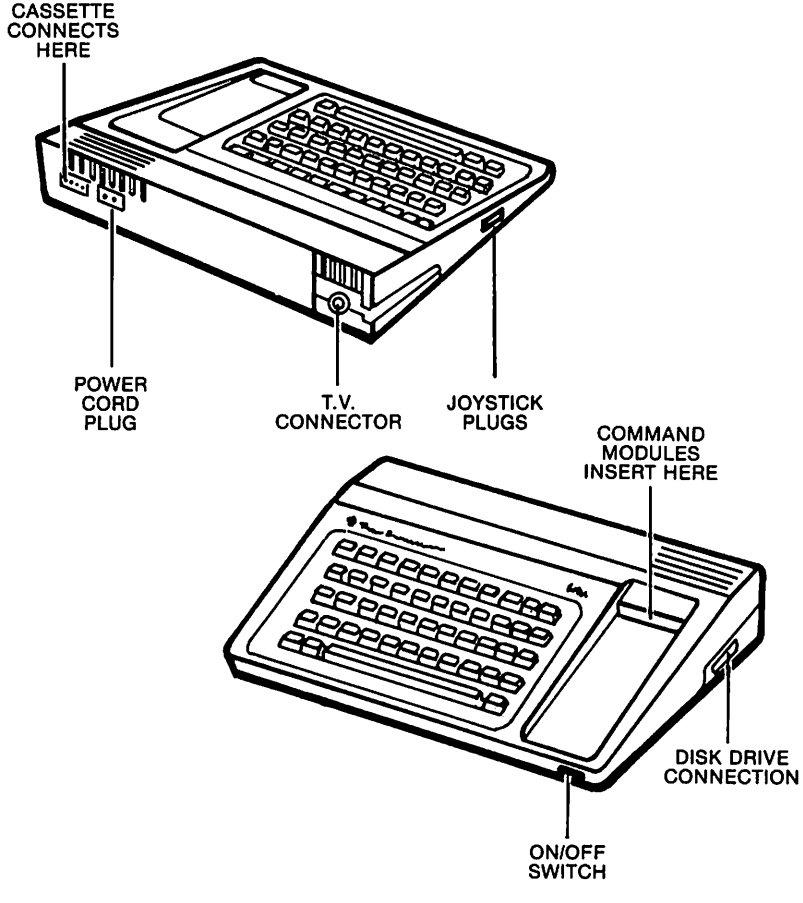
Two very important aids, a glossary and an appendix, are provided for quick reference at the end of this book.

The author wishes to thank the Texas Instruments San Francisco Learning Center and Mr. David Loennig of Texas Instruments' Lubbock, Texas office for their friendliness and cooperation.

STARTING UP THE TI 99/4A

Turning on a computer can be a little more complicated than turning on a television set, but with a little practice, the procedure will become second nature. If you carefully follow the steps explained below, you will be "up and running" in no time.

Refer to the illustration below of a Texas Instruments 99/4A home computer.



Place your computer in a convenient, well-ventilated spot where it will be safe from static electricity and the screen will not be exposed to glare.

To start, let's attach the computer to a television set or a TI monitor.

If you are using a monitor other than the TI monitor, it must have an audio connection (not all monitors do) or you will not be able to use the 99/4A's musical capabilities. You will also probably need a special adaptor, since the monitor socket on the TI is not standard.

If you are using a TI monitor, it will come with a connecting cable. The round plug at one end of the cable goes in a socket at the left rear of the computer. At the other end of the cable are a small audio (sound) plug and a slightly larger video (picture) plug. These go in the corresponding sockets in the monitor.

If you are using a television set, use the video modulator box that comes with your computer. A long, round cable is attached to one end of the modulator and a very short, flat antenna cable is attached to the other. At the end of the round cable is a round plug which goes in the socket at the left rear of the computer. If you want to use the television for normal watching as well as for computing, attach your normal antenna to the two screws on the side of the modulator. On the largest side of the modulator is a switch. In one position, marked 'TV antenna', this switch lets you watch television. In the other position, marked 'modulator', it lets you use the computer. On the bottom side of the modulator, next to the long, black cable is another switch. In one position it uses channel three on your television, and in the other position it uses channel four. Set this switch to the channel which does not broadcast in your area, and select that channel on your television dial.

Without turning anything on yet, plug the monitor or television into an electrical wall outlet.

The computer takes much lower voltage than comes from a wall outlet, so it uses a transformer. This is the rather heavy black box that looks like an overgrown electric plug. Coming out of the transformer is a long cable ending in a power plug that has four holes in it. On the right rear of the computer are two sockets. The one toward the center has four prongs in it. Plug the power plug into this socket. Then plug the transformer into a wall outlet.

TIP! It is a good idea to unplug the transformer when not in use.

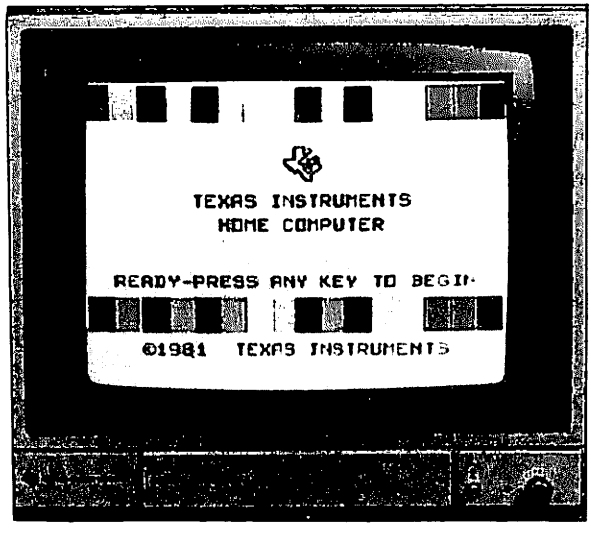
Face the computer, with its keyboard nearest you.

Sit comfortably.

Turn on the monitor or television.

On the far right of the narrow front of the computer, facing you, is a sliding switch, and to the left of it a small, red light. Slide the switch to the right. The light should come on and you should hear a beep. An instant later you should hear another beep as the display appears. If you did not hear a beep it is probably bacause the volume control on your television or monitor was turned down.

The display is bordered at top and bottom with bands of colored squares. In the center is the Texas Instruments logo, and below it the message, "PRESS ANY KEY TO CONTINUE".



You have "fired up the system" and are now ready to command the computer to do your bidding.

Start by responding to the message. Press any key on the keyboard except the keys marked 'ALPHA LOCK', 'CTRL', 'FCTN' or 'SHIFT'. You will hear a beep and the display will change to a solid background color with the TI logo at the top, and, near the top left, the word 'PRESS'. Under that will be

1. FOR TI BASIC

This is your first menu.

Right now the menu offers you only one choice: TI BASIC. Later we will see how you may increase your choices by using command modules or disks. For now, since we have only one choice, let's take it. Press the number key marked '1'. The computer will again beep, the display will go blank momentarily and the message "TI BASIC READY" will appear near the bottom of the screen. Below that a right arrow head (>), called

a prompt*, and a blinking square, called a cursor, will appear. On some televisions the prompt, the cursor and the 'T' in TI may be partially or completely hidden. These televisions cut off the two leftmost and two rightmost columns, especially near the bottom and top of the screen.

The message "TI BASIC READY" tells you two things. One is that you are using TI's version of the programming language called BASIC. The other is that the computer is not doing anything in particular at the moment and is waiting for you to tell it something in the BASIC language.

The Keyboard

The keyboard of the TI99/4A is very similar to a typewriter keyboard. What you type on the keyboard will be printed on the television display. However, some keys have special uses. To help you remember the special uses, Texas Instruments has provided some special "overlay strips" which can be added to your keyboard.

Before we go on, find the plastic keyboard overlay strips which came with your TI 99/4A. These are about a half inch wide and about ten inches long. On the front, they are divided into little squares. On one of the strips, some of these squares contain words like DEL, INS, ERASE, and so on. Lay this strip down at the top of the keyboard (on the little shelf over the number keys) so that the word DEL is directly above the 1 key and the word QUIT is directly over the key marked with a + and an = sign.

Now if the blinking square is still on your screen, we will proceed. If the square has disappeared, hit any key and it will

---

* Certain words, like "prompt", have special meanings in the world of computers. A prompt is a character displayed by a computer to tell the user that an action is expected of him.

Other special words are defined in the glossary at the back of this book.

come back. When the TI 99/4A has been left alone for about 9 minutes, the display goes blank to avoid leaving a permanent image on your television screen.

The blinking square is called a cursor. The cursor is simply a marker. It shows the point on the screen where your next typed character will appear.

Go ahead and type a letter. Have no anxiety—absolutely nothing you do at the keyboard can in any way damage the computer. You see that the letter you typed appeared where the cursor was, and the cursor moved over to the right.

This single letter does not mean anything to the computer, so let's erase it. Find the S key. Notice that on its front side is an arrow pointing left. This is the backspace key. If you press it by itself it prints the letter 's', but if you hold down the key marked FCTN and then press this key, the cursor will move one space to the left. Try it.

Now you have learned one of the most important differences between a typewriter keyboard and a computer keyboard. In addition to the usual letters, numbers and punctuation keys, computer keyboards have certain special duty keys. Let's see what these are on the TI 99/4A. As we explain the keys below, find each one on your keyboard. Try them!

Special Keys

FCTN FCTN stands for FUNCTION. We have already met this key. It is always used together with a letter key or a number key. If used with a letter key it will move the cursor or print whatever character is shown on the front of the key. If used with a number key, it will perform certain editing functions which we will learn about in the section on editing.

FCTN 4 The FCTN key pressed at the same time as the 4 key will do a number of things you should know about. If the computer is doing something, FCTN-4 will stop it in its tracks, and the computer will display a message telling you where it stopped. If a program is being listed, FCTN-4 will stop it too. (If you do not understand some of this yet, don't worry. We have to start somewhere, and you will understand shortly.)

FCTN + When the FCTN key is pressed along with the key that is labelled with a plus sign , it will stop everything. This is called a "reset" and will send you back to the start-up display.

SHIFT These keys are just like shift keys on a typewriter. While they are held down, any letter key will produce an upper case letter. Any number key will produce the mark shown above the number on that key.

ALPHA LOCK This key is like a shift-lock key on a typewriter. If it is pressed once it stays down until it is pressed again. In the meantime, all letter keys will produce upper case letters regardless of the SHIFT keys. *Number keys, however, will be unaffected.*

CTRL CTRL stands for CONTROL. Like the FCTN key, this key is only used together with other keys.

ENTER This key tells the computer to receive information that you have typed. When you type characters that appear on the screen, the computer could not care less. Once you press ENTER, however, the computer will try to do something about those characters.

When you talk to the computer in BASIC, you talk in "statements". (We say talk, but of course you actually type on the keyboard.) A statement usually consists of a command and something called an argument. For example, take the PRINT statement. The PRINT statement consists of the word PRINT followed by whatever it is you want the computer to print, which we call the "argument".

Try this. First, press the ENTER key. The computer beeps and displays the message: 'INCORRECT STATEMENT'. This is an error message, and it means that something happened that the computer could not understand. In this case, you typed in a letter and then later hit ENTER. The computer was expecting a BASIC statement. No BASIC statement consists of a single letter, so the computer was confused.

Now type this:

```
PRINT "SELL THE WINE."
```

Be sure to type it exactly as shown, with the quotation marks. (You get the quotation marks by holding the FCTN key and pressing P.) The quotation marks tell the computer to print exactly what is inside them. You might remember their importance by saying to yourself, "I want the computer to quote me."

Now press ENTER. The computer will display:

```
SELL THE WINE.
```

This will appear under your PRINT statement. The computer has just obeyed your first command!

Let's practice the PRINT statement a few more times. Type:

```
PRINT "WHY DID THE CHICKEN CROSS THE ROAD?"
```

Now press ENTER. The computer will display:

WHY DID THE CHICKEN CROSS THE ROAD?

Nothing to it, right? Try a few more on your own.

Exploring The Keyboard

Now that we are familiar with the layout of the keyboard, let's see what it can do. Type as many characters as you can—numbers, letters, punctuation marks, everything! Just don't use the FCTN, CTRL or ENTER keys.

After you have typed four lines the cursor will not go any further. Okay, let's back up. Use the backspace (**FCTN S**). The cursor moves to the left.

Hold the FCTN key down and hit the T, U, I, O and P keys. These are keys that, in combination with the FCTN key, type a right bracket, an underline, a question mark, an apostrophe and a quotation mark.

Now hold the backspace down. The cursor continues to move until it reaches the beginning of the first line you typed. The TI 99/4A has automatic repeat, which means that any key held down for more than a second will repeat itself until you release it. Try it with a few keys.

Let's play with the cursor control keys. Type a line of characters and use the backspace to move the cursor to the beginning of the line. Now hold down the FCTN key and the D key. The cursor moves to the end of the line and keeps going. But the line has not been erased. This is called non-destructive cursor movement because no characters are destroyed. The backspace is also non-destructive. Now backspace to the beginning of the line and hold down the space bar. The line is erased. This is destructive cursor movement.

Now press the ALPHA LOCK key so that it stays down. Type a few letters and numbers. Release the ALPHA LOCK key and repeat what you have typed. Notice the difference between upper and lower case characters on the screen.

Using The Display

The display screen of the TI 99/4A, whether it is a monitor or a television set, has room for 768 characters. It is divided into twenty-four horizontal rows of thirty-two columns each. Whenever we refer to locations on the screen we use rows and columns to describe the location.

You might think of the screen as a grid of little boxes, each one big enough to hold one character. The TI 99/4A lets you change the color of the screen, clear it, and put characters in any of the boxes.

Make sure the cursor is at the beginning of a line. (Press ENTER. You may get the INCORRECT STATEMENT or some other error message, but do not worry about that.) Type the following and press ENTER. (We will show you where to press ENTER by adding this sign: **ENTER**.)

CALL CLEAR **ENTER**

Once you press **ENTER**, the screen goes blank except for the cursor at the bottom left corner.

The CALL CLEAR statement actually tells your computer to run a small program that clears the screen. The TI user's manual refers to these as subprograms, which is just another way of saying that, as programs go, they are pretty small. This subprogram, and some others, are built into the computer's permanent memory. (This kind of memory is called ROM. It will

be explained in the next chapter.) When the computer is commanded to run such a subprogram, we say that the subprogram is 'called'.

Let's call another subprogram, whose name is HCHAR. (The H stands for horizontal and CHAR stands for character.) Type:

CALL HCHAR (10,10,65) **ENTER**

Remember that when we show **ENTER** it means to press the **ENTER** key. Also, in this book we will show zeros with a slash through them, like this: Ø. This is to make it look different from the letter "O". However, on your TV screen the TI computer shows zeroes as plain circles: 0.

The letter A appears on the 10th row, in the 10th column (and then hops up one line as the cursor moves down one line to prepare for your next command). Sixty-five is a code for the letter A, as we will see in the next chapter. The two tens and the sixty-five are all arguments of the CALL HCHAR statement.

These three arguments have to be there, inside parentheses, for the computer to understand the CALL HCHAR statement. You may add a fourth argument which is optional. Type:

CALL HCHAR (10,10,65,6) **ENTER**

(By now you know to press ENTER after a statement, so instead of mentioning it every time we will just use the sign **ENTER** to remind you.)

Using '6' as an extra argument causes the letter A to be repeated 6 times. Try the same statement using a different argument at the end. For example, try:

CALL HCHAR(10,10,65,15) **ENTER**

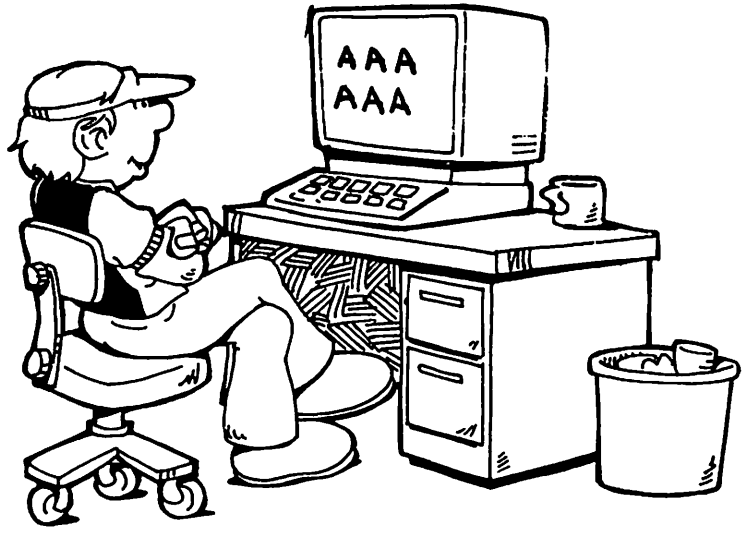
HCHAR has a sister subprogram called VCHAR. You guessed it, the V stands for vertical. Type:

CALL VCHAR (10,10,65,6) **ENTER**

A vertical row of six A's appear.

You might play with CALL HCHAR and CALL VCHAR using different arguments. Try putting characters on the edges of the display as well as various locations in the middle. Remember that the display has rows numbered 1 to 24 and columns numbered 1 to 32. If you use an argument that is smaller or larger than these numbers you will get an error message.

If you want to try other letters besides A, refer to the table of ASCII codes in the appendixes. Each character on the keyboard has its own code number. Have you ever sent a coded message in which every letter is replaced by a number—maybe 1 stands for A, 2 stands for B, and so on? The codes in the ASCII table are similar except that 65 is A, 66 is B, and so on. (The next chapter will explain what ASCII is.)



What Is A Program

A computer cannot do anything by itself. It does things only in response to your commands. You have already issued commands to your computer in the form of BASIC statements. You can also put a number of statements together, and the computer will obey them, one after the other. Such a list of statements is called a program. A computer obeying a program (we say "running" a program) can do much more complicated things than a computer obeying a single statement.

We will learn how to write programs in Chapter IV, but right now let's look at a simple program so that we can understand how a program works.

Here is a program that uses statements with which you are already familiar.

```
10 CALL CLEAR ENTER
20 PRINT "HELLO PAL, WHAT'S THE GOOD WORD?" ENTER
30 PRINT "THE GOOD WORD IS CHERRY JELLO." ENTER
40 END ENTER
```

Before typing this program, go through the start-up procedure described earlier so that you are in TI BASIC and the prompt and cursor are showing on the display. If the computer is already on, use **FCTN +** to reset it to the start-up display, and proceed from there as if the computer had just been turned on.

Type in the program exactly as shown, beginning each line with its number and ending each line by pressing the ENTER key. Make sure the ALPHA LOCK key is down so that all the characters are in upper case. This is a good idea whenever you are writing a program.

The numbers beginning each line are a feature of BASIC which we will discuss more fully later. Line 40 is an END statement which stops the program.

Once this program is entered, type:

RUN **ENTER**

The screen changes color momentarily, clears and displays:

```
HELLO PAL, WHAT'S THE GOOD WORD?
THE GOOD WORD IS CHERRY JELLO.
DONE
```

DONE is a message from the computer telling you that it has finished running the program. Now type:

RUN **ENTER**

The same thing happens. The program is still in the computer, waiting to be run. In fact, the program is stored in the computer's memory. Whenever you type RUN **ENTER,** it will run. As long as a program is in memory you may examine it by typing the LIST command. Type:

LIST **ENTER**

Beneath the LIST command the computer displays:

```
10 CALL CLEAR
20 PRINT "HELLO PAL, WHAT'S THE GOOD WORD?"
30 PRINT "THE GOOD WORD IS CHERRY JELLO."
40 END
```

You may,however, want to run a different program. To do that, this program must be removed from memory and a new one put in. You remove the current program by typing:

NEW **ENTER**

Try it. Now type:

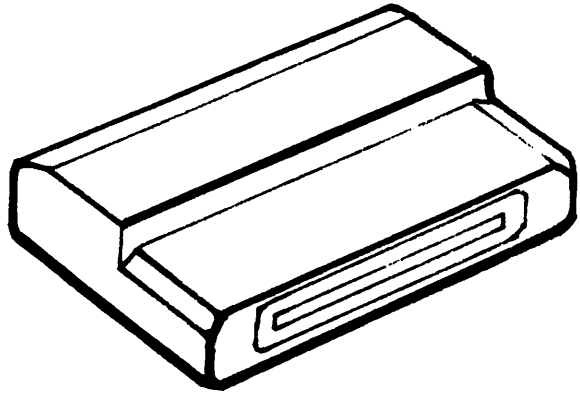
LIST **ENTER**

again. The message, "CAN'T DO THAT" appears, to tell you that, since no program is in memory, the computer cannot LIST one.

Now, if you wished, you could type in another program, or the same one. But it becomes very tedious typing every program you want to run, especially when some programs are hundreds of lines long. You can put a program into the computer's memory much more easily by using a command module, a cassette recorder or a disk drive. These are three methods of storing a program. In each case the program can easily be put into computer memory (we call it 'loading' a program) whenever desired. Let's discuss each of these methods of storage.

Command Module

A command module is a plastic box about four inches long and three inches wide. On one edge it has an open slot. This end slides into the opening (hidden by a hinged flap) on the right of the TI 99/4A keyboard.



A command module contains electronic circuits like those that are used in the computer's memory. This kind of memory is called ROM, as we will explain in the next chapter. For now, just remember that a program is stored permanently inside a command module. As soon as that command module is plugged into the computer, the program can be loaded into the computer's memory.

TIP! Sometimes, if you plug in or remove a command module while the computer is turned on, the data displayed on the screen will become garbled. If this happens try resetting the computer by pressing **FCTN +.** *If this does not fix things, turn the computer off and on. This problem may be avoided by always turning off the computer before plugging in or removing a command module. However, whenever you turn off the computer or press* **FCTN +,** *make sure that any data in memory which you want to keep is first saved to tape or disk, as described later in this chapter.*

If you have a command module, plug it in. If the computer is already on, the start-up display will appear and the computer will invite you to press any key. If the computer is not on, turn it on and you'll get the same display.

When you press any key, the start-up menu will appear, but in addition to choice number one, TI BASIC, another choice will be offered to you. If you select it, the program in the command module will be loaded and you will be ready to run it.

Using a Cassette Recorder

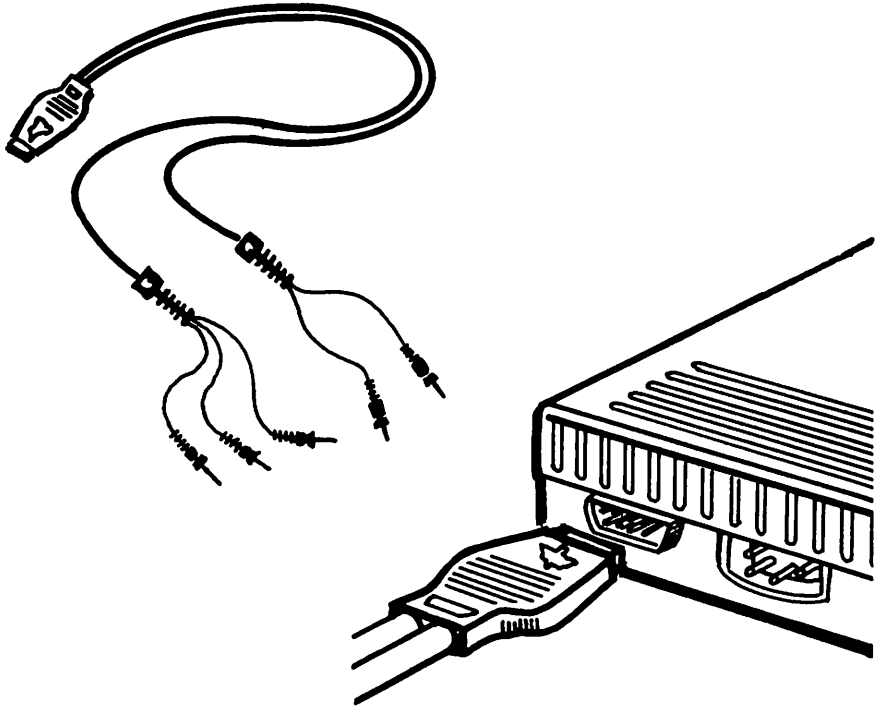
Another way to store a program is on a cassette tape. Programs stored on cassettes can be purchased from many stores, and some libraries will loan out cassette programs. On page 28 we will show you how to load a program from a cassette tape into your TI. First, however, we'll show you how to store a program on a cassette. Then you'll have a cassette to practice loading with!

Most quality cassette recorders can be used to save or load a program. Such a recorder should have these features:

- A volume control
- A tone control
- A microphone jack
- A remote jack
- An earphone or external speaker jack

It is also helpful if the recorder has a tape counter so that you can keep track of where a program is on a tape.

The recorder must be connected to the computer by a cassette interface cable which can be purchased wherever Texas Instruments computers are sold.



One end of the interface cable has a plug with nine holes. This plugs into a receptacle on the right rear of the computer. The other end has two sets of plugs: one set of three plugs and one of two. The two-plug set is for a second cassette recorder. The other set has a red-wired plug, a white-wired plug and a black-wired plug. The red-wired plug goes into the recorder's microphone jack. The black-wired plug goes into the recorder's remote jack (which is a smaller hole than the microphone jack.) The white-wired plug goes into the earphone jack, which may also be called the external speaker jack or the monitor jack.

The plug in the recorder's remote socket will let the computer start and stop the recorder. This will not work with all recorders, but you will still be able to use the recorder by starting and stopping it by hand. This is easy to do; just follow the directions on the display. We will proceed as if the remote function is not operative. (An adaptor is available for about $5 which will allow the remote function to work in some cases where it will not work otherwise.)

TIP! If you have trouble using a cassette player with your TI 99/4A, one thing to do is try it leaving the remote wire unplugged.

In order to store a program on cassette tape, the program must already be in the memory of your TI. Type a program into your TI by following these steps: Clear the computer's memory by typing NEW **ENTER.** Make sure no program is in memory by typing LIST **ENTER** and getting the CAN'T DO THAT message. Now type in the simple program we used earlier. (See page 19.)

Next, be sure that the recorder is correctly connected to the TI 99/4A, and the volume on the recorder is set at about the middle of its range, and the tone control is set to its highest position, which is sometimes called treble. Put a blank cassette in the recorder. The program will be "written" on the

tape as a series of special clicks and beeps, which the computer can understand.

TIP! It is a good idea to use short-length cassette tapes. Cassettes over 90 minutes (45 minutes on a side) should not be used.

Now type:

SAVE CS1 **ENTER**

CS1 is an argument of the SAVE statement. It means that you are saving to cassette number one. Be sure that you type the argument in upper case. If you don't, the computer will not be able to understand it and will display an error message. If that happens, just press the ALPHA LOCK key and type the statement over.

When you have typed in the SAVE CS1 statement correctly, the computer will prompt you with the message:

```
REWIND CASSETTE TAPE          CS1
THEN PRESS ENTER
```

Rewind the tape a little past where you want to record the program. If the cassette is blank, you will normally rewind to the beginning of the tape. Press ENTER. The computer will prompt you with the message:

```
PRESS CASSETTE RECORD         CS1
THEN PRESS ENTER
```

Push the RECORD and PLAY buttons on the recorder to start it running. Then press ENTER on the computer keyboard. The computer will display the message:

RECORDING

Wait while the recorder saves your program to tape. Unless your television or monitor volume control is turned down, you should hear a tone followed by garbled noise, as the program is sent from the computer to the recorder. You will hear this noise whenever you are recording or playing back data with the recorder.

When this is done the computer will display the message:

```
PRESS CASSETTE STOP          CS1
THEN PRESS ENTER
```

Stop the recorder and then press ENTER. The computer will prompt you with the message:

```
CHECK TAPE (Y OR N)?
```

This is to ask you whether or not you wish to verify that the program has been saved accurately.

IF YOU CHECK

If you wish to verify the recording, type Y. (You do not need to press **ENTER**.) The computer will prompt you with the message:

```
REWIND CASSETTE TAPE         CS1
THEN PRESS ENTER
```

Rewind the tape to a point just before the recording you just made. Press **ENTER**. The computer will display the message:

```
PRESS CASSETTE PLAY          CS1
THEN PRESS ENTER
```

Start the recorder playing and press **ENTER.** The computer will display the message:

```
CHECKING
```

Now wait until the computer receives data from the tape and compares it to the data in its memory. If they match, the recording is good, and the computer will display the message:

DATA OK

The computer will then prompt you with the message:

```
PRESS CASSETTE STOP          CS1
THEN PRESS ENTER
```

Stop the recorder and press **ENTER.** Your recording is complete.

If the recording was not accurate, the computer will display the message:

```
ERROR—NO DATA FOUND
PRESS R TO RECORD
PRESS C TO CHECK
PRESS E TO EXIT
```

If you press R the computer will let you try recording the program again. If you press C the computer will check the recording a second time.

If you press E the computer will prompt you with the message:

```
PRESS CASSETTE STOP          CS1
THEN PRESS ENTER
```

Stop the recorder and press **ENTER.** The computer will probably display an error message that will help you figure out what went wrong. The first thing you should do in this case is to make sure that all cables are connected properly, that the recorder is plugged in and that the recorder's tone and volume controls are set correctly.

IF YOU DO NOT CHECK

If, when the computer asks whether or not you wish to check a recording, you press N (remember, you do not need to press ENTER), the computer will return to TI BASIC. You will see the '>' prompt and the flashing cursor.

Loading an Old Program

Once you have saved a program you may get it back (we say *load* it) from the tape by using the OLD command. The OLD command is similar to the SAVE command. To try out the OLD command, type:

NEW **ENTER**

This will erase any program in the computer's memory. Now, type:

OLD CS1 **ENTER**

TIP! Remember that the ALPHA LOCK key must be UP or this instruction may not work. However, the command must be typed in capitals.

The computer will display:

```
REWIND CASSETTE TAPE        CS1
THEN PRESS ENTER
```

Rewind the tape to a point before the program you want to load, then press **ENTER.** The computer will display:

```
PRESS CASSETTE PLAY         CS1
THEN PRESS ENTER
```

Start the recorder playing and press **ENTER.** The computer will display:

READING

You will hear noises from the television set or monitor (unless the sound is turned off). When they stop and the recording is done the computer will display one of two messages, depending on whether or not the program was loaded successfully.

IF THE TRANSFER IS NOT GOOD

If the recording on tape was not transferred successfully to the computer's memory, the computer will display:

```
ERROR DETECTED IN DATA
PRESS R TO READ          CS1
PRESS C TO CHECK
PRESS E TO EXIT
```

If you press R, the computer will ask you to rewind the cassette tape and start all over. If you type C the computer will display:

```
REWIND CASSETTE TAPE     CS1
THEN PRESS ENTER
```

Rewind the tape and press **ENTER.** The computer will display:

```
PRESS CASSETTE PLAY      CS1
THEN PRESS ENTER
```

Start the recorder and press **ENTER.** The computer will display:

CHECKING

When the check is over either the error message will be displayed or the computer will display:

DATA OK

If the DATA OK message is displayed, you will be back in TI BASIC, ready to give the computer further instructions.

IF THE TRANSFER IS GOOD

If the recording was successful the computer will display the message:

DATA OK

PRESS CASSETTE STOP CS1
THEN PRESS ENTER

Press **ENTER,** and you will be returned to TI BASIC, ready to give more instructions to your computer. At this time you should see the > prompt. Type:

LIST **ENTER**

The program you have loaded from tape will be displayed.

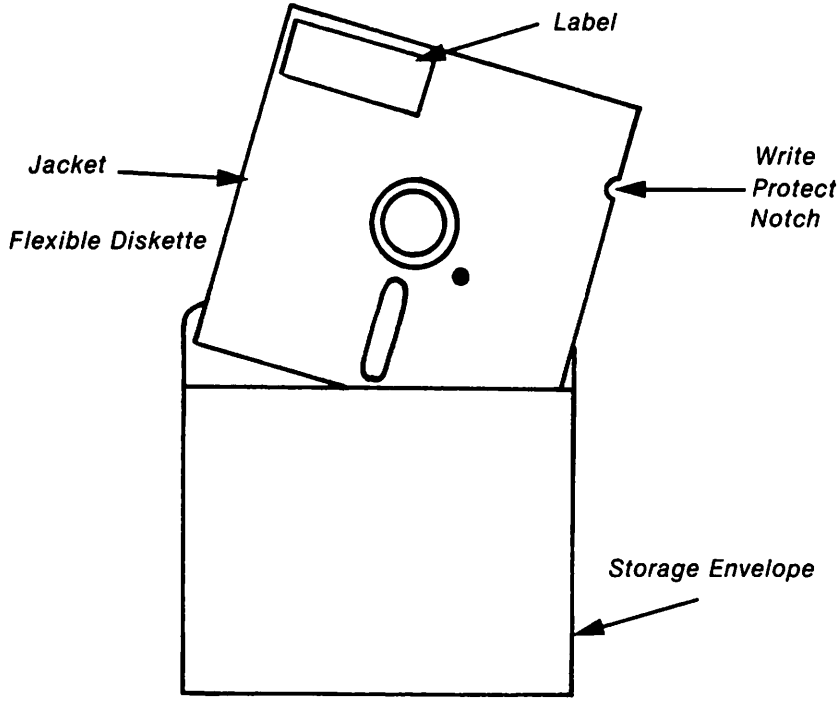
Using The Disk Drive

If you want to save data more quickly and easily than you can with a cassette recorder, you may wish to have a disk drive and save programs on diskettes.

Diskettes are thin, flexible, plastic platters, 5¼ inches in diameter, coated with a layer of iron oxide, the same material used to make cassette tape.

The difference between tape and diskettes is this: to find a piece of data on a tape you have to play all the tape before it. A diskette, however, spins like a phonograph record, and at the same time, the heads which read data from a diskette move from the center to the rim of the diskette. Thus data may be read almost instantly from any point on the diskette, just as you may put the needle down at any point on a phonograph record. This means that using a diskette drive to save or load programs is much faster than using a cassette recorder.

Diskettes must be treated with care. The diskette itself is permanently encased in a cardboard cover, and it is kept in a paper envelope.



Diskettes are very sensitive to magnetism, static, dust, smoke and heat. You should never touch a diskette's surface, and should keep diskettes away from magnetic fields, such as that around the TI monitor. Once a label is attached to a diskette, it should be written on only with soft felt-tip pens, never pencils or ball points.

The cardboard cover which encloses each diskette has a notch in the side. When this notch is covered by opaque tape, data may not be written to the disk. If you purchase a special program, or write one yourself, and you want to make sure it can never be erased from its diskette, then you can cover the notch of the diskette. Most diskettes are sold with little, sticky squares which can be taped over this write-protect notch.

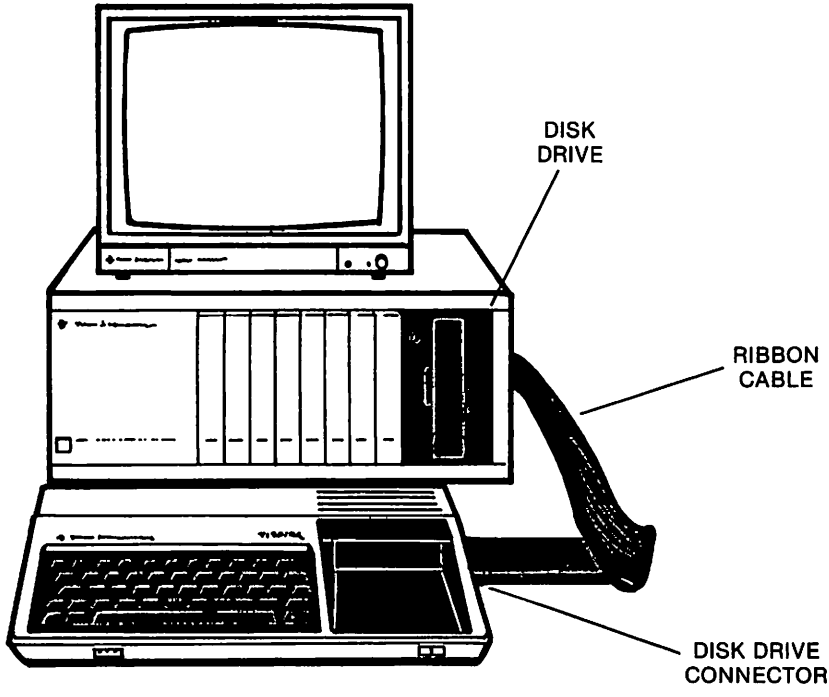
Data is sent between the computer and a diskette by a disk drive and a disk drive controller. The disk drive is a box which contains the motor that spins the diskette and the head which actually records data onto and reads data from a diskette. The controller sends the data between computer and disk drive.

Disk drives and controllers for the TI 99/4A may be either of two types. One type is a drive and controller which are each separate components, and are called external. The other type, called internal, is built into the TI Peripheral Expansion System.

An external disk drive is a box about three inches tall, six inches wide and ten inches long. An external disk controller is about two inches tall, six inches wide and ten inches long.

WARNING! Before connecting disk drives or controllers to the computer or to each other, make sure that power to all units is off.

The disk controller plugs into the slot on the right side of the computer. This is the same slot as the speech synthesizer uses. You may use the speech synthesizer and disk controller at the same time by plugging the speech synthesizer in as usual and plugging the disk controller into the slot on the right side of the speech synthesizer.



A flat cable (called a ribbon cable) comes out of the rear of the disk drive and plugs into a socket on the rear of the disk controller. Both disk drive and controller must be plugged into a power outlet. The disk drive and the controller should always be turned on before the computer is turned on. The disk drive power switch is located in back, and the disk controller power switch is a slide switch in front.

If you are using an internal disk drive, both the drive and the controller will be located inside a Peripheral Expansion System, a box about six inches tall, twenty inches long and ten inches deep. The Peripheral Expansion System connects to the computer with a wide, rubberized ribbon cable that ends in a flat plastic box. This box has a plug that fits into the socket on the right side of the computer. The Peripheral Expansion System must be turned on before the computer is turned on. The System power switch is on the left front of the box.

Before turning on the computer, you will need to plug in one of two command modules that let you use disks: Disk Manager 1 or Disk Manager 2. The first of these lets you use only single sided, single density disks. The latter lets you use single sided, single density or double sided, single density disks.

With a disk drive and controller, and one of the disk command modules, you may use commands described in the manual which comes with your disk drive. These will let you save programs to disks, just as you may save them to a cassette tape.

Whenever you save data to a disk, it is recorded on the disk in what is called a "file". Each file contains all the data saved at a single time. Each file has a name. When you do not remember what files are on a disk, you may see a list of them by *cataloging* the disk.

Disk Manager

When you are using one of the disk manager command modules, the TI start-up menu will include a choice called DISK MANAGER. (Other choices may be in foreign languages, and are the French and German versions of DISK MANAGER. The TI 99/4A is a real jet-setter.)

BEGIN, PROC'D, REDO, BACK

To use the DISK MANAGER you will need to understand the four function keys called BEGIN, PROC'D, REDO and BACK. Each of these is a combination of the FCTN key and one of the keys 5 and 6 or 8 and 9. There is a keyboard overlay strip for the DISK MANAGER which labels these keys for you.

FCTN 5 BEGIN tells the computer to display the first menu in a series of sub-menus. For example, if the computer were displaying the sub-menu titled 'Catalog Disk', pressing BEGIN would cause it to display the DISK MANAGER main menu.

FCTN 6 PROC'D (proceed) tells the computer to proceed with a menu selection that has already been made.

FCTN 8 REDO tells the computer to do over again from the beginning any procedure, even if you were in the middle of it.

FCTN 9 BACK tells the computer to display the menu that came before the one presently displayed. For example, if the computer were displaying the 'Catalog Disk' sub-menu and you pressed BACK, it would display the 'Disk Commands' sub-menu.

USING THE DISK MANAGER

Choose the DISK MANAGER option from the TI start-up display. If the computer is already on, press **FCTN +.** A title graphic will be displayed, similar to the TI start-up display, but saying DISK MANAGER instead. Press any key and you will see a menu. Choose the item called DISK COMMANDS. You will now see another menu.

INITIALIZATION

When you want to use a brand new disk, the first thing you must do is initialize it.

TIP! A disk only needs to be initialized once, when it is brand new. If you initialize it again, you will erase any programs recorded on it!

From the menu now displayed, choose the item: INITIALIZE NEW DISK. The computer will display:

```
INITIALIZE NEW DISK
MASTER DISK   1-3  ?
```

This is asking you which disk drive contains the disk you want initialized. If you have not already put the disk in a disk drive now is the time to do it. Then answer the question. If you have only one drive the answer can only be 1.

DEFAULT VALUES

Notice that as the cursor blinks, a number one appears. This is called a default value. If you do not type a number, and just press ENTER, the computer will assume that you are using drive number 1.

Now, to get back to initializing that disk. If you are using drive one, and you simply press ENTER, the computer will now display:

TRACKS PER SIDE

WHAT IS A TRACK?

The default value here is 40, as you can see when the cursor blinks. A track is a concentric ring around the center of a disk. Data is recorded only in these tracks. One thing that happens

to a disk when it is initialized is that these tracks are laid down on the disk's surface. The more tracks on a disk, the more information it can hold. Some disks only hold 35 tracks. If you are using such a disk, type 35 followed by **ENTER.** Otherwise, simply press **ENTER,** to get the dafult value, 40 tracks.

The computer will now display:

SINGLE SIDE (Y/N)

This is a yes/no question and the default is yes. If you have a disk drive that can record on either side of a disk, and your disk can be recorded on both sides (not all can), then type N followed by **ENTER.** Otherwise just press **ENTER.** The computer will display:

SINGLE DENSITY (Y/N)

If your disk drive and disk are capable of recording data at double the usual density, putting twice as much data in the same area of disk surface, then type N followed by **ENTER.** Otherwise just press **ENTER.** The computer will display:

INITIALIZE NEW DISK
WORKING. . . PLEASE WAIT

For a few seconds nothing will happen and then this display will be replaced with*:

Ø WORKING. . . PLEASE WAIT.

WHAT IS A SECTOR?

Each track is divided into nine sectors, and each sector can

---

*Remember, in this book we will show zeros with a slash through them, like this: Ø. This is to make it look different from the letter "O". However, on your TV screen the TI computer shows zeroes as plain circles: 0.

hold 256 bytes of data. A byte is the amount of data that is required to store one character. We talk about bytes some more in the next chapter.

The number 0 in this display is the number of the first sector to be initialized. As each new sector is initialized, the number displayed will change until the entire disk is initialized.

When the initialization is finished, you have a fresh, blank disk ready to record your programs. Remember, a diskette only needs to be initialized once in its whole life. Never initialize a disk that already has files on it, unless you don't mind erasing it.

OTHER DISK MANAGER COMMANDS

From the DISK MANAGER main menu you may choose file commands, disk commands, disk tests or single disk processing.

DISK MANAGER

Command	*What It Does*
File Commands	Displays a sub-menu from which you may copy a file between disks, rename a file, erase a file from a disk or change a file's protection.
Disk Commands	Displays a sub-menu from which you may catalog the disk, make a backup disk, change a disk's name or, as we have seen, initialize a disk.
Disk Tests	Displays a sub-menu from which you may run tests to make sure that the diskette, disk drive and controller are in good order.

Set Single Disk Processing	Lets the computer know that you are using only one disk drive. This is necessary when you are copying files between disks or copying disks.

Saving And Loading Files

Saving a program on a disk, or loading a program from a disk, is very similar to saving to or loading from cassette tape. The big difference is that programs saved on disk have *names.* You must know the name of the file you are loading or saving. If you are saving a program for the first time, you get to make up the name yourself.

You do not use the DISK MANAGER to save or load a disk file. You use the same instructions as with cassette tape: SAVE and OLD. Remember that to save to cassette tape you used the instruction SAVE CS1. Once you have a program in the computer's memory, that is to say, a program that you can see by typing LIST, you can save it to disk by typing:

SAVE DSK1.FILENAME **ENTER**

The number 1 after the DSK is the number of the drive you are using. If you have more than one drive this might be a two or three instead. Where we put the word FILENAME you type any name that you want to give the file. The only rules about file names are that they cannot be longer than 10 characters and may contain any characters except the period or a space. The DSK1 and the FILENAME must be separated by a period, and no spaces.

When you use the SAVE instruction, the disk drive light will go on and the drive will make a whirring sound for a few seconds. Then your cursor will return and the file has been recorded.

You may check to see that the file has been recorded by using the DISK MANAGER, selecting "Disk Commands", and cataloging the disk.

Now let's see how to load a file from the disk into the computer's memory. Type:

NEW **ENTER**

This will remove any program already in memory. Now type:

LIST **ENTER**

No program will be there, and the computer will display:

CAN'T DO THAT

Now type:

OLD DSK1.FILENAME **ENTER**

To load the same program you just saved on the disk, you must use the same FILENAME after DKS1 and the period. (If you ever forget what files are stored on a disk, just use the DISK MANAGER, choose "Disk Commands", and catalog the disk. This will give you a list of the names of all the files stored on that disk.)

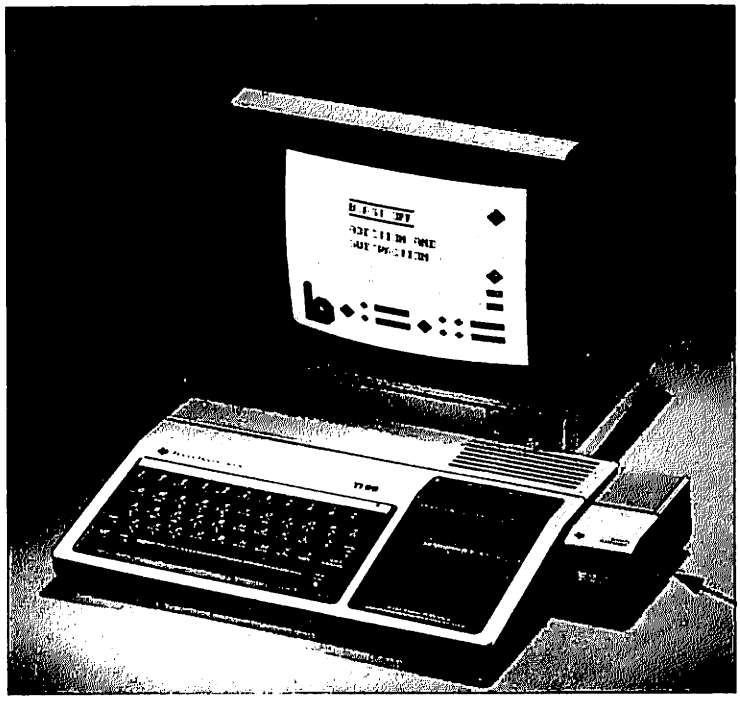
Again the disk will light up and whir. When it is done, type:

LIST **ENTER**

You will see that the program has been loaded back into the computer from the disk.

Using The Voice Synthesizer

The Texas Instruments Voice Synthesizer is a box about two inches high, two inches wide and five inches deep.



Looking at it from the front, where the Texas Instruments logo appears along with the words 'Speech Synthesizer', you will see on the left side a protruding plug about two inches long. On the right side of the 99/4A is a little plastic door. Lift the door and you will see a socket behind it. The speech synthesizer plug fits tightly into this socket.

WARNING! Do not plug the speech synthesizer in unless the computer is turned off and unplugged. This is to avoid damage from static electricity to the circuitry inside the speech synthesizer.

The speech synthesizer is sold separately from the TI 99/4A. It gives the 99/4A a voice so that it can talk to you, or to members of your family. It has a limited vocabulary of only 366 words, but the suffixes -ing, -ed and -s may be added to any of them. The speech synthesizer cannot be used without one of the command modules that support it. We will use the Extended BASIC command module to demonstrate the speech synthesizer. The Extended BASIC module contains BASIC commands beyond those that are already in the TI 99/4A. Two of these let you use the speech synthesizer.

If you do not have a voice synthesizer and command module for it, you may want to skip to page 43.

If you have the EXTENDED BASIC command module, plug it in. Plug in the speech synthesizer. Turn on the computer. The start-up menu will have two choices:

1 FOR TI BASIC

2 FOR TI EXTENDED BASIC

Press the 2 key. The screen will go blank and then the extended BASIC prompt will appear. This prompt is shorter than the TI BASIC prompt and just says:

'READY'

Type the simple program:

```
10 CALL SPGET("HELLO",WORD1$) ENTER
20 CALL SAY(,WORD1$) ENTER
30 END ENTER
```

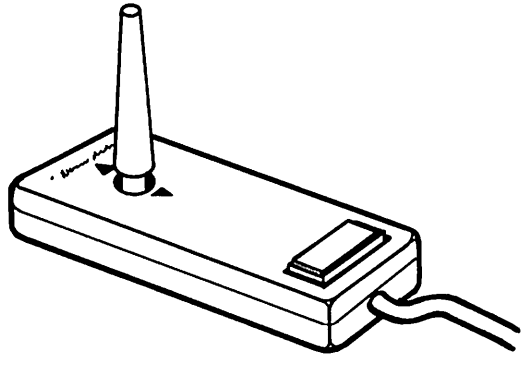
Be sure to put the comma in before the W in line 20. Now RUN the program. The computer will greet you with a cheery hello.

Do not worry that you do not yet understand how this program works. You will before finishing this book. Just be happy that you and your TI 99/4A are on speaking terms.

Using The Joysticks

When you are writing on the screen, the cursor moves along from left to right and from the end of one line to the beginning of the next. Sometimes, however, you may want to move something around the screen, from one point to another. Some games, for example, let you move space ships here and there on the screen. To do this, you use joysticks.

TI joysticks come in pairs, both joysticks connected to a single plug. This plug goes into a socket in the middle of the left side of the computer.



The joysticks are mounted in a plastic box. Whichever direction you move a joystick will be the direction in which your spaceship or whatever will move on the screen.

TIP! With some game software and other programs, the joysticks will not work unless the ALPHA LOCK key is up. The ALPHA LOCK key affects the signals sent by the joysticks to the computer. It is located at the lower left corner of the keyboard. Pressed once, it stays down, and pressed again it pops up. When using the joysticks make sure it is popped up.

To make the joysticks work you must have a command module or program that uses the joysticks.

Conclusion

In this chapter we got started with the TI 99/4A. You should now understand how to set up the TI 99/4A: how to plug it in, how to connect various peripherals such as the speech synthesizer and disk drives. You should be familiar with the procedures for saving a program to a cassette tape and loading it from tape back into the computer's memory.

You should be able to make the TI 99/4A obey a few simple commands, and you should understand how to use the keyboard to write text, backspace, forward space and perform several special functions.

Remember that you cannot hurt the computer in any way by playing around at the keyboard. Let your imagination roam freely. Use the "I wonder what will happen if I do this" approach. Make things happen. You will not understand all of them, but don't be confused. Everything will become clear as we go along. After all, the more questions you have, the more answers you will recognize when you come to them.

COMPUTER HARDWARE

History Of The Computer

Humans have used devices to help them calculate since before we began to record our history. No doubt the first calculating device was our fingers. Then came simple scratches in the ground, or rows of stones to keep track of numbers being calculated. As calculations became more complicated, early mechanisms were developed.

After the Renaissance in Europe, science, commerce and manufacturing began to advance quickly. By the seventeenth century, mathematics had become an important tool for solving the problems met in these and other activities. As mathematical problems became more complicated, especially after the discoveries of trigonometry and calculus, some scientists started thinking about how to use a machine to do calculations which took people a long time to do, and at which people made mistakes. In the seventeenth century, two mathematicians invented the earliest of such machines. In France, Blaise Pascal built a machine which could add numbers. The machine had several wheels, with the digits 0 through 9 on them. If you turned some of the wheels to the two numbers you wished to add, other wheels would automatically be turned so as to display the result. In Germany, Gottfried Leibniz built a similar machine which could multiply as well as add, but it did not always work.

The next great inventor of calculating machines was Charles Babbage, an Englishman who worked in the first half of the nineteenth century. Babbage contributed more to the history of computers as a thinker than as an inventor, because most of the calculating machines he thought up were never built. He did, however, build one machine that worked, and he was the first person to envision the basic parts that a computer must have.

The working machine that Babbage built was called a Difference Engine. It could add bigger numbers with more accuracy than Pascal's earlier machine, but it was still crude by our standards. Babbage's thoughts about what parts a true computer must have were not at all crude. He was the first person to think of a calculating machine as having input/output, an arithmetic unit (he called it a mill), a means of transferring data within the machine (electric circuits today, but mechanical gears in his machines), and a memory (he called it a store). Babbage also recognized the need for conditional operations; that is, he saw that no important automatic calculations could be done unless the machine could choose the route a calculation would take, depending on the result part way through the calculation.

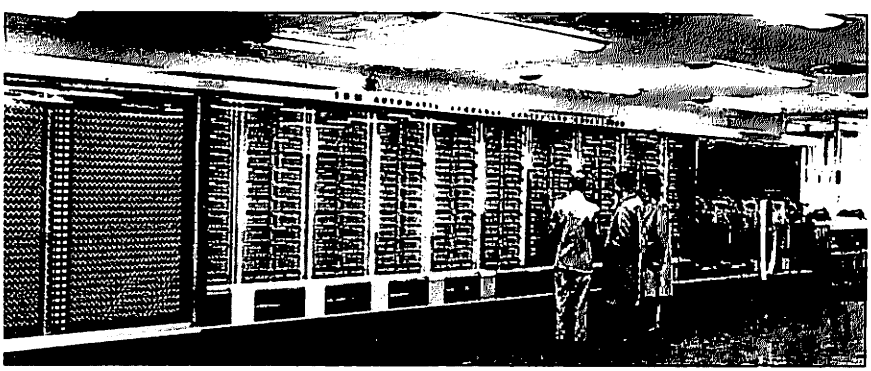
Finally, Babbage borrowed an idea from the French inventor, Joseph Jacquard. Jacquard had devised a way of punching holes in cards, and using these cards as part of a mechanism to control a loom. As different cards, with different patterns of holes passed through the mechanism, the pattern woven into the cloth changed accordingly. In effect, the loom was programmed with punched cards. Babbage was the first person to see that this method could be used to program a calculating machine.

Herman Hollerith was working for the United States census in 1880, and saw that a lot of time could be saved if some way could be found to count people automatically. During the next ten years he invented a system to use punched cards to hold data. Each card had a number of columns and rows, dividing it into dozens of little boxes. A piece of data collected about someone during the census could be represented by punching out certain of the holes. Then, all the cards could be put into a machine, the machine could add up all the holes in each position. Hollerith made these cards on the same machines used by the mint to make dollar bills. He used his invention to save immense amounts of time during the 1890 census, and went on to found a company which was the forerunner of IBM.

In the late 1920s, Vannevar Bush built the first useful analog calculator at the Massachusetts Institute of Technology. Like the machines of Pascal and Babbage, it was mechanical, but technological developments in the meantime made possible a more complex machine. The machine was constructed of many gears and rods, and filled a small room. It was used to do the large number of multiplications necessary to find the answer to certain kinds of mathematical problems. Some of its principles were later used in the first actual computer, the ENIAC.

Until the 1930s, calculating machines were thought of as being used only for "number crunching", that is, for solving mathematical problems that required many arithmetic operations. The goal was to use a machine to solve these problems much faster and with fewer errors than humans could. During the thirties, however, work on more complicated calculating machines gradually led various people to think about a machine that could do more than just calculate. Such a machine would be able to perform conditional operations, would be electrical instead of mechanical, would be programmable, would have some kind of memory—would be more than a calculator. It would be a computer.

One of the first machines that could be called a computer was the Mark I, built at IBM by Howard Aiken between 1939 and



1944. Like other early computers, it was built primarily for use in World War II. Weapons had become so complex that lengthy calculations were required to do things like aim long range artillery or set bombing sights. The military needed machines that could do these calculations faster than people. The Mark I was the first large digital computer. However, it was not electronic. It was electro-mechanical. That means that it used mechanical relays to open and close electronic circuits. The relays, in turn, were raised and lowered by electronic signals. But the relays themselves were still mechanical, and very slow compared to a completely electronic device.

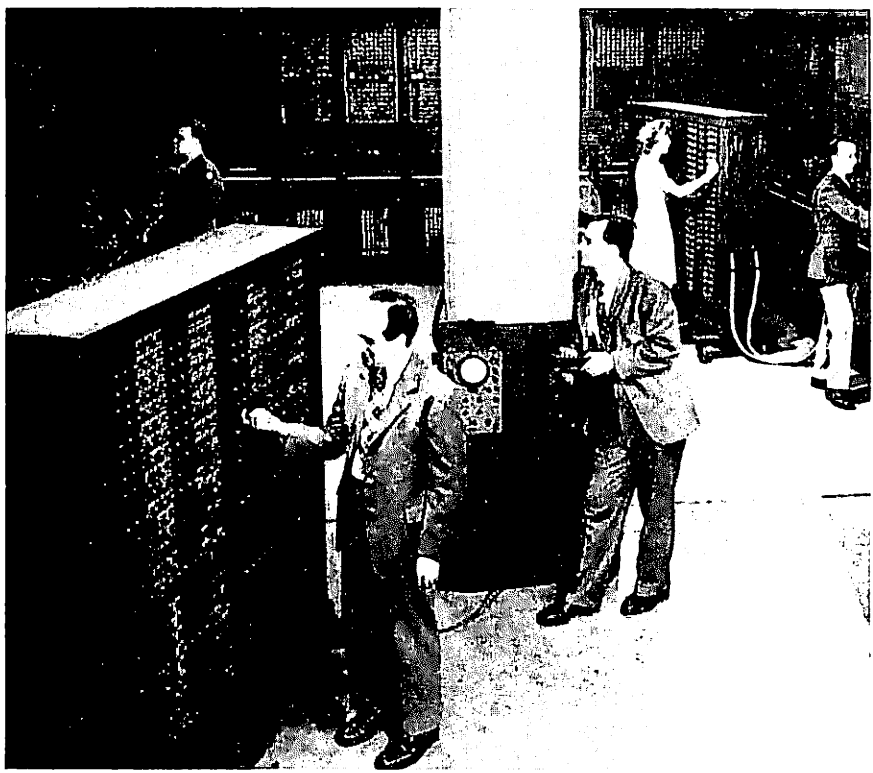
Another electromechanical device was built at Iowa State University by John Atanasoff and Clifford Berry in 1942. Atanasoff invented several new ways of designing a computer. Although his machine was not widely used, many of his ideas were adopted in building the first true electronic digital computer, the ENIAC.

The ENIAC (Electronic Numerical Integrator And Calculator) was built by a team headed by J. Presper Eckert and John Mauchly, at the University of Pennsylvania, between 1943 and 1946. The ENIAC was built for the army and was expected to perform the same kind of military calculations as the Mark I. Unlike that machine, however, the ENIAC was completely electronic. Instead of using mechanical relays to switch circuits on and off, it used vacuum tubes, the same kind that were used in radios. The ENIAC used over 18,000 vacuum tubes, which made it very large—about a hundred feet long, ten feet high, and three feet wide. It had a memory and was programmable and far faster than any calculating machine then in existence. Everything that the ENIAC could do can be done today by a small desk-top computer.

While Mauchly and Eckert were working on ENIAC, their work came to the attention of John von Neumann, a mathematician who was working on the atomic bomb project. Von Neumann

saw that the ENIAC would be useful for doing some of the long mathematical computations required by his work. As a result, he became a consultant on the ENIAC project and contributed some important ideas to it. He also helped write a number of reports in which the ideas being developed on the ENIAC project were systematically set forth, and expanded. The design of a computer, as described in these reports, is, with slight changes, the design of all modern computers, and it is often referred to as 'the von Neumann machine'.

The first generation of true modern computers followed soon after ENIAC. These computers stored their programs in memory, and could be programmed for much more complex jobs than was formerly possible.



The first fully programmable computer to be built was EDSAC, and it was completed in 1949, at the University of Cambridge in England. EDVAC was completed at the University of Pennsylvania in 1951, the same year as another computer was built by Mauchly and Eckert for Remington-Rand. This computer was the famous UNIVAC, the first commercial, electronic, digital computer. With its delivery to the Bureau of the Census, the era begun by Hollerith had come full circle, and a new age of computers was underway. Throughout most of the nineteen-fifties, large vacuum tube computers were made by such companies as IBM, Sperry-Rand (UNIVAC), and Burroughs. These machines were used for scientific calculations, and for a few commercial data-processing jobs.

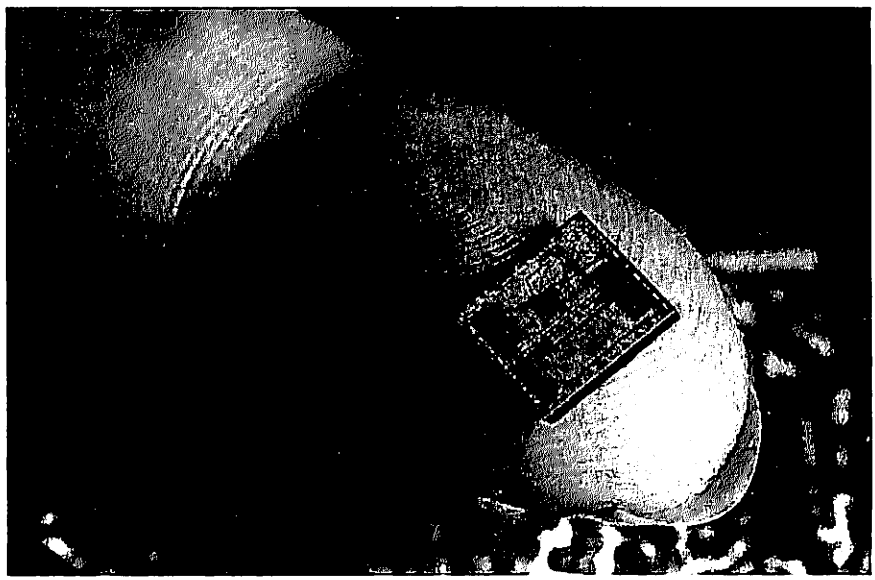
All the computers discussed so far were vacuum tube machines. In 1948 at the Bell Laboratories, John Bardeen, Walter Brattain and William Shockley invented a transistor. In 1956 they received a Nobel Prize for this work. The transistor performs the same functions as a vacuum tube, but it is tiny in comparison and uses far less electricity. With the transistor, new possibilities in computers were opened. In 1959 the first transistor-based digital computers were delivered. The solid-state era had begun, and with it, the second generation of computers.

Transistors brought greater speed, smaller size and increased flexibility. Because transistors occupied a fraction of the space that vacuum tubes did, second generation computers were much smaller. Also, computers with much more power could be built and take up no more space.

Second generation computers had more sophisticated memories. By the beginning of the second generation, internal memories had advanced from crude, early forms like mercury delay lines and electrostatic charges to magnetic drum and magnetic core memories, and tape storage had made an appearance. In the mid-60's systems were built with disk storage devices. Computers were used more and more for

commercial data processing in banks and other large institutions, as well as for military, government and scientific purposes. The leading companies during the sixties included IBM, NCR, UNIVAC and Burroughs. Control Data Corporation was founded during this time.

In 1964 the third generation of computers made their appearance. The third generation, which is still current, is identified with use of integrated circuits. An integrated circuit is a 'chip' of silicon, about a quarter inch square, on which thousands of microscopic transistors can be deposited. The circuitry on such a 'chip' is more complicated than that of the entire ENIAC.



By the early seventies, integrated circuits could be mass produced cheaply enough to make possible inexpensive computers whose entire processing unit was on a single chip called a microprocessor. At first the new microcomputers

were built by hobbyists, but soon products like the Texas Instrument 99/4A, Apple II, TRS-80, Commodore PET, Atari 800, and others proved that people would buy these small computers for their homes and businesses.

The result of all this is that today more and more people are learning about computers and using them. The computer power that fifteen years ago was available only to large corporations and government agencies now costs less than half the price of an automobile. Some computers cost less than $100. In the field of computer technology, new discoveries are made virtually every day. Improvements come along so fast that it is impossible for one person to keep up with all of them. The search for ways to make tasks easier, that began when a person first counted on his fingers, has never stopped.

How A Computer Works

A computer is not a single device, but a number of machines connected to each other. When the computer does a job, each device does a part of the job. That is why we often speak of computer *systems.*

The part of your TI 99/4A that looks like a small typewriter consists of a central processor, a memory and a keyboard. The central processing unit, or 'CPU', is the part that controls what the computer does. The CPU in your TI 99/4A is all one integrated circuit, or "chip", that is called a microprocessor.

The microprocessor used in the TI 99/4A is called a 9900. If you are interested in technical things, you might like to know that the 9900 is what is called a 16-bit processor, which makes it faster than the 8-bit processors in most other home computers.

DO IT NOW OR DO IT LATER

Computers do things only in response to instructions from a human. Two kinds of instructions are used with computers: immediate instructions and deferred instructions.

You may give your TI 99/4A an immediate instruction (called a 'command') and it will obey immediately. Or you may group several instructions (called 'statements') together and then tell the computer to obey them all in order. Such a group of instructions is called a program. When a computer is obeying the list of statements in a program, we say it is *running* the program. Once a computer is running a program, it will obey each instruction in the program automatically, until the program has finished running.

COMPUTERS DO THREE THINGS—
INPUT, OUTPUT, AND PROCESSING

A computer system does only three things: it inputs data, processes data and outputs data. ('Data' is just another word for 'information'.)

Typing a letter on the keyboard is one example of computer input. Input simply means data going into a device.

Processing is what the CPU does. Processing data simply means changing it or transferring it from one place to another.

What goes in may also come out, and data coming out of a device is called output. When the computer prints information on the TV screen, you are seeing an example of output.

Input/Output

Input/Output, or 'I/O', is how the computer shares information

with you. In order to tell the computer what to do, you must give it input. You have to "put in" your instructions.

The most common way of giving input to the TI 99/4A is through the keyboard. Each time you type a letter on the keyboard, a message is sent to the CPU (central processing unit). If you type a command into the computer, the CPU receives it letter by letter, and processes your 'input' when you press ENTER.

The computer can also input data from the joysticks, a tape recorder or disk drive, and other input devices.

When the computer has finished processing your input, when it has obeyed your command or run a program, it has to give 'output'. Otherwise, you wouldn't know it had done anything! The TI 99/4A usually displays its output, results or pictures, on the TV screen. You can also make it print output on a printer, or save it on cassette tape or floppy disk. If you have a speech synthesizer, the computer's output can be verbal!

Some I/O devices can only do one part of the communications job. A keyboard is *only* useful for input. A printer can *only* give output. However, some devices, like the disk drive and tape recorder, can both give data *to* the computer (input), and record data *from* the computer (output).

How Data Is Processed

The central processing unit of a computer only does two things: it transfers numbers from one place to another, and it adds, subtracts, multiplies and divides them. Yet the CPU is the "brain" of the entire computer system.

To understand how the CPU actually processes data, we should know something about how the insides of a computer are arranged and how the different parts work.

ENCODING

Computers use numbers to stand for all kinds of information: words, measurements, sounds, pictures and any other kind of information. This is called encoding.

Encoding is quite simple. We use codes all the time. For example, when we select channel seven on the television set we are actually telling the television to show us the program being broadcast at such and such a frequency. Seven is just a code for that frequency. It is much easier to remember the number seven than to remember, say, 80.2 kilohertz. Similarly, all information inside the computer is represented by numbers.

ELECTRONIC DATA

Numbers and letters are called alphanumeric characters. "q" is a character. "Q" is another character. All of the following symbols are characters: 1 " $ j @ ? H o 0 . Each character can be represented by a code. The most common computer code for characters is called the ASCII code, usually pronounced "AS-kee". In ASCII, the upper case letters A through Z are 65 through 90. The question mark is 63; the plus sign is 43; and so forth.

What this means is that whenever you press a key on a computer keyboard, the corresponding number is sent to the computer's CPU. Here is a table of the ASCII codes for all the different characters:

32	(space)	65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
35	#	68	D	100	d
36	$	69	E	101	e
37	%	70	F	102	f
38	&	71	G	103	g
39	'	72	H	104	h
40	(	73	I	105	i
41	)	74	J	106	j
42	*	75	K	107	k
43	+	76	L	108	l
44	,	77	M	109	m
45	-	78	N	110	n
46	.	79	O	111	o
47	/	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	:	91	[	123	{
59	;	92	\	124	|
60	<	93	]	125	}
61	=	94	^	126	~
62	>	95	_	127	(delete)
63	?	96	`		
64	@				

The Memory

Every computer must have a place to store data.

A computer's memory is composed of thousands of cells. Each cell can hold a number from zero to 255. Every character on the keyboard is represented by some number in this range, so we also say that each memory cell can hold one character. *This amount of memory, needed to fill one cell or store one character, is called a "byte" (pronounced "bite").*

K = KILO

The memory in a TI 99/4A can hold 16,384 bytes of data. We call this 16K of memory. The K stands for kilo, which means one thousand. Of course, 16,384 is more than 16,000. Computer memory is built in chunks, usually chunks of 4096 bytes. The smallest memory a home computer would have is 4096 bytes, or 4K. The next largest memory would usually be four times this, or 16K (16,384). No computer would have a memory of exactly 4,000 or 16,000 bytes. It is an approximation to refer to "16K".

RAM AND ROM

Computer memory is divided into two kinds, RAM and ROM.

ROM stands for Read-Only Memory. The computer can "read" data out of the ROM, but cannot "write" new data into it. The data in ROM is "built in" when the computer is assembled. Some of this data helps control the computer's most fundamental operations. The Command Modules you use with your TI 99/4A also contain ROM memories. ROM memory is permanent, and cannot be changed nor erased.

Most of a computer's memory, however, is used to store data temporarily. This kind of memory is called RAM. RAM stands

for Random Access Memory, which means that data in any part of the RAM can be read as quickly as that in any other part. The 16K we mentioned in the last section was 16K of RAM. The standard version of the TI 99/4A comes with 16K of RAM.

The data in RAM memory can be changed at any time, and is used for many different purposes by the computer. Program instructions are stored in RAM. So are words and numbers which are needed by the computer while running a program. There is a special section of RAM memory set aside as a sort of "scratchpad", which the central processor uses while doing arithmetic. Even the TV screen display is in a certain part of the changeable RAM memory.

ADDRESSING

That brings us to the question, how does the TI 99/4A find the right data in memory?

Each memory cell has a location, and each one is given a number, from 0 to 16,383. (In the computer world, numbering almost always starts with 0 instead of one.) The number of a memory location is called its "address". The computer is built so that the CPU can send a byte of data to any memory address or fetch a byte of data from any memory address.

So what kind of data does the computer keep in these regions, anyway? The most important are, two kinds: program data and storage data.

PROGRAM DATA

A program is a series of instructions which tell the computer to do things. Each instruction which makes up a program consists of several bytes of data. A program is stored in memory.

STORAGE DATA

Any data in memory that are not part of a program instruction, are called storage data. This includes many different kinds of data, but for our purposes we will concentrate on strings and variables.

STRINGS

A string is a group of characters always treated as a single unit. For example, the word 'computer' is a string consisting of eight characters. The expression '1234' is a string of four characters. Now, the string '1234' is not the same as the *number* 1234, as far as the computer is concerned. A string must always be treated as a whole. To be able to do anything worthwhile with a number, such as add, subtract, multiply or divide it, you must be able to break it into parts and change the parts around. That is why a computer cannot calculate with a number that is expressed as a string. The computer stores numbers in a very different way from strings.

VARIABLES

A variable is a number in a computer's memory. Actually, a variable is a special place or address in memory which you have reserved and given a label. Let's say you give address 13123 the label 'Mice' as part of a program which counts mice going through a maze. The number of mice that have gone through will be kept at that memory address. Each time a mouse goes through, the number will increase. That is, the number kept at the location 'Mice' will vary. This number is called a variable. The computer can use variables to keep track of results during a calculation.

The Central Processing Unit (CPU)

Besides the memory, the other major part inside the computer is the Central Processing Unit, or CPU. The CPU is the "brains"

of the computer. In our home computer, the entire CPU is in a single integrated circuit, called a microprocessor.

The CPU controls everything that a computer system does. Yet, as we said earlier, the CPU does only two things: it moves numbers from one place to another, and it performs arithmetic operations on those numbers. (By arithmetic operations we mean addition, subtraction, multiplication and division.)

However, since everything is represented (or encoded) by numbers inside the computer, the CPU can get a lot done by "just moving numbers" and doing arithmetic.

When you type on the keyboard, the CPU moves the ASCII code numbers into memory, and accepts your input. To give output, it moves more numbers, sending codes to the TV screen or printer. And by moving strings and variables, and combining the arithmetic operations, the computer can carry out commands, play video games, or "talk" through the speech synthesizer.

The parts of the CPU which concern us now are the Arithmetic-Logic Unit (ALU) where the arithmetic operations take place, and the Control Unit.

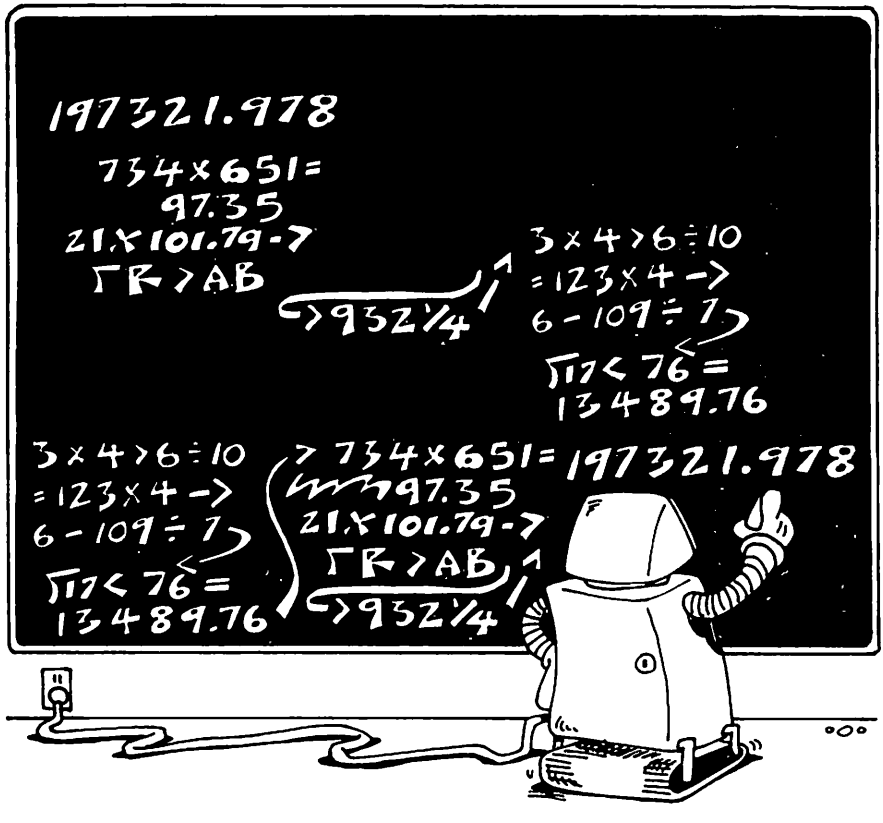
THE CONTROL UNIT

The Control Unit is the part of the CPU that keeps track of what is going on. Everything the computer does is in response to a program. Sometimes you write the program yourself. Sometimes it is stored in a command module or on a disk. The most fundamental things the computer does are controlled by a program built into the ROM. These include things like sending the start-up display to the monitor screen when you turn on the computer or sending a character to the screen when you press a key on the keyboard.

Whatever program is controlling the computer, the Control Unit is "running" it, command by command. For each command, the Control Unit tells the other parts of the computer what to do.

THE ARITHMETIC-LOGIC UNIT (ALU)

The ALU is the part of the computer that performs arithmetic operations. If the control unit wants to add, subtract, multiply or divide two numbers, it sends them to the ALU. The ALU performs the required operation and sends the result back to the control unit.



THINGS YOU CAN DO WITH TI BASIC

Now that you are familiar with the parts of the TI 99/4A computer system, and you know something about how it works and about computer history, it is time to play with it. Professionals in the computer world often speak of playing with computers, even large, powerful business systems. In fact, using a computer can be a lot like play. You can try anything that enters your imagination, and, like a playmate, the computer will respond to whatever you do. Sometimes it will even throw a fit. If that happens, do not be upset. It will all blow over.

So now, let's play. As we proceed, try out the instructions we discuss, and feel free to try any changes you wish. If the computer throws a fit you can always try pressing **FCTN** + or even turning it off, and starting over. The more fun you have playing with your computer, the faster you will learn how to make it obey you.

In this chapter we will see many of the most useful commands you can give the TI 99/4A in the programming language called BASIC. BASIC is one of the easiest to use of all programming languages, and most personal computers can be programmed in some version of BASIC. We will look at four major aspects of TI BASIC: numbers, text, graphics, and music. We will also make the computer talk, using an extended version of BASIC that has some additions to TI BASIC. But first let us take a moment to discuss some miscellaneous but important topics.

ERRORS, INSTRUCTION MODES, FUNCTION KEYS, AND RESERVED WORDS

Before we start looking at TI BASIC instructions, let's look at some things that will help you avoid being frustrated and will

make some of the exercises we do easier.

ERRORS

As you will see shortly, it is very easy to make a slight mistake when typing an instruction to the computer. Leaving out a single character, or putting one in the wrong place, or giving the computer one number when it expects another are just a few of the errors that are easy to make. If you do not make several of these errors, you will be the first person in the history of computing not to do so!

The main thing to remember about errors is that they can always be explained and corrected. That may take some careful reading of the section titled Error Messages in the appendix of your User's Reference Guide. And it will take patience. If you do encounter an error that you cannot figure out yourself, a Texas Instruments Users' Group may exist in your community. Users' Groups are clubs whose members all use a certain make of computer, and they are excellent sources of help and information. A list of TI Users' Groups appears as an appendix in the back of this book.

INSTRUCTION MODES—IMMEDIATE AND DEFERRED

To illustrate the TI BASIC instructions we are going to learn, we will give two kinds of examples: immediate mode and deferred mode. We met these two modes in the last chapter, but it will be useful to review them here.

In immediate mode, you type in an instruction, press ENTER, and the computer does whatever you instructed it to do. This is quick, but it does not let you perform more than one instruction at a time.

Some instructions, however, cannot be used in immediate mode. In such cases we will use the deferred mode, which is just another way of saying that we will use a short program. In

that case, we will type in more than one instruction. Each instruction will begin with a number called a "line number". The computer will not perform any of the instructions until we type RUN and then it will perform all of them, one after another.

THE FUNCTION KEYS

When you make a mistake in a program, you will have to go back and fix the mistake. Since we will be using short programs, it is time to learn about those function keys which are used to alter program lines in TI BASIC. Knowing how to use these keys will save you a lot of time when you want to change or fix a program line. With the function keys you do not have to retype the line, you can simply display the line and then change only the characters that need to be changed.

In the world of computers, when you change something, such as a program, by removing characters, adding characters or substituting one character for another, we say you are "editing".

The keys we will be concerned with here are **FCTN E, FCTN S, FCTN D, FCTN 1, FCTN 2, FCTN 3** and **FCTN 4.** Some of these keys have different uses, depending on what software is running. We are only going to discuss their uses in editing lines in a TI BASIC program.

The FCTN (Function) key is like a "SHIFT" key: you hold it down while pressing another key. For instance, to type FCTN E, hold down the FCTN key and press the 'E' key.

FCTN E This combination of keys is used to display a program line. Type the line number and then **FCTN E.** The line will appear with the cursor over the first character on the line.

FCTN S moves the cursor to the left within a program line.

FCTN D moves the cursor to the right within a program line.

FCTN 1 erases the character under the cursor and closes the space which is left, by moving everything to the right of the cursor one space to the left.

FCTN 2 starts the "insert mode". In insert mode, every key press will insert a character under the cursor and move everything to the right of the cursor one space over. To get out of insert mode, press either **FCTN D** or **FCTN S.**

FCTN 3 erases the line completely.

FCTN 4 erases anything you have added to the line and ends the edit. To see the line, type its number followed by **FCTN E** again.

Let's practice just a little with these keys. Type the following, exactly as shown:

```
10 CALL CLEAR ENTER
20 PRINT "HOW DO YOU DO?" ENTER
30 END ENTER
RUN ENTER
```

This is an example of deferred mode. The CLEAR and PRINT instructions are not executed when you type them, but only later when the program is run. This simple program will make the screen clear and print the phrase:

HOW DO YOU DO?

If this program does not work the way it should, type it over, being careful to put in all the spaces and punctuation marks exactly as shown. This is the long way of correcting a program. We will learn a simpler way now.

Let's change the program. Type:

10 **FCTN E**

The computer will display line 10, with the cursor:

10 ■ALL CLEAR

Use **FCTN D** and **FCTN S** to move the cursor back and forth on this line. Then press **ENTER.**

Now type:

20 **FCTN E**

The computer will display line 20:

20 ■RINT "HOW DO YOU DO?"

Move the cursor to the H and type WHERE ARE YOU, so that the line now looks like this:

20 PRINT "WHERE ARE YOU?"

Now press **ENTER** and run the program by typing:

RUN **ENTER**

See the difference? Next type:

20 **FCTN E**

The computer will display:

20 ■RINT "WHERE ARE YOU?"

Move the cursor over the W and press **FCTN 1** six times. Watch the screen as you do this. The line now looks like this:

20 PRINT "■RE YOU?"

Now press **ENTER** and run the program. Remember that when you run a program the computer executes the instructions in it automatically. To run a program type RUN and then press **ENTER.**

Okay, that does not look complete. Let's use **FCTN E** to edit line 20 again. Move the cursor over the A and type **FCTN 2** followed by HOW F and **ENTER.** Run the program. The result should look like this:

HOW FARE YOU?

Now display the line again. Type anything you want, followed by **FCTN 4.** When you display the line again with **FCTN E,** it will be as it was before you made the last edit. Now type **FCTN 3.** I hope you did not want to keep that line, because you just erased it. Try looking at it again with **FCTN E.**

An even quicker way to erase a line, without using a single function key, is to type the line number and press **ENTER** immediately. Type:

10 **ENTER**
LIST **ENTER**

The computer will display the program with no line 10. (Line 20 was erased by our previous experiment, with FCTN 3.)

Now you should be familiar with the line editing functions of TI BASIC. When we start chapter four, on programming, we will remind you to review line editing. In the meantime, you may want to use these functions in some of the example programs we will use throughout the rest of this chapter.

A summary of these editing functions appears as an appendix in the back of this book.

RESERVED WORDS

When we start writing programs, you will learn that you can make up words to be the names of variables. You can use almost any combination of characters in making up these names, but certain words are off limits. These words are, for the most part, instructions in TI BASIC. When the computer sees these words it thinks that you are giving it an instruction. A list of these words appears in the appendix at the end of this book.

Numbers

Numbers are to computers what blood is to humans. We have already looked at binary numbers and decimal numbers. Now we will see that computers have special ways of writing large numbers.

EXPONENTS

A number multiplied by itself is said to be raised to the second power. A number multiplied by itself three times is said to be raised to the third power, and so on. For example: 2 to the second power is four. 2 to the third power is eight.

2*2=4 2*2*2=8

(Computers use the symbol '*' to mean multiplication.)

When using TI BASIC, we write two to the second power like this:

```
2^2
```

The upside-down "v" is typed by pressing the **SHIFT** key and the 6 key at the same time. Try typing:

PRINT 2^3 **ENTER**

The computer will display the answer like this:

```
PRINT 2^3
 8
```

In this example the the PRINT instruction is used in the immediate mode. As soon as you press **ENTER,** the instruction is executed.

SCIENTIFIC NOTATION

The TI 99/4A, like most small computers, only has so much room in its memory. One result of this is that it cannot display all the digits of numbers over a certain size. Try typing this:

PRINT 9999999999 **ENTER**

The computer will display the number as you typed it. But try this:

PRINT 10000000000 **ENTER**

The computer will respond with:

1.E+10

This is called scientific notation. 1.E + 10 means 1 times 10 to the tenth power, or one followed by ten zeros. 2.38E + 4 would be the number 23,800 or 2.38 times 10000. (10000 is 10 to the fourth power.)

Arithmetic Operations

As we have seen, the arithmetic operations—addition, subtraction, multiplication and division—are at the heart of what a computer does. It is very easy to make the TI 99/4A perform these operations.

ADDITION

Turn on your TI 99/4A and get TI BASIC running. You should see the message and prompt:

```
TI BASIC READY
>
```

Type the following, making sure to leave a space between the T in PRINT and the first 2:

```
PRINT 2 + 2  ENTER
```

The computer will print the answer, and your screen will look like this:

```
PRINT 2 + 2
 4
```

You have just instructed the computer to add two numbers and print the answer.

INCORRECT STATEMENTS

Now try this. Type the following, which is the same instruction that you just typed, but without any space between the T and the first 2.

PRINT2+2 **ENTER**

What happened? You got an error message which looks like this:

INCORRECT STATEMENT

Computers, for all that they can do, have no intuition. They have to have things spelled out for them exactly. The TI 99/4A cannot recognize the end of a word, such as PRINT, unless it is followed by a space. When the space is left out, the TI 99/4A thinks that the term PRINT2+2 is all one word. It does not recognize it, since the BASIC language contains no such word. Therefore, it displays an error message.

Every time you give an instruction to the computer, you must make sure that you type it exactly the right way. The way an instruction is written is called its syntax. If the syntax is correct the computer will recognize it, if not it will stick its tongue out at you, figuratively, and display an error message.

Don't let that bother you. By the time you have finished this book, correct syntax will be second nature to you.

SUBTRACTION

Subtraction, in TI BASIC, works almost the same way as addition. Type the following (don't forget the spaces):

PRINT 4 – 2 **ENTER**

The computer should display the answer so that the screen looks like this:

```
PRINT 4-2
 2
```

Now try this:

PRINT 2-4 **ENTER**

The result should look like this:

```
PRINT  2-4
 -2
```

TI BASIC can tell the difference between positive and negative numbers. As you can see, a negative number is displayed with a minus sign in front of it. It is not apparent, but positive numbers are displayed with a blank space in front of them.

DIVISION

Division is a lot like addition and subtraction. Type this:

PRINT 4/2 **ENTER**

The computer should respond so that the display looks like this:

```
PRINT 4/2
 2
```

Now try this:

PRINT 2/4 **ENTER**

The result should look like this:

```
PRINT  2/4
 .5
```

TI BASIC knows decimal fractions. Next try this:

PRINT 2/Ø **ENTER**

WARNING:
NUMBER TOO BIG
9.99999E+**

Gotcha! You cannot divide by zero; if you try, the computer will give you this friendly warning not to violate the laws of mathematics.

MULTIPLICATION

In TI Basic, we use the * to mean multiplication. Try this:

PRINT 2*4 **ENTER**
8

All right, how about this:

PRINT 1ØØØØØØØØØØ*1ØØØØØØØØØØ **ENTER**
1.E+2Ø

HIERARCHY

A problem arises with arithmetic operations. Say you write:

PRINT 2*3+6/4 **ENTER**

You might mean multiply two and three, add six to the result, and divide that result by four.

2*3=6
6+6=12
12/4=3

Or you might mean add three to six, multiply that by two and divide that by four.

```
3+6=9
2*9=18
18/4=4.5
```

You might even mean multiply two by three and add to that the result of 6 divided by four.

```
2*3=6
6/4=1.5
1.5+6=7.5
```

TI BASIC has a built-in rule to get around this problem. The rule is: perform all multiplications and divisions first, then perform additions and substractions.

```
first     2*3=6
          6/4=1.5

second    1.5+6=7.5
```

The result you would get from the statement you wrote would be 7.5, because the computer would first multiply two and three to get 6, then divide 6 by 4 to get one and a half, then go back and add the two together.

So how can you get the result you wanted, which is three? By using parentheses to tell the computer how you want the operations grouped. All operations inside parentheses are performed before those outside parentheses.

For example, if you wrote your statement like this:

PRINT (2*3+6)/4 **ENTER**

the computer would first multiply two and three, then add six and finally divide the result by four.

first	2*3=6
second	(6+6)=12
third	12/4=3

The answer would be three.

Built-In Numerical Functions

One of the things computers do best is to perform complex mathematical calculations, such as figuring out the square root of an eight digit number. If you were a pretty good mathematician, you could program the TI 99/4A to calculate a square root using the arithmetic operations. But most of us are not such whizzes with numbers, so, since finding a square root is something that might come up fairly often, a program for finding square roots is built into TI BASIC.

In fact, several mathematical operations which might be used often are built into TI BASIC. These are called, reasonably enough, built-in numerical functions. Sometimes, when time is short, they are simply called 'intrinsics'. We are not going to discuss them all, but we will look at these three of them:

SQR	square root
INT	integer
RND	random number

SQR

The square root of a number is a second number which, when multiplied by itself, gives the first number. For example, two is the square root of four (2 times 2 is four), and 10 is the square root of one hundred (10 times 10 is 100.)

Try this:

```
PRINT SQR(100) ENTER
  10

PRINT SQR(25) ENTER
  5
```

Notice the form (we say *syntax*) of the SQR instruction. The number whose square root you are finding must be enclosed in parentheses.

INT

This instruction takes a number that has a decimal point in it, and chops off everything to the right of the decimal point. Notice that this is not the same as rounding a number. Type:

```
PRINT INT(79.09876) ENTER
  79

PRINT INT(1.9999) ENTER
  1
```

RND

The random number instruction is just that—it returns a random number between 0 and .9999999999, inclusive. Type:

PRINT RND **ENTER**

The computer will print some number in the range mentioned.

But, you might ask, what good is that? What if I want a random number bigger than one? The answer is: multiply. For example, say you want a random number between 1 and 6, inclusive, to simulate the roll of a die. Since RND returns a number between

0 and .9999999999, the number it returns multiplied by 6 will return a number between zero and just under six. Now if you add one to that you will have a number between 1 and just under seven. Next, use the INT instruction to lop off everything between six and seven, and you have a number between 1 and six, inclusive. Type:

```
PRINT INT(RND*6 + 1) ENTER
```

The computer will print a number between 1 and 6, inclusive.

Words

If computers could only use numbers, they would not be much fun. Luckily, they can deal with words as well. We have already seen that they do this by encoding each character on the keyboard. We have also learned what a string is: several characters treated as a unit. Let's take that definition a little further. Type:

```
PRINT 2 + 2  ENTER
  4
```

Now type:

```
PRINT "2 + 2"  ENTER
   2 + 2
```

Well, that's interesting. Quotation marks make a big difference. In fact, putting the expression 2+2 inside quotation marks tells the computer to treat it as a *string,* not as two numbers and an arithmetic command.

Text is composed of strings. With TI BASIC you can play with strings in a number of useful ways. Let's look at a few of them.

ASC

Remember that in the last chapter we explained how every character on the keyboard is represented by a code called an ASCII code—the letter A is ASCII code 65, B is ASCII 66, and so on. This instruction tells you (or the program you run) what the ASCII code of a given character is.

The ASCII instruction tells the computer to return the ASCII number of the first character in a string. The string, in quotations is typed in parentheses following the instruction. Type:

```
PRINT ASC("A CAT") ENTER
  65

PRINT  ASC("ZEBRA") ENTER
  90
```

CHR$

The CHARACTER instruction is the reverse of ASC. It returns the character represented by an ASCII number that is typed in parentheses following it. Type:

```
PRINT CHR$(65) ENTER
A
```

LEN

The LENGTH instruction tells you the length of a string. Type:

```
PRINT LEN("A CAT") ENTER
  5
```

The length is five, not four, because the space between A and CAT is inside the quotation marks and counts as one character.

SEG$

The STRING SEGMENT instruction returns a portion of a string, that is, a substring. A substring is a string that is part of another string. For example, "am" is a substring of "ex*am*ple".

What substring is returned by SEG$ depends on the arguments you give the instruction. You always give it three arguments: the string with which you start, the position within that string of the first character of the substring you want and the length of the substring. Type:

```
PRINT SEG$("A BIG, OLD, ORNE
RY, STRIPED CAT", 21, 11) ENTER
STRIPED CAT
```

The substring, STRIPED CAT, starts at the 21st character within the string, A BIG, OLD, ORNERY, STRIPED CAT. (Remember, commas are characters too.) The substring is eleven characters long.

POS

The POSITION instruction returns the location within a string of a substring. Type:

```
PRINT POS("A BIG, OLD, ORNE
RY, STRIPED CAT","STRIPED", 1) ENTER
 21
```

Within the string A BIG, OLD, ORNERY, STRIPED CAT, the substring STRIPED begins at the 21st character. The

arguments are the starting string, the substring and the position within the starting string at which we wish to begin looking for the substring. In this case we began at the first letter.

VAL

The VALUE instruction looks at a string and returns a number that is the value of the string. The characters in the string must be those of a number, and they must be in quotes. Type:

```
PRINT VAL("462")
  462

PRINT VAL("9.834")
  9.834
```

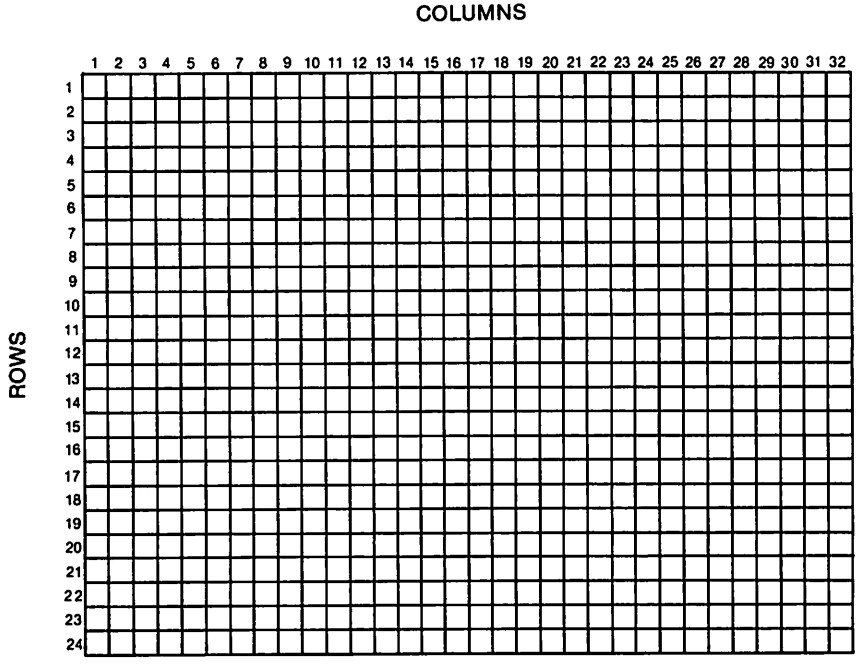
It may be difficult to understand why such an instruction is necessary. As we will see in the next chapter, sometimes data exists in the form of a string, but you want the computer to perform arithmetic operations on it. For that to happen, the string must first be translated into a number.

Graphics

Graphics is a word that is used in the computer world to mean pictures. The TI 99/4A can draw pictures, color them almost any way you want and even make them move. In the following section we will meet some of the instructions used to make graphics. In the next chapter, on programming, we will use these graphics capabilities in a BASIC program.

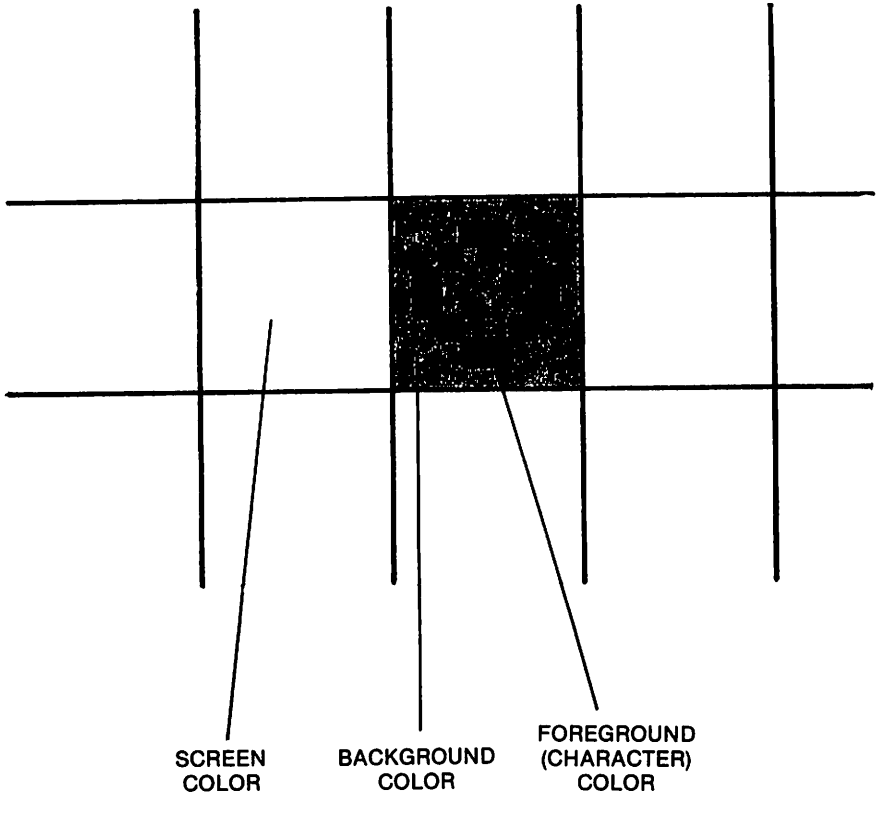
In TI BASIC, graphics are built out of characters—not just the characters on the keyboard, but characters to which you can give any shape you like. You might, for instance, create several

characters so that when they are displayed next to each other, they form a spaceship. Each character takes up one location on the screen. Remember that the screen is divided into 32 vertical columns and 24 horizontal rows.



Colors

The TI 99/4A can display 13 different colors, plus black, white and transparent. Unless a character is a solid square, it will not fill an entire location on the screen. Therefore, the character might be one color, the parts of the location it does not fill might be another color and the screen could be a third color.



Normally, the foreground color of a character is black, the background color is transparent, and the screen color is light blue.

These colors may be changed using two subprograms. (If you do not remember what a subprogram is, refresh your memory by re-reading the section in chapter one titled *Using the Display.*) These two subprograms are called COLOR and SCREEN. COLOR and SCREEN can be demonstrated using simple programs.

SCREEN

Type this:

```
10 CALL CLEAR ENTER
20 CALL SCREEN(2) ENTER
30 FOR J = 1 TO 500 ENTER
40 NEXT J ENTER
50 END ENTER
RUN ENTER
```

The screen will go black for a couple of seconds. (If something else happens, make sure you have typed exactly what is shown. Remember how picky computers are about correct commands.)

This little program uses the SCREEN subprogram to change screen color. The SCREEN subprogram only works while a program is running, so we have added lines 30 and 40 to make the program run long enough to notice the color change. We will learn the instructions used in lines 30 and 40 in the next chapter.

Run this program a few more times, changing line 20 each time so that each time you run it, the argument of the SCREEN subprogram is a different number between 2 and 16.

COLOR

This subprogram changes the color of a character and the color of its location. The character's color is called the foreground color and the color of the location is called the background color. Using the above program which we wrote to demonstrate SCREEN, change line 20 and add line 25 so that the program looks like the one below, and then type RUN, as shown.

(To change line 20 use the function keys for editing, as we discussed at the beginning of this chapter. Type 20, then

FCTN E. The line should be displayed with the cursor on it. Make the change, then press **ENTER.**)

You can change line 20 and add line 25, without disturbing the rest of the program. The TI 99/4A will take care of putting the statements in order according to the line numbers! You do not have to retype lines 30 to 50.

```
10 CALL CLEAR ENTER
20 CALL COLOR(5,16,7) ENTER
25 PRINT "ABCDE" ENTER
30 FOR J=1 TO 500 ENTER
40 NEXT J ENTER
50 END ENTER
RUN ENTER
```

The computer will print the string ABCDE in white letters on a red background over a green screen for a couple of seconds. The COLOR subprogram has three arguments. The second two arguments are the foreground color and the background color, respectively.

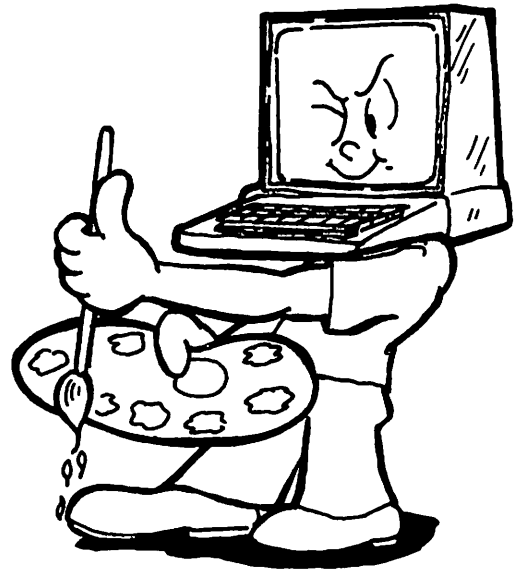
The first argument is the number of an ASCII group. Each ASCII code belongs to one of these groups. There are sixteen of these groups. The letters A through E (ASCII 65-69) belong to group 5. A list of these groups appears below and in an appendix at the end of this book.

ASCII CODE	GROUP NUMBER	ASCII CODE	GROUP NUMBER
32-39	1	96-103	9
40-47	2	104-111	10
48-55	3	112-119	11
56-63	4	120-127	12
64-71	5	128-135	13
72-79	6	136-143	14
80-87	7	144-151	15
88-95	8	152-159	16

When the program you have typed is done, the colors will return to their normal condition: black on a transparent background over a blue screen. Color changes only last while the program is running.

Try different color combinations by changing the last two arguments of the COLOR subprogram. A list of available colors and their numbers appears below and in an appendix at the end of this book.

COLOR CODE	COLOR
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Gray
16	White



As you use this color chart, bear in mind that the colors you see on the screen are affected by how the color controls on your television or monitor are adjusted. If you do not get the color you expect, try adjusting them differently.

Placing Characters Anywhere On The Screen

HCHAR

As you may have noticed, the PRINT instruction only displays data at the bottom of the screen. The subprogram, HCHAR, and its sister, VCHAR, let you display data anywhere you want. Type this:

```
CALL CLEAR ENTER
CALL HCHAR(12,16,65,10) ENTER
```

The computer will print a line of ten A's, starting in the center of the screen. The four arguments we gave HCHAR are as follows: The first two numbers are the row and then the column on the screen of the first character to be printed. In this example we ask for the twelfth row down, sixteen spaces over. The third argument is the ASCII code of the character, in this case, 65 for A. The last argument is the number of times we want the character repeated. We could leave this last argument out, in which case the character would be printed just once.

VCHAR

VCHAR is just like HCHAR, except that it prints vertical rows of characters. Type this:

```
CALL CLEAR ENTER
CALL VCHAR(12,16,65,10) ENTER
```

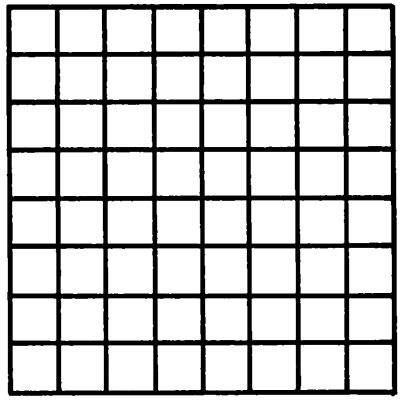
You should see a column of ten A's, starting in the middle of the screen and running downward.

Character Definition

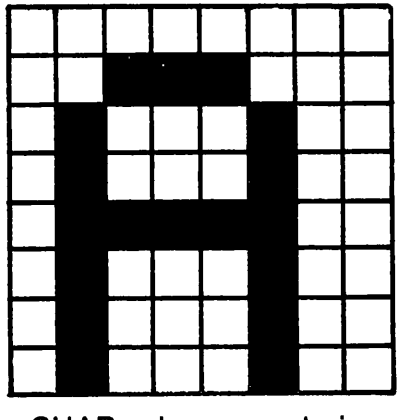
We mentioned earlier that you may create your own

characters, for instance, to form a spaceship. This is done with the CHAR subprogram.

Every character in TI BASIC is made up of tiny squares. Imagine a tiny checkerboard, eight squares on a side.



The letter A (ASCII 65) is formed like this:

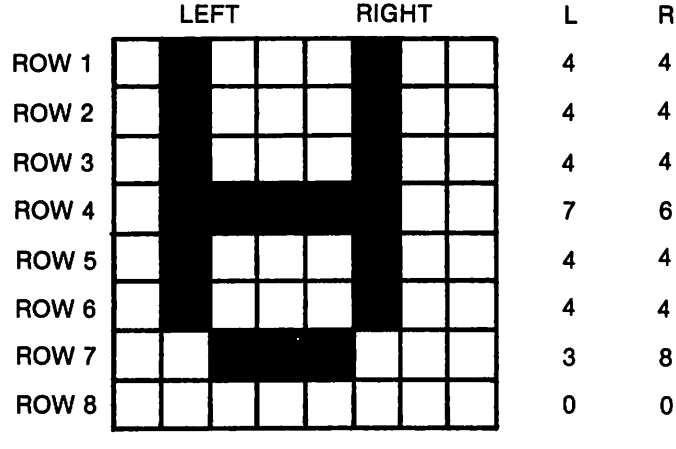


We can use the CHAR subprogram to invert the letter A. Let's give our upside-down A the ASCII code 128. Here is how we would do it. Type the following program, being sure that all letters are in upper case (from here on out we will assume that you will remember to press the ENTER key after each line and after the run command without being reminded):

```
10 CALL CLEAR
20 CALL CHAR(128,"4444447C44443800")
30 CALL HCHAR(12,16,65,1)
40 CALL HCHAR(11,16,128,1)
50 CALL HCHAR(12,15,128,1)
60 CALL HCHAR(12,17,128,1)
70 CALL HCHAR(13,16,128,1)
80 FOR J=1 TO 2500
90 NEXT J
RUN
```

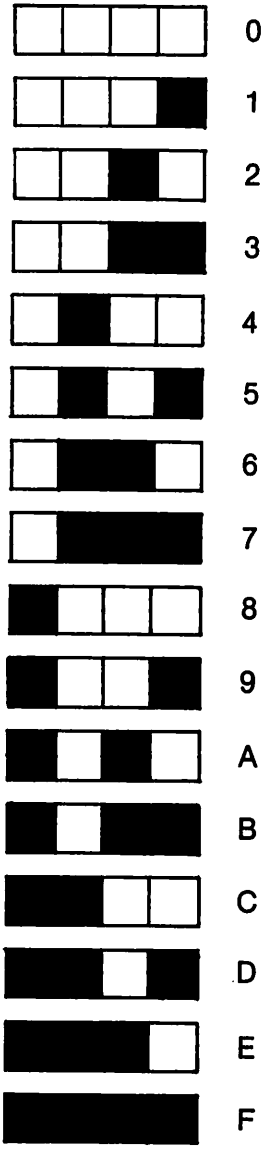
The computer will print an A in the center of the screen and surround it with inverted As. The first A is our old friend, ASCII 65. The other As are all ASCII 128, which we have created ourselves. Wow! How did we do that?

Here's how. Let's go back to our 8 by 8 checkerboard.



Notice that the checkerboard is divided into 8 horizontal rows, and each row is divided into a left side and a right side. So the character area is divided into sixteen separate parts, each composed of a horizontal row of four boxes. Some of these boxes are dark, and form the shape of the inverted letter A. The

remaining boxes are light. When we define a character with the CHAR subprogram we are telling the computer which boxes are to be dark and which light. We do this according to a kind of code which we will call our "character definition code".



These sixteen numbers represent all the possible combinations of black and white boxes in a four-box row. The numbers used are 0 through 9 and A through F. These are the digits of a numbering system based on sixteen. This sixteen-base system is called hexadecimal, or hex, for short, and it is used often in the world of computing. Don't worry if it puzzles you; just remember that in hex, A is ten, B is eleven, C is twelve, D is thirteen, E is fourteen and F is fifteen.

Now look back at the inverted A, as we divided it up into eight rows. Let's look at the left half of row one. Only the second square is black, so it would be the same as number 4 in our character definition code. The right side of row one would also be a 4. Both sides of rows two and three would also be 4s, since in each of them, only the second from the left box is dark. The left side of row four would be a 7, since the three right-most boxes are dark. The right side of row four would be a C, since the two left-most boxes are dark. If you continue assigning code numbers to the inverted A, going from left to right and top to bottom, you would wind up with the following sequence: 4444447C44443800. Does this look familiar?

Right! Back to our little program. In line 20 we gave the CHAR subprogram two arguments. The first was 128, the ASCII code to which we are assigning a new character. The second argument was a string (in quotation marks) composed of the 16 numbers we just created: 4444447C44443800. The rest of the program simply placed an old ASCII 65 A in the middle of the screen and then surrounded it with the new ASCII 128 A's, and then kept the program running for about six seconds.

Try changing the second argument of the CHAR subprogram, and see what new characters you can create.

Sound

TONES

In addition to handling numbers, text and graphics, the TI 99/4A can also produce music. The basic unit of music, of course, is the single tone. TI BASIC creates a tone with the SOUND subprogram.

Make sure that the volume control on your television or monitor is turned up, and type this:

```
CALL SOUND(1000,300,15)
```

The computer plays a note through the loudspeaker of your television or monitor. The first argument you gave the SOUND subprogram determined the duration of the tone, the second controlled the frequency of the tone, and the third gave the loudness of the tone.

DURATION

Duration is how long the tone lasts. The duration argument may be any number between 1 and 4250, inclusive. A duration of 1 would last .001 second, and a duration of 4250 would last 4.25 seconds.

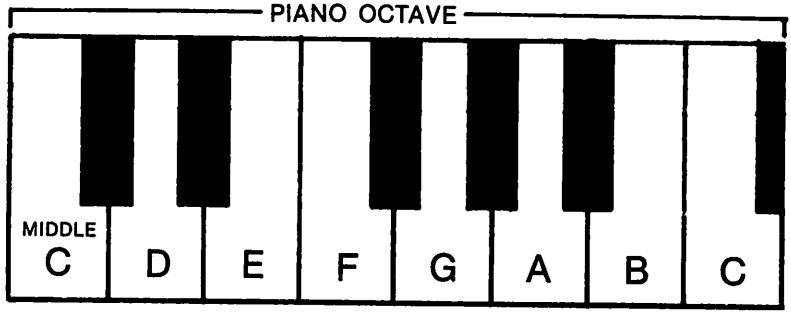
You may also give a duration argument of -1 to -4250 inclusive. In a single SOUND subprogram, such as we typed above, a minus duration would sound the same as a positive duration. However, if you follow one SOUND subprogram with another one in a program, a negative duration in one will cut off the duration of the previous one. For example, type:

```
10 CALL SOUND(4000,300,15)
20 CALL SOUND(-500,600,15)
RUN
```

The first tone will be cut off almost immediately.

FREQUENCY

Frequency determines the pitch of a tone, whether it is high or low. The lowest frequency argument is 110 and the highest is 44733, which is way above what a human can hear, although it may make your dog smile and wag its tail. For those of you who read music, a frequency of 220 is A below middle C, and 110 is an octave below that; a frequency of 880 is A above high C and 1760 is an octave above that. For other musical equivalents, see the appendix in the back of this book.



SOUND	262	294	330	349	392	440	494	523
FREQUENCY	277	311		370	415	466		554

Type this:

```
CALL SOUND(1000,110,15)
```

Listen to the tone this produces. Then try the same subprogram substituting different frequencies for 110.

The frequency argument may also be in another range: -1 to -8 inclusive. -1 through -4 produce what Texas Instruments calls

"periodic noise", which sounds like a robot trying to hum. Frequencies of the -1 through -3 noises are fixed, but the frequency of the -4 noise varies, as we will see in the section on harmony, below.

Frequencies in the range -5 through -8 produce what Texas Instruments calls "white noise", which sounds something like radio static. Noises -5 through -7 are fixed, and noise -8 varies, as we will see. The best way to become familiar with these noises is to play with them. Call the SOUND subprogram a few times, each time using one of the minus frequencies.

Remember that the duration must be long enough to hear the tone and that the three arguments must all be present, inside parentheses, separated by commas.

LOUDNESS

Loudness is the third argument in the SOUND subprogram. The value of 0 for this argument gives the loudest tone and the value of 30 gives the quietest. Of course, the loudness is also affected by the setting of the volume control on your television or monitor. Type:

```
CALL SOUND(1000,300,4)

CALL SOUND(1000,300,13)
```

Try using the SOUND subprogram with various different loudness arguments.

Tunes

A tune is simply a number of tones played one after the other. Let's see how you can play a tune on the TI 99/4A. Let's be patriotic today, and try the first few notes of the Star Spangled

Banner. Type this:

```
10 CALL SOUND(500,300,15)
20 CALL SOUND(500,250,15)
30 CALL SOUND(500,200,15)
40 CALL SOUND(500,250,15)
50 CALL SOUND(500,300,15)
60 CALL SOUND(500,400,15)
RUN
```

Play ball! Whoops, I thought for a moment...boy, that was realistic, wasn't it? Now, see if you can finish the whole national anthem. (You don't have to stand up while you play it unless you want to.) Or, if that seems a tedious assignment, let's get fancy and move on to...

Harmony

TI BASIC can play not only one note, but three notes at a time, and three notes plus one noise. This gives you almost endless possibilities for combining different notes. To see how this works, type this:

```
CALL SOUND(2000,300,0,500,15,700,15)
```

The computer plays a chord, the lowest tone of which is louder than the other two. The first argument is still the duration, and it will be the same for all three notes in the chord. The second, fourth and sixth arguments are the three different frequencies. The third, fifth and seventh arguments are the loudnesses for each frequency. You may even add a noise by including a fourth argument with a minus frequency. Type:

```
CALL SOUND(2000,300,0,500,15,700,15,-4,15)
```

Here we used the -4 noise. Although it is hard to tell, the sound of this noise varies depending on the frequency of the third

tone in the chord. The same is true of the -8 noise.

It may be difficult to tell the difference between different versions of the SOUND subprogram, especially if they are not played one right after the other. Play around with different combinations and try writing short programs like the one we used for the Star Spangled Banner, so that you can hear different harmonies played one after the other.

Speech

This section on speech takes you beyond TI BASIC. TI BASIC by itself has no speech capabilities. However, a number of command modules add speech to TI BASIC. One of these is the TI EXTENDED BASIC module, which adds several capabilities to what TI BASIC can do, including speech. Of course, to make your TI 99/4A talk, you also have to have the TI Speech Synthesizer.

If you have the TI EXTENDED BASIC command module and the speech synthesizer, you can make the computer speak any of 366 words, in any combination. Furthermore, you can add the suffixes *ing, ed* and *s* to any of these words. For some reason, this vocabulary leaves out some of the most important words in the English language, such as love, money, food and happy, to name a few. Oh well, we will just have to make do.

The instruction that makes the TI 99/4A talk is the SAY subprogram. Make sure the Speech Synthesizer is plugged in; get out of TI BASIC by pressing **FCTN +**; if necessary, turn off the computer, insert the TI EXTENDED BASIC command module, turn on the computer and select TI EXTENDED BASIC from the start-up menu. Then type:

```
CALL SAY("I WORK WHEN I CAN")
```

In a voice like a slightly hoarse, but friendly robot, the computer will tell you that it works when it can. Computers are so humble.

As you can see, the argument of the SAY subprogram is a string of words selected from the built-in vocabulary. This vocabulary consists of some 366 words, as mentioned above. The string of words for the computer to pronounce must be enclosed in quotes, inside parentheses.

Conclusion

In this chapter we looked at some of the instructions in TI BASIC. With these instructions you can make your TI 99/4A do some very impressive things: play music, speak, produce a rainbow of colors. In the next chapter we will look at the workhorse instructions of TI BASIC. These instructions work behind the scenes to control the effects which we learned how to make in this chapter.

PROGRAMMING IN TI BASIC

Before we start learning how to write programs, go back and review the sections at the beginning of Chapter III, titled ERRORS, INSTRUCTION MODES, FUNCTION KEYS and RESERVED WORDS.

A Few Words About Program Lines

By now you know what a program written in TI BASIC looks like. It is composed of a set of lines. Each line begins with a line number and consists of instructions and their arguments. These instructions and arguments look a little like English words, but you cannot read them as English.

REM

We are going to spend the rest of this book writing programs in TI BASIC. By the time we are finished we will have written quite a few lines of BASIC. A few months from now, we might look at these lines and not remember what they mean. It would be even harder for someone else to understand them without spending a lot of time studying them. We use the REM instruction to make it easier for a human to understand a program.

REM is short for "Remark". REM statements are messages that you add to your programs. These messages explain the programs in which they appear. They do not affect the computer; they are for the benefit of people reading the program listing.

The REM instruction must go at the beginning of a line. It tells the computer to ignore everything that follows on that line.

Here is how the REM instruction works. Type:

```
10 REM A SHORT DEMO PROGRAM
20 PRINT "THIS IS A PROGRAM"
30 PRINT "WRITTEN IN TI BASIC"
40 END
RUN
```

This is a demonstration. In line 10 we use a REM to remind anyone reading the program that it is a "short demo". When you run the program, line 10 does not cause anything to happen because it begins with a REM.

Memory Use

A computer can store information in its memory. We will discuss several kinds of information that the computer can "remember", including numbers, strings, numeric variables, string variables, and arrays.

Numeric Variables

A numeric variable is a place in the computer's memory. Think of it as a box. It contains a number, but that number may change from time to time. This box has a name, called a variable name.

The name of a numeric variable may be up to 15 characters long. The name may include any letter or any number, but it must start with a letter.

Examples of names for numeric variables would be:

P PEOPLE PPL PI

For example, say we are writing a program to keep track of people going in and out of an elevator. We may use the name PEOPLE to stand for the number of people in the elevator at a given time. As the number of people in the elevator changed, the number stored in the variable PEOPLE would have to be changed.

LET

In TI BASIC a number is put into a numeric variable by using the LET instruction. Type:

```
LET A = 4
```

This will put a 4 into the variable named A.

```
PRINT A
```

The computer will print the number 4, which is the number that is stored in the variable A.

```
10 REM DEMO LET STATEMENT
20 CALL CLEAR
30 LET A = 4
40 LET B = 5
50 PRINT A * B
60 LET B = 6
70 PRINT A * B
80 END
RUN
```

The computer will print:

```
20
24
```

In this program we start off with a REM instruction in line 10, so that we have an explanation of what the program does. In line 20 we use the CLEAR subprogram to clear the screen and avoid confusion. In lines 30 and 40 we put the numbers 4 and 5 into variables. The names of these variables are single letters: A and B.

In line 50 we multiply the number in A by the number in B and print the result. Remember that the asterisk is the BASIC sign for multiplication.

In line 60 we change the number in variable B. In line 70 we again print the result of B times A, and of course, we now get a different answer, since there is a different number in B.

In line 80 we end the program with an END instruction. It is a good idea to get into the habit of ending all programs with an END instruction.

TIP! The LET instruction consists of the word LET, a variable name, an = sign, and a number:

```
LET A = 4
```

However, in this instruction, the LET is not necessary. You can leave it out if you wish:

```
A = 4
```

The computer assigns a value to the numeric variable whenever it sees a variable name, an = sign and a number. Try changing the program with the FCTN keys so that the word LET is eliminated. Then run the program. It should work exactly as before.

Strings

A word, a sentence, or any other group of characters is called a string. For example, type:

```
PRINT "HIYA FELLA"
```

The computer will display the string:

```
HIYA FELLA
```

The group of characters, HIYA FELLA, is a string.

String Variables

Just as numbers may be stored in numeric variables, strings may be stored in string variables. A string variable is a space in memory that can hold a string. The name of a string variable stands for that place in memory. A string is stored in that space by using the LET instruction (or just an = sign).

The following commands all store strings into string variables:

```
LET X$ = "NOW IS THE TIME"
LET PHONE$ = "BEACHWOOD 4-5769"
ADDR$ = "328 ELM ST."
```

The name of a string variable must begin with a letter, may be up to fifteen characters long, and may contain any letters or numbers. It *must* end with a $. The $ tells the computer that this is a string variable and not a numeric variable. When assigning a string into a string variable, the string must be in quotation marks.

Type:

```
10 REM TEST STRING VARIABLES
20 CALL CLEAR
30 LET A$ = "HIYA FELLA. WHAT'S HAPPENING?"
40 LET B$ = "GOOD DAY, SIR. I FEAR RAIN."
50 PRINT A$
55 PRINT B$
60 LET B$ = "A POX ON ALL MISANTHROPES."
70 PRINT
80 PRINT A$
85 PRINT B$
90 END
RUN
```

The computer will print:

```
HIYA FELLA. WHAT'S HAPPENING?
GOOD DAY, SIR. I FEAR RAIN.

HIYA FELLA. WHAT'S HAPPENING?
A POX ON ALL MISANTHROPES.
```

Notice our use of the PRINT instruction all by itself in line 70. This causes the computer to print a blank line, skipping a line.

We have now learned what strings and string variables are. Strings are groups of characters. String variables are spaces in the computer's memory that can hold groups of characters, like words or sentences. The name of a string variable will stand for whatever string is stored in it.

Arrays

A numeric or string variable is a single box in the computer's memory. An array is a group of these boxes. It may be a row of boxes or a checkerboard of horizontal and vertical boxes.

An array is a list of numbers or strings. A list has more than one item in it. Therefore, when you want to get a particular number or string out of an array, you have to tell the computer both the name of the array and which item on the list you want. With an array, this is done by putting the item number in parentheses after the array's name.

For example, say you want to keep track of the heights of students in a class of six. If students in the class were represented by the array name S, then the height of student number one would be S(1), the height of student number two would be S(2), and so on. The height of the sixth student would be S(6). Each of these would be a member of the array S.

ARRAY OF STUDENTS' HEIGHTS (IN INCHES)

S(1)	S(2)	S(3)	S(4)	S(5)	S(6)
58	61	65	56	61	68

Like variable names, array names must begin with a letter. The name may be up to fifteen characters long, and may consist of any letters or numbers. If an array name is the name of a string array it must also end in a $.

An array may be a list of strings instead of numbers. In that case each element in the array would be a different string. For example, each element in our array might contain a student's name, instead of his or her height. But a name is a string, not a number, so the array name would have to be S$, and the elements would be S$(1), S$(2), etc.

ARRAY OF STUDENTS' NAMES

S$(1)	S$(2)	S$(3)	S$(4)	S$(5)	S$(6)
JOE	SUE	BILL	JIM	MARY	JANE

DIM

The DIM instruction sets aside space in the computer's memory for an array. The DIM instruction must be used only once in a program for each array in that program—once an array is DIMensioned in a program, it may not be reDIMensioned.

To see how to use DIM, let's use a program that adds together the heights of all the students in our class. Type this:

```
10 REM TEST DIM INSTRUCTION
20 CALL CLEAR
30 DIM N(6)
40 S(1) = 58
50 S(2) = 61
60 S(3) = 65
70 S(4) = 56
80 S(5) = 61
90 S(6) = 68
100 PRINT S(1) + S(2) + S(3) + S(4) + S(5) + S(6)
110 END
RUN
```

In this program, line 30 reserves six places in array N. Lines 40 through 90 put numbers into the six places in the list. Finally, line 100 adds these numbers up and prints the sum.

The computer will display:

369

Conclusion

In this section on memory use we have learned about numeric variables, string variables and arrays. Each of these is a space in the computer's memory. The numeric variable holds a number. The string variable holds a group of characters. The array holds a list of either numbers or strings.

Numeric variables, string variables and arrays all are given names. These names may be up to fifteen characters long and must start with a letter. The name of a string variable or a string array must end in a $.

INPUT/OUTPUT

One of the most valuable features of a computer is that you can communicate with it. It can talk to you, and you can talk to it. When you talk to the computer it is called input, and when the computer talks to you it is called output.

TI BASIC uses several instructions to receive and to send information.

PRINT

You are already familiar with the PRINT instruction. You know that it displays numbers or text on the screen. What it displays is a form of output, since it is information being given *to* you. PRINT is the most important output instruction in TI BASIC.

We have not yet looked at how the PRINT instruction works when it is printing more than one item at a time. For example, you might want to print two string variables with one PRINT statement. The PRINT instruction would put both strings on one line by using one of three characters called print-separators. The characters are the semi-colon, the colon and the comma.

The semi-colon is used to link print elements together. Type:

```
PRINT "HIYA FELLA";"HIYA YOU
  RSELF."
```

The computer will display:

```
HIYA FELLAHIYA YOURSELF
```

Whoops, the two sentences are a little too close together. We'll fix that next time. For now, it shows how the semi-colon works. Items that are to be printed, such as these two strings, are called print elements. The string HIYA FELLA is the first print element, and the string HIYA YOURSELF is the second print element. We could have more print elements if we wanted. For example, we could have a PRINT statement that looked like this:

```
PRINT "HI";"THERE";"HOW";"ARE";"YOU"
```

TIP! If the combined length of print elements is greater than twenty-eight characters, the print element that takes the total over twenty-eight will be printed on the next line, semi-colon or not.

The colon is used to print an element on the next line every time. Type:

```
PRINT "HIYA FELLA  ";"HIYA YO
    URSELF":"YUP"
```

The computer will display:

```
HIYA FELLA HIYA YOURSELF
YUP
```

See the difference?

An Aside

Remember a moment ago we said that we would fix the problem of words FELLA and HIYA running together? In our last example we did that simply by adding a space after FELLA and before the following quotation mark. The space becomes the last character in the string, HIYA FELLA .

Back To PRINT

To understand how the comma print-separator works you must know something about how the TI 99/4A organizes its display. The screen is divided into a right half and a left half, each 14 columns wide. The comma causes print elements to be printed on alternating halves of the screen. Type:

```
PRINT "A-ONE","AND A-TWO","A
    ND-A THREE","AND A-FOUR"
```

If you are not a Lawrence Welk fan, try the same example with different print elements.

The three print separators—semi-colon, colon, and comma—work the same way with numbers, numeric variables and string variables as they do with strings. Numbers will not be run together with the semi-colon, however, because they are always printed with a space following them, and they always have either a minus sign or a space preceding them. Type:

```
PRINT "A";1;"B";2;"C";"D"
```

The computer will display:

```
A 1 B 2 CD
```

INPUT

INPUT causes the program to wait until you have typed something at the keyboard and pressed the ENTER key.

Type:

```
10 REM TEST 'INPUT'
20 CALL CLEAR
30 PRINT "PLEASE TYPE YOUR NAME"
35 INPUT NAME$
40 CALL CLEAR
50 PRINT "HI THERE  ";NAME$
60 END
RUN
```

The computer will invite you to type your name and will display a question mark, although on some television screens you may not be able to see the question mark. Type your name, then press ENTER. The computer will erase the screen and then display whatever you typed.

The string variable NAME$ now stores what you typed. You can check that by using the immediate mode to print NAME$.

Type:

```
PRINT NAME$
```

You are already familiar with the REM statement and the CALL CLEAR subprogram we use in lines 10, 20 and 40. Line 30 asks you to type your name. Line 35 is the INPUT instruction that stores what you type in the string variable NAME$. Line 50 prints the string stored in NAME$. Line 60 ends the program.

If you wanted, you could combine 30 and 35. The INPUT instruction lets us give the user a message. The message must follow the word INPUT and be in quotes. It must be followed by a colon which is followed by the string variable. If we used this form of INPUT our program would work exactly the same way and would look like this:

```
10 REM TEST 'INPUT'
20 CALL CLEAR
30 INPUT "PLEASE TYPE YOUR N
   AME":NAME$
40 CALL CLEAR
50 PRINT "HI THERE ";NAME$
60 END
```

You do not have to use any message with INPUT. If you don't, you simply follow the INPUT instruction with a variable. Change the program so that line 30 looks like this, then run it:

```
30 INPUT NAME$
```

The INPUT instruction may return information into a string variable, as we have done, or into a numeric variable. INPUT also may return information into a combination of variables. If more than one variable is used, each variable must be separated by a comma from the one following. Change lines 30 and 50 so that they look like this, and run the program: (Notice that we break the lines so that they look as they will when you type them on the TI 99/4A. Be aware that neither line 30 nor line 50 ends until you finish typing PHONENUMBER, so don't press ENTER until then.

```
30 INPUT FIRSTNAME$,LASTNAME
   $, AGE,PHONENUMBER

50 PRINT "HI THERE "; FIRSTN
   AME$;" ";LAST NAME$;" WHO IS
     ";AGE;" YEARS OLD AND WHOSE
   NUMBER IS ";PHONENUMBER
```

Try running the program now.

Note that in line 50 we again insert a space between two strings (the first and last names) so that they will not be run together when the computer prints them out.

When a single INPUT instruction asks for several variables you must separate them with commas. You must also make sure that you input a string when the computer expects a string and a number when it expects a number.

KEY

The KEY subprogram is used to send a single character to the computer. Its advantage is speed: you do not have to use the **ENTER** key. Its disadvantage is that you can input only one character.

The KEY program has three arguments: a number, a return variable and a status variable. You may decide what you want to name the variables. When you use the KEY subprogram, a code that represents the pressed key is stored in the "return variable". In other words, the return variable is used to store a key that you type. Type:

```
10 REM DEMO KEY SUBPROGRAM
20 CALL CLEAR
30 PRINT "PLEASE PRESS A KEY "
40 CALL KEY(3,RETVAR,STATUS)
50 PRINT RETVAR, STATUS
60 IF STATUS = 0 THEN 40
70 END
```

Note that the reason we name the return variable RETVAR instead of RETURN is that RETURN is a reserved word and cannot be used as the name of a variable. We discussed reserved words in the previous chapter. We name the status variable simply STATUS.

When you run this little program the computer, in obediance to line 30, prints an invitation to press a key. Then it executes the next lines, lines 40 and 50. This all happens in a tiny fraction of a second. You have not had time to press any key yet, so the

value of RETVAR, when it is printed at line 50, is -1 and the value of STATUS is 0, meaning that no key has been pressed. Since the value of STATUS is 0, at line 60 the program goes back to line 40 and repeats. This is called a loop, and we will see more of loops before long. The program will repeat a loop or two before you can press a key. Note that each time a loop repeats, the KEY subprogram is executed.

When you finally press a key, line 50 prints the code for that key—the value of RETVAR—as well as a new value for STATUS. STATUS is no longer 0 so line 60 does not send you back to line 40, and the program ends. Normally the code stored in the return variable is the ASCII code for the character whose key you pressed. Try running the program a few times, pressing different keys, including function and control keys.

As we have seen, the status variable (which we have named STATUS) stores a zero if no key is pressed. When a new key is pressed, STATUS stores a one.

TIP! A common use for the KEY subprogram is simply to cause a program to pause, for instance, while the user reads a display. He may read for as long as he likes, while the program waits for him to press a key. Then, at his own time, he may press a key and continue. To use KEY this way you would leave out line 50. We will be using KEY as a program pause in the next chapter.

We still have not discussed the first argument of the KEY subprogram, which is not a variable but a number between 0 and 5, inclusive. This argument is called a "key unit".

From now on, we will use the key unit 3, in which the computer ignores control codes. This is the best key unit for beginners in TI BASIC. The other key units have special purposes.

Key units one and two are used with the joysticks. They are used to take advantage of the fire button on the joysticks.

When the button is pressed, the KEY subprogram will return the number 18 in the return variable if the key unit is a one or a two. See the section below on the JOYST subprogram.

Key units 0, 4 and 5 are for advanced uses. When you have gone beyond the beginner stage you may want to find out more about them.

CONTROL INSTRUCTIONS

Control instructions are used to control the path that a program takes. Left alone, TI BASIC will start at the first line of a program, then obey each line in order until it reaches the end. Control instructions are used to alter that order, so that a program may jump from one line to a line above it or below it.

TI BASIC uses four major control instructions: GOTO, GOSUB-RETURN, IF-THEN-ELSE and FOR-TO-STEP-NEXT.

GOTO

GOTO Is the simplest control instruction. It tells the program not to execute the next line, but to jump instead to some other line; that is, to GO TO another line. The number of that line is the argument of the GOTO instruction. The line number, and only the line number, must follow the GOTO instruction, on the same line.

First, here is a program with no GOTO in it. Type:

```
10 REM DEMO GOTO
20 CALL CLEAR
30 PRINT "AAAAA"
40 PRINT "BBBBB"
50 PRINT "CCCCC"
60 PRINT "DDDDD"
70 PRINT "EEEEE"
80 END
RUN
```

The computer will display:

```
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE

DONE
```

Now type a line 35 so that the program looks like this:

```
10 REM DEMO GOTO
20 CALL CLEAR
30 PRINT "AAAAA"
35 GOTO 70
40 PRINT "BBBBB"
50 PRINT "CCCCC"
60 PRINT "DDDDD"
70 PRINT "EEEEE"
80 END
RUN
```

The computer will display:

```
AAAAA
EEEEE
```

Now try this, erasing line 35 and typing a line 75 like this:

```
10 REM TEST GOTO
20 CALL CLEAR
30 PRINT "AAAAA"
40 PRINT "BBBBB"
50 PRINT "CCCCC"
60 PRINT "DDDDD"
70 PRINT "EEEEE"
75 GOTO 30
80 END
RUN
```

The computer will type lines of letters forever, because whenever it comes to line 75, that line directs it back to line 30, where it starts all over, forever. To stop it press FCTN 4 .

Loops

This kind of repetition is called a loop. In this case, since there is no way of escaping from the loop, short of stopping the whole program, it is called an endless loop.

How could we have escaped, other than by pressing FCTN 4 ? We'll see that in the next chapter, so stay tuned.

GOSUB-RETURN

A "subroutine" is a part of a larger program. The lines in a subroutine do some special job. Each time that job needs to be done, the program jumps to the first line of the subroutine and obeys the lines in the subroutine. When the program has finished with the subroutine, it jumps back to the line it came from.

This is done with the GOSUB instruction. GOSUB sends the program to a subroutine, and RETURN sends it back. Every subroutine must end with a RETURN instruction.

Type:

```
10 REM GOSUB-RETURN
20 CALL CLEAR
30 S$ = "A"
40 GOSUB 1000
50 S$ = "B"
60 GOSUB 1000
70 S$ = "C"
```

```
80 GOSUB 1000
90 S$ = "D"
100 GOSUB 1000
110 S$ = "E"
120 GOSUB 1000
130 END
1000 REM SUBPROGRAM
1010 PRINT "LETTER ";S$
1020 RETURN
RUN
```

The computer will display:

```
LETTER A
LETTER B
LETTER C
LETTER D
LETTER E

DONE
```

In this program we assign different one-letter strings to the string variable S$, in lines 30, 50, 70, 90, and 110. Each time we do that, we then send the program to line 1000 by using a GOSUB instruction. In line 1010, the word LETTER is printed followed by whatever letter is stored in S$. At line 1020, a RETURN instruction sends the program back to the point where it was before jumping to the subroutine that begins at line 1000.

The suffix SUB in GOSUB stands for subprogram, sometimes called subroutine. This is a part of a program which does something that you will want repeated several times. In this case it printed the string "LETTER " with whatever the variable S$ was. It is important to remember that every subprogram must end in a RETURN.

This is a program in which it is vital to include the END instruc-

tion. Try leaving out line 130 and see what happens. (The reason it happens is that the program reaches line 120 and keeps going or, as we say, "falls through", to lines 1000-1020. In this case, a subprogram is being entered without a GOSUB, and that causes an error.) The END instruction at line 130 prevents the program from falling through into lines 1000-1020.

IF-THEN-ELSE

This is the instruction that lets a TI BASIC program make a decision.

Like or Unlike

One of the things that makes computers powerful is the ability to make decisions. For example, a program might instruct the computer to do one thing if a condition is true and another thing if it is false.

TI BASIC uses six different true/false tests. They are:

=	Equal to
<>	Not equal to
>	Bigger than
<	Smaller than
<=	Smaller than or equal to
>=	Bigger than or equal to

When you use one of these true/false tests you are instructing the computer to test whether something is equal to something else, bigger than something else, smaller than something else, and so on. To illustrate, type:

```
10 REM DEMO IF-THEN
20 CALL CLEAR
50 PRINT "1 + 2 = 3"
60 IF 1 + 2 = 3 THEN 90
70 PRINT "CONDITION IS FALSE"
80 GOTO 100
90 PRINT "CONDITION IS TRUE"
100 END
RUN
```

The computer will display:

```
1 + 2 = 3
CONDITION IS TRUE
DONE
```

At line 60 the program made a decision. If 1 + 2 = 3 were true it jumped to line 90, just as it would for a GOTO. You might think of the IF-THEN-ELSE instruction as containing a built-in GOTO. In fact, a line number and nothing else must follow both the THEN and the ELSE, just as with a GOTO. If 1 + 2 = 3 had not been true, the program would have kept going to line 70. Try it and see—change lines 50 and 60 to look like this:

```
50 PRINT "1 + 6 = 3"
60 IF 1 + 6 = 3 THEN 90
```

Now run the program and see what happens.

The IF-THEN instruction may be used with ELSE. Change the program again to look like this, and then run it:

```
10 REM DEMO IF-THEN-ELSE
20 CALL CLEAR
50 PRINT "1 + 2 = 3"
60 IF 1 + 2 = 3 THEN 90 ELSE 70
65 PRINT
70 PRINT "CONDITION IS FALSE"
80 GOTO 100
90 PRINT "CONDITION IS TRUE"
100 END
RUN
```

FOR-TO-STEP-NEXT

This instruction is used to do something repetitively, a certain number of times. FOR-TO-STEP-NEXT does that by starting with the FOR instruction.

FOR uses a variable, often named J. That variable will start off being equal to a beginning number and will increase until it reaches an ending number. FOR every increase of the variable, something will be done. Then the NEXT instruction sends the program back to the line where FOR occurs, and the process repeats. Type:

```
10 REM DEMO FOR-TO-STEP-NEXT
20 CALL CLEAR
30 FOR J = 1 TO 5
40 PRINT J
50 NEXT J
60 END
RUN
```

The computer will display:

```
1
2
3
4
5

DONE
```

Line 30 tells the computer to do everything between the FOR instruction and the NEXT instruction a number of times. In this case it is five times; once when J equals 1, once when J equals 2, and so on through J = 5. The FOR instruction is always followed by a numeric variable, in this case J, but it could be any numeric variable. In this example the variable J will have five values, from 1 through 5. Each time line 30 is executed, it adds one to the value of J.

For each value of J, line 40 will simply print the value of J, which is one greater each time it happens.

At line 50, the NEXT instruction will send the program back to line 30, until J = 6. When J = 6 it is larger than the limit set in line 30. When this happens, NEXT does not send the program back to line 30, but instead sends it on to the next line, line 60, where it ends.

This going back and starting over is called a loop. Since it only happens a certain number of times, it is called a controlled loop. A controlled loop will not go on endlessly, but will automatically end after a certain number of repetitions.

So where does STEP come in? Change line 30 so that it looks like this and then run it:

```
30 FOR J = 0 TO 10 STEP 2
```

The computer will display:

```
0
2
4
6
8
10

DONE
```

The STEP instruction takes an argument, in this case 2, which tells the computer to count only certain values of J. In this case the computer only counted every second value of J. Try running this program using different arguments for the STEP instruction.

As you can see, the STEP instruction may be used or left out. If you do not actually write it, the computer assumes you want the value of the STEP argument to be 1. This is known as a "default value"—a value which the computer automatically uses unless you give instructions to do otherwise.

JOYST

Joysticks are used with many games and other software available in command modules, but you may also write your own programs to use the joysticks.

We discussed how to install the joysticks in the first chapter. If you have joysticks, plug them in now. (If you do not, you may want to skip to the next chapter.) Look at the joysticks. Each has a lever and a red bar, called a fire button.

The JOYST instruction is used to make the computer respond to movements of the joystick. The following program

demonstrates how to use the JOYST instruction. It is a rather long program, so be careful to type it exactly as shown, and to check every line that you type. If you do make a mistake, use the FCTN keys to go back and change the line, as we learned how to do in chapter 3. Before you run this program, make sure that the ALPHA LOCK key is *not* pressed.

```
10 REM DEMO JOYST
20 CALL CLEAR
30 CALL CHAR(128,"00247EDB7E
  3C1800")
40 R=12
50 C=16
60 X=0
70 Y=0
75 CALL CLEAR
80 CALL HCHAR(R,C,128)
90 CALL JOYST(1,X,Y)
100 IF (X=0)*(Y=0) THEN 200
110 IF (X=4)*(Y=0) THEN 300
120 IF (X=-4)*(Y=0) THEN 400
130 IF (X=0)*(Y=4) THEN 500
140 IF (X=4)*(Y=4) THEN 600
150 IF (X=-4)*(Y=4) THEN 700
160 IF (X=0)*(Y=-4) THEN 800
170 IF (X=4)*(Y=-4) THEN 900
180 IF (X=-4)*(Y=-4) THEN 1000
190 END
200 REM NO MOVE
230 GOTO 75
300 REM MOVE RIGHT
310 C=C+1
330 GOTO 75
400 REM MOVE LEFT
410 C=C-1
430 GOTO 75
```

```
500 REM MOVE UP
520 R=R-1
530 GOTO 75
600 REM MOVE UP & RIGHT
610 C=C+1
620 R=R-1
630 GOTO 75
700 REM MOVE UP & LEFT
710 C=C-1
720 R=R-1
730 GOTO 75
800 REM MOVE DOWN
820 R=R+1
830 GOTO 75
900 REM MOVE DOWN & RIGHT
910 C=C+1
920 R=R+1
930 GOTO 75
1000 REM MOVE DOWN & LEFT
1010 C=C-1
1020 R=R+1
1030 GOTO 75
```

The first thing you may want to do now that you have typed in this program, is to save it to cassette or disk. If you save it, you will never have to type it in again. You will then be able to load it from the disk or tape using the 'OLD' command.

Now start the program by typing:

```
RUN
```

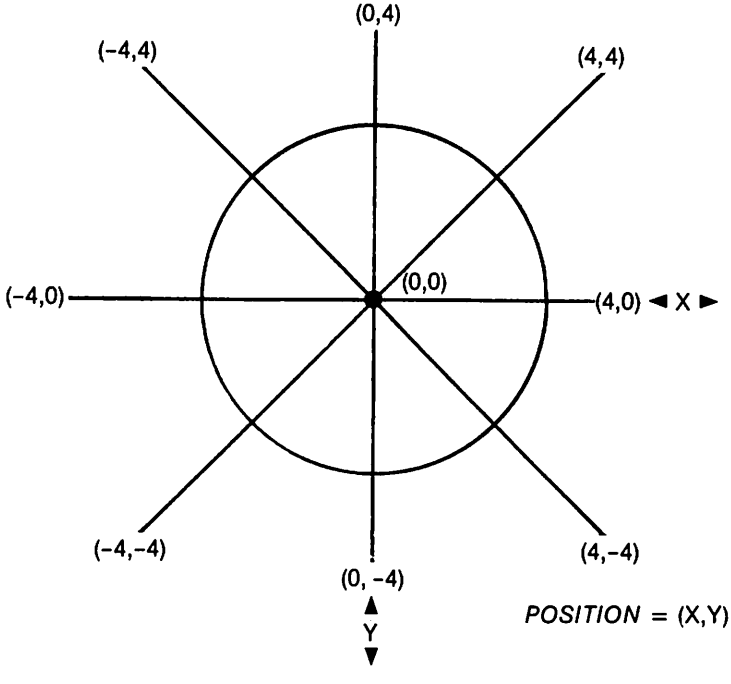
Remember, the way to stop this program is to press **FCTN 4** or to run the saucer off the edge of the screen.

This program will display a flying saucer in the middle of the screen. Nothing else will appear on the screen. The saucer will remain where it is until you move the joystick. Then it will move

in the direction you move the joystick (provided the ALPHA LOCK key is not pressed).

This is our most complicated program so far, and it will reveal many things about programming. It has one serious flaw (we call them "bugs")—when the little flying saucer reaches the edge of the screen the program is interrupted by an error (we say it "crashes"). In the next chapter we will improve upon this program, and fix that bug.

Now let's take a look at this program. First, we must understand how the JOYST subprogram works, as shown in the following illustration.



The X axis is the right and left direction, and the Y axis is the up and down direction. The JOYST subprogram has three arguments. The first is the key unit, which we will discuss

shortly. The other two are variable names that you make up yourself. The first one will store one of three numbers, 4, 0 or -4, depending on whether the joystick is moved right or left, or not at all. The second variable will store one of three numbers, 4, 0 or -4, depending on whether the joystick is moved up or down, or not at all. Why the JOYST instruction uses 0, 4 and -4 instead of some other numbers is a mystery, but Texas Instruments must have had some good reason for it.

When you move the joystick, one of these numbers is stored in each of the two variables, according to the diagram above. For example, if you move the joystick up, a 0 will be stored in the first variable and a 4 will be stored in the second. If you do not move the joystick at all, a zero will be stored in both variables.

Here, then, is how the program works:

Line 30 defines the shape of the flying saucer and gives it the ASCII code 128. This is similar to what we did on page 88 when we defined an upside-down letter A.

Lines 40 and 50 set the initial value of (we say "initialize") the variables R (row) and C (column) so that the first time the HCHAR subprogram is executed the flying saucer will be in the center of the screen (row 12, column 16).

Lines 60 and 70 initialize X and Y to zero so that the first time the JOYST subprogram is executed the flying saucer will not move. As you can see in the above illustration, as long as X and Y are both zero, the flying saucer stays where it is. When the joystick is moved, the values of X and Y change from zero to either 4 or -4.

Line 75 clears the screen, so that it does not become cluttered with flying saucers at old positions. This line is what causes the flying saucer to blink, even if you do not move it.

Line 90 assigns new values to X and Y, depending on how the joystick is moved or not moved. The first argument to the JOYST subprogram may be a one or a two, depending on which of the two joysticks you are using. In our example, we assumed you would use "key unit 1" (Joystick #1).

Lines 100 through 180 each refer to one of the possible combinations of movement—up and right, down and left, up only, etc. Each of these sends the program to a set of lines between 200 and 1000. You will not understand how lines 100-180 work, but do not worry. We will explain what they do in the next chapter in a section called Truth Tables.

Line 190 ends the program. This line is there for form's sake. The program is an endless loop, and if the program works correctly, it should never get to line 190.

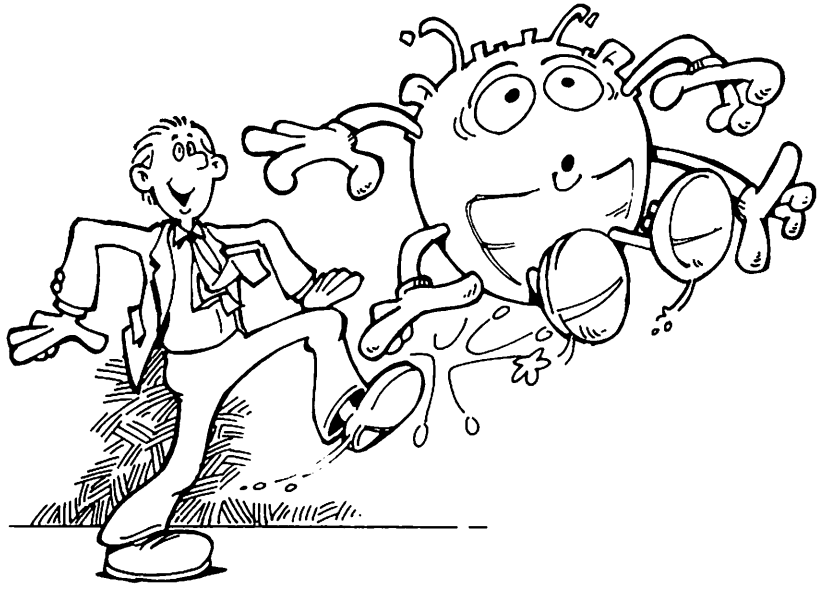
Lines 200 through 1000, which we have already mentioned, take care of changing the value of R and C so that the next time HCHAR is executed, the flying saucer will be displayed in a new position (unless the joystick has not been moved, in which cause the saucer will stay where it is). Each group of numbers starting with a different digit (200, 300, etc.) winds up sending the program back to line 75, where the loop starts over.

One minor, but important, feature of this program is the way it is laid out. Each group of lines in the series 200, 300 and so on, all reset the value of C and R (or leave them as they are).

Recall that the numeric variable C is the number of the column on which the saucer sits at any time. The numeric variable R is the number of the row on which the saucer sits. For example, if the saucer is at the point where column 8 and row 16 come together, C = 8 and R = 16.

The value of C is always set in a '10 line: 310, 410, etc. The value of R is always set in a '20 line. Also, each group in this series is documented with a REM statement. Anyone looking at this program for the first time would be able to tell from the REM statements that each group of lines moves the saucer in a particular direction.

Lines 100 through 180 are spaced so that they are very easy to see and compare. The pattern is apparent: in each of these lines, the value of X changes in a pattern that repeats three times; the value of Y changes three times, once every three lines. As you will see as you study the program, these patterns make is easy to detect a mistake in the pattern, like spotting one soldier out of step in close order drill. This, in turn, often makes finding a bug much easier and quicker. All these things make reading and debugging a program a little easier.



SOME TRICKS

As we begin to actually write programs, you should know a few simple tricks which make it a little easier to compose programs. These tricks involve two instructions which you already know from chapter one.

Trick #1—LIST

You already know that to show a program you simple type LIST. Often, however, you only want to examine a part of a program. You can still do this with the LIST instruction, by giving it arguments. The arguments are the line numbers you want to see. Type in the following program:

```
10 PRINT "AAAA"
20 PRINT "BBBB"
30 PRINT "CCCC"
40 PRINT "DDDD"
50 PRINT "EEEE"
60 PRINT "FFFF"
70 PRINT "GGGG"
80 PRINT "HHHH"
90 PRINT "I I I I"
100 PRINT "JJJJ"
```

Now type:

```
LIST 50-80
```

The computer will display:

```
50 PRINT "EEEE"
60 PRINT "FFFF"
70 PRINT "GGGG"
80 PRINT "HHHH"
```

As you see, the LIST instruction will display a sequence of lines if you give it the first and last line numbers as arguments, separated by a hyphen.

Now type:

```
LIST 50-
```

The computer will display:

```
50 PRINT "EEEE"
60 PRINT "FFFF"
70 PRINT "GGGG"
80 PRINT "HHHH"
90 PRINT "I I I I"
100 PRINT "JJJJ"
```

If you give a line number followed by a hyphen, the LIST instruction will display the entire program from that line number on.

Now type:

```
LIST -40
```

The computer will display:

```
10 PRINT "AAAA"
20 PRINT "BBBB"
30 PRINT "CCCC"
40 PRINT "DDDD"
```

If you give a hyphen followed by a line number, LIST will display the entire program from the beginning to that line.

Finally type:

```
LIST 70
```

The computer will display:

```
70 PRINT "GGGG"
```

LIST also displays a single line, if that line number is the only argument.

You can see that the LIST instruction is much more versatile than we had seen up to now.

Trick #2—RUN

We have already met RUN, the instruction used to tell the computer to execute a program that is in its memory. But RUN is also a handy tool for testing parts of a program without having to run the whole program or take the program out of memory.

Add lines 900 to 940 to our program so that it looks like this:

```
10 PRINT "AAAA"
20 PRINT "BBBB"
30 PRINT "CCCC"
40 PRINT "DDDD"
50 PRINT "EEEE"
60 PRINT "FFFF"
70 PRINT "GGGG"
80 PRINT "HHHH"
90 PRINT "I I I I"
100 PRINT "JJJJ"
900 PRINT "1111"
910 PRINT "2222"
920 PRINT "3333"
930 PRINT "4444"
940 PRINT "5555"
```

If you now type RUN, the computer will display:

```
AAAA
BBBB
CCCC
DDDD
EEEE
FFFF
GGGG
HHHH
IIII
JJJJ
1111
2222
3333
4444
5555
```

Now type:

```
RUN 900
```

The computer will display:

```
1111
2222
3333
4444
5555
```

If you give the RUN instruction an argument that is a line number, it will start running the program at that line number. If that line number is in the middle of the program, the computer will start running the program in the middle. Sometimes you may want to do that—for example, to see how the end of the program runs without waiting to go through the first half of it.

Often, you might want to test a few lines of a program before putting them into the program. To do that, first type them in with higher line numbers than the rest of the program and run

them alone. If they work, you may then retype them in the middle of the program, wherever you want them to go.

Programs

We will now write some interesting programs. We will use only instructions that you have already met. If you wish, please feel free to go back to chapters three and four to refresh your understanding of the instructions we will use.

Input Routines

A routine is a part of a program that performs a specific task. Thus the part of a program that receives input is an input routine. This input routine will be used to receive data from the person running the program.

Input routines are used very often, so we are going to start by writing a couple of them, one for inputting numbers and one for inputting strings.

In this routine, the line numbers may puzzle you. Don't worry, remember that in TI BASIC a program may begin with any line number, as long as its highest line number is less than 32000.

String Input Routine

```
1000 REM INPUT STRING
1050 CALL CLEAR
1100 INPUT "TYPE SOMETHING P
  LEASE ":CH$
1200 CALL CLEAR
1210 PRINT "SAY WHAT?"
1300 PRINT CH$
1310 PRINT "HOW ABOUT THAT?"
2000 END
```

To run this routine type:

```
RUN
```

This is a simple program that you use to send information to the computer by typing at the keyboard. All the work is done in line 1100 by the INPUT instruction which we have already seen. The INPUT instruction firsts asks for a string, and then stores what you type in the string variable CH$. Line 1200 clears the screen. Line 1210 asks a question. Line 1300 prints back the string that you typed before. Line 1310 prints another question.

Note the message "TYPE SOMETHING PLEASE". When we use this routine as part of a larger program, we may want to use some other message, such as "TYPE A NOUN, PLEASE". We can do that by changing line 110 so that it looks like this:

```
1100 INPUT MS$:CH$
```

Here, the string variable MS$ stands for a prompt. Each time we go to the routine we first assign a string to this variable with an instruction like:

```
300 MS$ = "TYPE SOMETHING PL
  EASE"
```

Then, until we change the string assigned to MS$, that will be the message used by the string input routine. Add line 300, change line 1100 as shown above, and try out the new program.

Number Input Routine

```
2000 REM INPUT NUMBER
2030 CALL CLEAR
2050 PRINT "PLEASE PRESS A N
  UMBER BETWEEN 3 AND 6"
2100 INPUT N
2300 IF (N<3)+(N>6)THEN 2030
  ELSE 2600
2600 PRINT N;"IS A NICE NUMB
  ER!"
2700 END
```

Once more, remember that we are showing these programs as they will appear on the TI 99/4A. A line like 2050 does not end until the quotation mark after the six, so don't press ENTER before then.

To run this routine type:

RUN

Line 2300 says that if the key pressed is less than 3 or more than 6, go back to line 2030, otherwise go on to line 2600. How does this line work? To find out, let's talk about truth tables.

Truth Tables—OR

In line 2300 we see the expression:

(N<3)

If N is smaller than 3 this expression will be true. If N is larger than or equal to 3 this expression will be false. So we can show these two possibilities like this:

N<3 true
N≥3 false

Now let's do the same thing for the expression:

(N>6)

As you may have already figured out, this will be true if N is bigger than 6 and false if N is less than or equal to 6. So...

N>6 true
N≤6 false

Now, here's the good part. We combine the two expressions so that a true plus a true is true, and true plus a false is true. A false plus a false is false. Let's show these four possibilities like this:

+	N<3 TRUE	N<3 FALSE
N>6 TRUE	TRUE	TRUE
N>6 FALSE	TRUE	FALSE

This kind of diagram is called a truth table. The four squares in the middle represent the four possible results of (N<3)+(N>6).

Study line 2300. Do you see that the expression (N<3)+(N>6) will be false only if N is 3, 4, 5 or 6?

Another way of saying this is that the expression is true if N<3 *or* if N>6. That is why this kind of expression is sometimes called an OR statement.

If the number pressed is outside the range of 3 through 6, then line 2300 then sends us back to the beginning to try again.

If the number pressed is within the range of 3 through 6, we go to line 2600. At 2600 we print the number and at 2700 we end.

More About Truth Tables—AND

While we are talking about expressions like the one in line 2300, and about truth tables, you should know that besides OR statements, the IF-THEN instruction can be used with AND statements. An AND statement looks just like an OR statement, except that the + sign is replaced with a * sign. If line 2300 were an AND statement, it would look like this:

```
IF (N>2)*(N<7)THEN 2600 ELSE 2030
```

The truth table for this statement looks like this:

*		N>2	
		TRUE	FALSE
N<7	TRUE	TRUE	FALSE
	FALSE	FALSE	FALSE

You can see that a true times a true is true, but a false times a true is false. A false times a false is also false.

Try using this truth table to see how such an AND statement would work.

We used AND statements in the JOYST routine in chapter 4.

Draw A Pyramid

The following program demonstrates the kind of graphics that the TI 99/4A can display.

When you type RUN, the screen will clear. Then a 12-story, colorful pyramid will grow before your eyes. When it is finished it will stay on the screen, in full color, until you press a key.

```
1 REM DRAW A PYRAMID
5 CALL CLEAR
10 CALL CHAR (128,"FFFFFFFF
  FFFFFFF")
20 C=5
30 L=24
40 R=24
60 FOR J=1 TO 12
65 CALL COLOR(13,7,1)
70 CALL HCHAR(R,C,128,L)
90 C=C+1
100 L=L-2
110 R=R-1
150 NEXT J
155 PRINT
160 PRINT "PRESS ANY KEY TO END"
170 CALL KEY(3,RV,SV)
180 IF SV=0 THEN 170
190 END
```

Typing in a program like this you are bound to make a few typographical errors. This will cause errors when you try to run the program. You will probably get error messages, such as INCORRECT STATEMENT, BAD VALUE, or some other error.

If you get an error, LIST your program and look it over carefully. Remember that even a missing space may cause an error. When you find the mistake, simply use the editing functions (see appendix) to correct it.

To run this program type:

RUN

The screen will clear and a bright, red pyramid will grow from the bottom up. Then the computer will wait until you press a key.

Line 10 defines a new character with the ASCII code of 128. This is defined as a solid block.

Lines 20 through 40 are preparing for the HCHAR subprogram to follow. They each assign a starting value to the arguments of the HCHAR subprogram. C, R, and L are all numeric variables. Each stands for one of the arguments of HCHAR: column, row, and length.

Lines 60 and 150 together are a FOR-NEXT instruction. Everything between them will be repeated twelve times. This will correspond to twelve steps in the pyramid. For each step, the numeric variable J will have a different value, from 1 through 12.

Line 65 is the COLOR subprogram. Its arguments are 13, 7 and 1. The first argument, 13, is the number of the character group to which ASCII code 128 belongs. If you do not remember what a character group is, go back and review chapter 3 (page 84).

The second argument of the COLOR subprogram is 7, which stands for the color red. You could try changing the second argument and see the different colors when you re-run the program. Or try using the variable J for the second argument.

The third argument is the background character. We have defined ASCII code 128 as a solid block that takes up the entire space available for a character. Therefore, the background color will not be seen, no matter what it is. We will assign it the value of 1, which stands for transparent.

Line 70 is the HCHAR subprogram. It draws a horizontal row of squares, which will be the steps of the pyramid. The squares are the character ASCII 128, as we have defined it. The number 128 is the third argument of HCHAR.

The first time around the FOR-NEXT loop, HCHAR's first, second and fourth arguments will be 24, 5 and 24, as they were

assigned in lines 20 through 40. This means that the pyramid's bottom step will start at row 24, the bottom of the screen; it will start 5 columns in from the left; it will consist of squares (the character of ASCII 128, as we have defined it in line 10); and it will be 24 columns long.

But these values will be changed by lines 90 through 110. The value of C will change from 5 to 6. The value of L will change from 24 to 22. The value of R will change from 24 to 23.

The second time around, the loop will return to line 60 where J will be given the new value of 2.

The second time around, the CALL HCHAR subprogram in line 70 will be changed, because the values of C, L and R were changed. They will be 23, 6 and 22. This will mean that the second step of the pyramid will be one row above the first, start one column to the right, and will be two columns shorter. To see how this works, look at the following table:

Value of J (time through loop)	Value of C (columns from left)	Value of L (length of line)	Value of R (rows from top)
1	5	24	24
2	6	22	23
3	7	20	22
4	8	18	21
5	9	16	20
6	10	14	19
7	11	12	18
8	12	10	17
9	13	8	16
10	14	6	15
11	15	4	14
12	16	2	13

Each time around the loop, the value of J will be different, and so will the values of C, L and R. Therefore, each time around the loop, the line of blocks drawn will be shorter and higher.

Try changing the second argument of the COLOR subprogram, and see that the pyramid's color will change. Also, try using the variable J as the argument.

This and the following programs may require a little review of the previous chapters, and a little concentration before they become entirely clear. Take your time. After a while a light bulb will go on over your head, and you will be well on your way to understanding programming.

Make Up A Story

This demonstrates the use of arrays, and the DIM instruction with which arrays are set up. It also shows how string variables are used to store strings that the user types in him/herself.

```
1 REM MAKE UP A STORY
3 DIM NOUN$(4)
6 DIM ADJ$(3)
10 FOR J=1 TO 4
15 CALL CLEAR
20 INPUT "PLEASE TYPE A NOU
  N ":NOUN$(J)
30 NEXT J
40 FOR J=1 TO 3
45 CALL CLEAR
50 INPUT "PLEASE TYPE AN AD
  JECTIVE ":ADJ$(J)
60 NEXT J
```

```
65 CALL CLEAR
70 INPUT "PLEASE TYPE A PLU
  RAL NOUN (E.G. BATS) ":PLN
  OUN$
80 CALL CLEAR
90 INPUT "PLEASE TYPE A VER
  B ":VERB$
95 PRINT
100 PRINT "PLEASE PRESS ANY
  KEY TO CONTINUE"
110 CALL KEY(3,RV,SV)
120 IF SV=0 THEN 110
135 CALL CLEAR
140 PRINT "ONCE UPON A TIME,
  A ";NOUN$(1);" LIVED IN A "
  ;ADJ$(1);" ";NOUN$(2);"."
150 PRINT "THIS ";NOUN$(1);"
  WAS SO AFRAID OF ";PLNOUN$;
  " THAT HE NEVER"
160 PRINT "WENT OUT OF THE "
  ;NOUN$(2);" WITHOUT A ";ADJ$
  (2);" ";NOUN$(3);"."
170 PRINT "ONE DAY THE ";NOU
  N$(1);" DECIDED TO ";VERB$;"
  HIS FAVORITE ";NOUN$(4);"."
180 PRINT "THAT EXPERIENCE W
  AS SO ";ADJ$(3);" THAT HE NE
  VER FEARED ";PLNOUN$;" AGAIN "."
190 END
```

Again, remember that it is easy to make typing mistakes. If the program does not work as it should, go back and check your typing with a fine tooth comb.

To run this routine type:

RUN

This program first asks you to type nouns, adjectives, a plural noun (a group of things, like ducks, houses, etc.) and a verb. Then the computer automatically makes up a story using the words that you have typed, and it prints that story on the screen.

The story is printed to the screen in lines 140 through 180. Notice the spacing—spaces often come after beginning quotation marks or before ending quotation marks so that the words will not be run together.

Within the program several words are missing from the story. Their place is held by string variables. These variables are NOUN$ for nouns, ADJ$ for adjectives, PLNOUN$ for a plural noun, and VERB$ for a verb. We use four nouns and three adjectives, so these variables are each combined into an array. In lines 3 and 6 we set aside space in memory for these two arrays. The four nouns are NOUN$(1) through NOUN$(4), and the three adjectives are ADJ$(1) through ADJ$(3).

In the first half of the program we ask the user to type in the actual words that these string variables will stand for. To input the nouns and adjectives we use FOR-NEXT loops.

Since we will input four different nouns, the FOR-NEXT loop for inputting NOUN$(1) through NOUN$(4) must go around four times. We ensure this in line 10, where we say that J will go from 1 through 4. Then, in line 20, the user is asked to input a noun, and that noun is assigned to NOUN$(1), NOUN$(2), NOUN$(3) or NOUN$(4), depending on the value of J. Notice that we always clear the screen before asking for the next noun. We do this in line 15.

In lines 40 through 60 we do for adjectives the same thing we did for nouns, but we only do it for three of them. The value of J goes from 1 through 3.

In lines 65 and 70 we clear the screen and input one plural

noun. Notice that we give the user an example of a plural noun in case he is not certain what a plural noun is. We do not use a loop here because we only need one plural noun.

In lines 80 and 90 we clear the screen and input a verb.

In line 95 we skip a line so that our next message, asking the user to hit any key to continue, will not be jammed up against the last line we printed.

In lines 100 through 120 we pause until the user hits any key. We could leave these lines out, and the program would still work. But if we did that, the story would be displayed very abruptly, as soon as the user finished inputting that last verb. Computers are so much faster than people that it is often a good idea to slow things down a little bit. In this case, it also gives the user a little extra feeling of control over what is happening. The program stops until *he* or *she* wants it to continue.

Then in lines 135 through 180 we print the story with the words that the user supplied. Sometimes the story will come out as nonsense, but other times it will make a funny kind of sense. Try it a few times!

Create A Word

In the last program we made up a whole story out of real words. In this one we will make up new words. The words we make up will all be five letters long, with their first, third and fifth letters consonants and their second and fourth letters vowels.

This program demonstrates the use of the GOSUB-RETURN instruction. It also shows how to use the RND instruction to randomly select one item from a list.

When you run this program the screen will clear, and a five letter word will appear. A message will also appear asking you to press Y to create another word, or press any other key to quit.

You may have heard that sometimes corporations use computers to pick their names. This is a simplified version of the kind of program they use to do that.

```
10 REM CREATE A WORD
13 RANDOMIZE
15 CALL CLEAR
20 CON$ = "BCDFGHJKLMNPQRSTV
   WXYZ"
25 VOW$ = "AEIOU"
30 GOSUB 1100
35 A$ = CN$
37 B$ = VW$
40 GOSUB 1100
45 C$ = CN$
47 D$ = VW$
50 GOSUB 1100
55 E$ = CN$
60 PRINT A$;B$;C$;D$;E$
70 PRINT
80 PRINT "TO CREATE ANOTHE
   R WORD PLEASE PRESS Y"
81 PRINT
83 PRINT "PRESS ANY OTHER KEY TO
   QUIT"
85 CALL KEY(3,RV,SV)
87 IF SV = 0 THEN 85
88 IF RV = 89 THEN 10 ELSE 89
89 IF RV = 121 THEN 10 ELSE 99
```

```
99 END
1100 REM SELECT RANDOM CHARA
  CTER
1110 CN$ = SEG$(CON$,INT(RND*21) + 1,1)
1120 VW$ = SEG$(VOW$,INT(RND*5) + 1,1)
1130 RETURN
```

To run this routine type:

```
RUN
```

This is a fairly straightforward program. The most complicated parts are that it uses a subroutine (lines 1100 through 1130), and it uses the RND function in lines 1110 and 1120, which require a bit of mathematics.

In line 13 we use the RANDOMIZE instruction. As you will recall, the RND instruction returns a series of random numbers. It returns the same series of numbers every time it is used. How boring! That would mean that every time we ran this program, we would get the same series of made-up words.

However, if we use the RANDOMIZE instruction, RND will return a different series of numbers each time it is used. This way, every time we run the program we will get a different series of made-up words.

In line 20 we create a string called CON$ which consists of all the consonants in the alphabet. In line 25 we create a string called VOW$ which consists of all the vowels.

In line 30 we go to the subroutine that starts at line 1100. It is in the subroutine that the real work of the program is done.

In line 1110 we do several things. Let's look at what they are. We will start with the expression:

```
INT(RND*21)+1
```

We want a number from 1 to 21 because we are going to pick out one of the consonants, and there are 21 of them. RND returns a number greater than 0 and less than 1, as we saw in Chapter 3. Whatever that number is, if we multiply it by 21, the result will be greater than 0 and less than 21. But we need a whole number, not a number with a decimal fraction. If we apply the INT instruction to this number, we will get a whole number from zero through 20. We then add 1 to get a whole number from 1 to 21.

We do our picking, still in line 1110, with the SEG$ instruction. By now INT(RND*21)+1 has been converted to a number from 1 to 21, so we may represent this:

```
SEG$(CON$,INT(RND*21)+1,1)
```

...as this:

```
SEG$(CON$, (1 through 21), 1)
```

Recall from Chapter 3 that the SEG$ instruction takes three arguments. The first is a string, in this case CON$, which is all the consonants. The second argument of SEG$ is the location within the string of a substring. In this case the location will be between 1 and 21. The third argument is the length of the substring, in this case 1. In other words, we are picking out a substring consisting of a single consonant, and it might be any of the 21 consonants, chosen at random.

So now the above expression-SEG$(CON$,INT(RND*21)+1,1)-has been reduced to one of the 21 consonants, and we may

write line 1110 as:

CN$ = "a random consonant"

The consonant we have picked is assigned to the string variable, CN$.

In line 1120 we do the same thing with the vowels that we did in line 1110 with the consonants. The arithmetic is slightly different since there are only 5 vowels. We pick one at random and assign it to the string variable VW$.

In line 1130 we return to where we came from, in this case the end of line 30.

Next we execute lines 35 and 37. These simply assign our consonant to the string variable A$ and our vowel to the string variable B$. Why not leave them in the string variables CN$ and VW$? Because we are about to go back to the subroutine and pick out a different consonant and a different vowel, as we continue constructing our 5 letter word. In the meantime we do not want to lose our first two letters, so we put them in A$ and B$.

In lines 40 through 47 we again pick out a consonant and a vowel, and assign them to the string variables C$ and D$.

In lines 50 and 55 we go back one more time, pick out our last letter, a consonant, and assign it to E$.

In line 60 we print our word by printing the five string variables, A$,B$,C$,D$ and E$. The word will be something like TOQIT or BASOP or JEPIX, etc.

Lines 80 through 88 give the user a chance to create another word or quit. If he presses Y (ASCII 89), the loop that creates a word goes around again. If he decides to quit, the program skips to line 99, and ends. Notice the necessity of the END

instruction here. If it were missing the program would keep going into the subroutine at 1100 (we say it would *fall through* to the subroutine). Then it would come to line 1130 and encounter a RETURN without having had a GOSUB. This would cause a problem, and the program would stop and give an error message.

A word about subroutines: as you have probably noticed, we could have written this program without a subroutine, by simply writing the program lines in the subroutine three separate times, between lines 25 and 35, between lines 37 and 45, and between lines 47 and 55. Obviously, we save space by using a subroutine instead.

Compose A Tune

This program lets you compose a simple tune of up to 100 notes. The 100 notes start out all the same, with a frequency too high to hear. First, you can compose a tune by changing any of these to notes that you can hear. Then you can go back and play the tune as many times as you wish.

This program demonstrates the use of an array to store the tones that make up a tune. It also shows you how to make sure that a user types in only information that the computer can use. It uses subroutines to take information from the user. Finally, it will give you practice in using the SOUND instruction, with which the TI 99/4A makes music.

```
1 REM COMPOSE AND PLAY A TUNE
5 TUNLEN = Ø
6 DIM TONE(1ØØ)
7 FOR J = 1 TO 1ØØ
8 TONE(J) = 4ØØØØ
9 NEXT J
1Ø CALL CLEAR
2Ø PRINT "THIS PROGRAM LETS
  YOU COMPOSE AND PLAY A
  TUNE OF UP TO 1ØØ NOTES."
```

```
30 PRINT
35 PRINT "PLEASE PRESS ANY K
   EY TO CONTINUE."
40 GOSUB 2000
100 REM EDIT TUNE
101 J=1
102 CALL CLEAR
103 IF J>100 THEN 200
105 PRINT "NOTE #";J
110 PR$="PLEASE ENTER A NUMB
   ER FROM 110 TO 44733, INCLU
   SIVE "
115 GOSUB 1000
116 PRINT
117 IF CH$="" THEN 102
135 IF (ASC(CH$)<49)+(ASC(CH
   $)>57) THEN 102
139 TONE(J)=VAL(CH$)
140 IF (TONE(J)>109)*(TONE(J)
   <44734) THEN 160
150 PRINT "BETWEEN 110 AND 4
4733, PLEASE"
151 FOR K=1 TO 400
152 NEXT K
153 PRINT
155 GOTO 102
160 CALL SOUND(1000,TONE(J),1)
161 TUNLEN=J
162 J=J+1
165 CALL CLEAR
170 PRINT "PLEASE TYPE F TO
   FINISH, ENTER TO GO ON."
172 PRINT
175 GOSUB 2000
179 REM TEST FOR ENTER
180 IF RV=13 THEN 102
```

```
190 REM TEST FOR UPPER OR LO
  WER-CASE F
191 IF (RV=70)+(RV=102) THEN
   210
199 GOTO 170
200 PRINT "THAT IS ALL THE N
  OTES I CAN STORE NOW."
205 FOR K=1 TO 300
206 NEXT K
210 REM PLAY TUNE
215 FOR J=1 TO TUNLEN
220 CALL SOUND(1000,TONE(J),1)
225 PRINT TONE(J);
230 NEXT J
231 PRINT
232 PRINT
235 PRINT "PLEASE PRESS ANY
  KEY TO CONTINUE."
236 GOSUB 2000
238 CALL CLEAR
240 PRINT "PLEASE TYPE P TO
   PLAY AGAIN, ENTER TO
  QUIT."
250 GOSUB 2000
260 IF RV=13 THEN 299
270 IF (RV=80)+(RV=112) THEN
   210
280 GOTO 240
299 END

1000 REM INPUT STRING
1100 INPUT PR$:CH$
1990 RETURN
2000 REM INPUT SINGLE KEY
2100 CALL KEY(3,RV,SV)
2200 IF SV=0 THEN 2100
2300 RETURN
```

To run this program, first make sure that the two input subroutines are as shown. These two subroutines are lines 1000 through 1990 and lines 2000 through 2300. These are changed a little bit since we first saw them as routines at the beginning of this chapter.

Then type:

RUN

First you will be shown an introductory message and asked to press a key. You then will be greeted with a message that asks you to enter the frequency of your first tone. Once you do that you will hear the tone. Then you will have a choice of going on to more tones or pressing F to indicate you are finished.

NOTE: There is a table of musical tone frequencies in the appendix.

When you have composed as many notes as you wish (up to 100), press F. The computer will play the tune you have composed. After that, you may play the whole tune again, or quit.

1. Initialization

Lines 5 through 9 initialize certain values which the program will use. You will recall that many programs have a section, usually near the beginning, in which values are initialized. Usually it is variables that are initialized. Later in the program

they will be assigned different values, but it is important that they always start off with their initialized values.

One such variable is the numeric variable TUNLEN (tune length). The value in TUNLEN at any time is the number of notes in our tune at that time. It starts at zero and grows up to the value 100. If TUNLEN started at some value other than zero, its value throughout the program would be thrown off, and the program would not work correctly.

You will recognize line 6 as setting aside room in an array for 100 different values of the numeric variable TONE. This array will hold the frequencies of all the tones that will make up our tune. TONE(1) will hold the first tone, TONE(2) will hold the second tone, and so on, up to TONE(100).

Lines 7 through 9 assign the value 40000 to every element in the TONE array. This FOR-NEXT loop goes around 100 times. Each time, another element in the TONE array is assigned the value 40000. The 40000 frequency cannot be heard by humans. If we did not initialize the TONE array this way, we would have no way of knowing what frequencies its elements would start out with. We might produce some very strange sounds.

2. Compose a Tune

We compose our tune in lines 100 through 206. Line 101 sets J to 1. We are going to compose the tune one tone at a time. As before, the first tone will be TONE(1), the second will be TONE(2) and so forth. So for the first tone, J must equal 1.

Line 102 is the beginning of a loop, but not a FOR-NEXT loop. This loop ends at line 180, with a GOTO that sends it back to line 102. Each time we go around this loop we will come back to 102, and the screen will clear. Then line 103 will check to see if J is greater than 100. If it is, that means that we have reached

the maximum length of our tune—remember that the most tones it can have is 100. If line 103 finds that we have reached our limit, it sends us to line 200 which tells us so. Then line 210 starts the process of playing the tune.

In the meantime, before we reach the limit of 100 tones, we go on to line 105, which prints a message that looks like this:

NOTE # 1

Each time around the loop, the number of the note will increase by one.

Line 110 assigns a message to the string variable PR$. Line 115 sends us to the input subroutine that begins with line 1000. That subroutine will use PR$ to print out an input prompt. This prompt asks that you enter a frequency for this tone.

But what if you enter an invalid frequency, say 100, or press ENTER without typing any frequency first? Without lines 117 and 135 an error condition would arise. But these lines make sure that if you try to enter an incorrect number, or no number, you are notified and sent back to line 102 to try again.

The input routine assigns the new frequency to the variable CH$, and returns us to the end of line 115. Line 117 checks the value of CH$. If you pressed ENTER instead of selecting a frequency, the value of CH$ will be nothing (we say *null*). A null string is nothing between quotation marks, just as a normal string is one or more characters between quotation marks. If CH$ is null then you are returned to line 102. J has not been changed.

On the other other hand, if CH$ was not null we go to line 135, which is another OR statement. Let's look at it:

IF (ASC(CH$)<49)+(ASC(CH$)>57) THEN 102.

Recall that the ASC instruction returns the ASCII code of the first character in a string. In this case the string is CH$, the input you have just made. If this input is a number, its first character will be 1 through 9, or ASCII code 49 through 57. If it is anything else, we do not want it. If the first character of our input has an ASCII lower than 49 or higher than 57, we simply go back to 102, and start the loop over by asking for a number between 110 and 44733, inclusive. (This happens in the 1000 subroutine entered from line 115.) Again, nothing has changed.

If, however, we have a good input for CH$, we drop through to line 139. Here, the frequency of the current tone is made equal to VAL(CH$), that is, the numerical value of the string CH$.

Line 140 is the AND statement we promised to see. It says that if TONE(J) is greater than 109 and TONE(J) is also smaller than 44734, that we should go to line 160. If TONE(J) is greater than 109, then this condition is true. If TONE(J) is smaller than 44734 then this condition is true as well. Now, in an AND statement we multiply, instead of adding as we did in the OR statement. Here is the truth table for this AND statement:

		TONE(J) > 109 TRUE	TONE(J) > 109 FALSE
*	TRUE	TRUE	FALSE
TONE(J) < 44734	FALSE	FALSE	FALSE

Notice that out of four possibilities, only one is true. That is if TONE(J) is both above 109 *and* below 44734. This is why this is called an AND statement.

If the input, CH$, is not within this range, we drop through from line 140 to line 150. Lines 150 through 155 remind us what the acceptable range is and send us back to line 102 to try again.

Lines 151 and 152 are a delay loop to give us time to read the

message in line 150 before continuing. Notice that we use the variable K instead of J. The value of J must not be changed during the loop we are in, where it is keeping count of which note is current. If we also used J in a delay loop, we would lose our count.

If the input, CH$, is within the acceptable range, we jump down to line 160. Line 160 plays the current tone.

Line 161 changes the value of TUNLEN to the new length of the tune. This will be one tone longer than before, since we have just added one new tone. Line 162 adds one to the value of J. We do this to keep track of which go-around of the loop we are about to finish.

Line 170 then tells us to press the letter F if we are finished composing, or to press the ENTER key to continue composing.

Line 175 then takes our answer to this prompt.

In lines 180 and 191, we test the input taken by the subroutine at 2000. ASCII 13 is ENTER. ASCII 70 and 102 are upper and lower case F. We make sure to include the lower case, because the ALPHA LOCK key might be up.

If we press **ENTER**, line 180, takes us back to repeat the loop again.

If we pressed an F (or an f), line 191 sends us down to line 210, taking us out of the loop.

If we pressed any other key, lines 180 and 191 do nothing. We go on to line 199 which sends us back to 170 to ask again for either **ENTER** or **F.**

3. Playing a Tune

The tune you have now composed is played by the lines starting at 210. At line 215 we begin a FOR-NEXT loop that goes around as many times as there are notes in the tune, that is, from 1 to TUNLEN.

During each go-around of the FOR-NEXT loop between lines 215 and 230, the program plays a tone at line 220 and prints the frequency of the tone at line 225. We have given the tones a duration of 1000 and a loudness of 1. Only the frequency changes, from tone to tone. Each tone is different, because for each tone the value of J is different. Thus, the frequency stored in TONE(J) in line 220, would be different for each go-around of the loop. (Remember that during this FOR-NEXT loop J goes from 1 to the value of TUNLEN. If, for example, the tune were 10 tones long, the value of TUNLEN would be 10, and the value of J would go from 1 to 10.)

Lines 231 and 232 skip lines. Lines 235 through 236 stop things until the user presses a key. Line 238 clears the screen. Line 240 prints a prompt message. In line 250 we go to the subroutine at line 2000.

Lines 260 and 270 test our input from the subroutine. If we have pressed P, we go back to line 210, and the tune is played again. If we have pressed ENTER, we will go to line 299, where we end. If neither of these conditions are true, we will keep going and hit line 280, which sends us back 240, asking for **P** or **ENTER.**

Pilot A Flying Saucer

This is the same routine we used in Chapter IV to illustrate use of the joystick. We have changed it slightly. Also, we have added lines to prevent the saucer from flying off the screen and to let you stop the program without having to press either

FCTN 4 or **FCTN +**. Also, the line numbers are different.

When you run this program the screen will clear and you will see messages that remind you to make sure that the ALPHA LOCK key is up. Another message tells you to start the program by pressing the fire button on joystick one (the red bar). If you do not know which joystick is number one, try them both.

Once you press the fire button the screen will again clear, and you will see a little flying saucer in the center of it. The saucer will move in whatever direction you move the joystick. When you are done, simply press the fire button again.

```
100 REM PILOT A FLYING SAUCER
110 CALL CLEAR
113 PRINT "PLEASE MAKE SURE
   ALPHA LOCK KEY IS UP"
117 PRINT
120 PRINT "PLEASE USE JOYSTICK"
121 PRINT "NUMBER ONE"
122 PRINT
123 PRINT "THE LEVER MOVES
  THE FLYING SAUCER—THE FIRE
  BUTTON QUITS"
124 PRINT
125 PRINT "PRESS THE FIRE
  BUTTON TO START"
126 CALL KEY(1,RV,SV)
127 IF SV=0 THEN 126
128 IF RV<>18 THEN 126
129 REM INITIALIZE
130 CALL CHAR(128,"00247EDB
  7E3C1800")
```

```
133 R=12
136 C=16
140 X=0
150 Y=0
160 RV=0
170 FOR J=1 TO 400
171 NEXT J
174 REM INPUT FROM JOYSTICK
175 CALL CLEAR
176 CALL HCHAR(R,C,128)
177 CALL KEY(1,RV,SV)
179 IF RV=18 THEN 1200
180 CALL JOYST(1,X,Y)
185 REM HANDLE INPUT
190 IF (X=0)*(Y=0) THEN 300
200 IF (X=4)*(Y=0) THEN 400
210 IF (X=-4)*(Y=0) THEN 500
220 IF (X=0)*(Y=4) THEN 600
230 IF (X=4)*(Y=4) THEN 700
240 IF (X=-4)*(Y=4) THEN 800
250 IF (X=0)*(Y=-4) THEN 900
260 IF (X=4)*(Y=-4) THEN 1000
270 IF (X=-4)*(Y=-4) THEN 1100
300 REM NO MOVE
310 GOTO 175
400 REM MOVE RIGHT
410 C=C+1
415 IF C<33 THEN 420
417 C=C-1
420 GOTO 175
500 REM MOVE LEFT
510 C=C-1
515 IF C>0 THEN 520
517 C=C+1
520 GOTO 175
600 REM MOVE UP
610 R=R-1
```

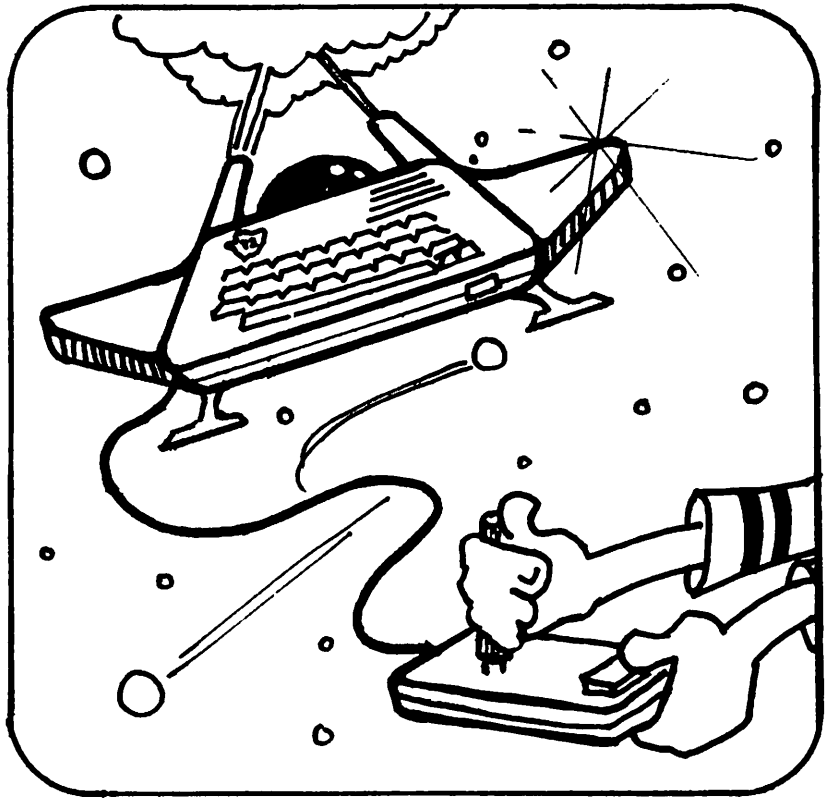
```
615 IF R>0 THEN 620
617 R=R+1
620 GOTO 175
700 REM MOVE UP & RIGHT
710 C=C+1
720 R=R-1
725 IF (C<33)*(R>0) THEN 730
727 C=C-1
728 R=R+1
730 GOTO 175
800 REM MOVE UP & LEFT
810 C=C-1
820 R=R-1
825 IF (C>0)*(R>0) THEN 830
826 C=C+1
828 R=R+1
830 GOTO 175
900 REM MOVE DOWN
910 R=R+1
915 IF R<25 THEN 920
917 R=R-1
920 GOTO 175
1000 REM MOVE DOWN & RIGHT
1010 C=C+1
1020 R=R+1
1025 IF (C<33)*(R<25) THEN 1030
1027 R=R-1
1028 C=C-1
1030 GOTO 175
1100 REM MOVE DOWN & LEFT
1110 C=C-1
1120 R=R+1
1125 IF (C>0)*(R<25) THEN 1130
1127 C=C+1
1128 R=R-1
1130 GOTO 175
1200 CALL CLEAR
1210 END
```

If you like, you can save this program on a disk or cassette. If you do, you will not have to retype it when you want to use it again.

To run this program type:

```
RUN
```

Since we studied this program in Chapter 4, we will look closely only at the changes we have made in it. To refresh your memory, however, here is a table of the variables used in this program, which lets you use the joystick to move a flying saucer around on the screen.



variable's name	variable's use	variable's values
X	joystick input, horizontal	-4, 0, or 4
Y	joystick input, vertical	-4, 0, or 4
R	current row where saucer is	1 to 24
C	current column where saucer is	1 to 32
RV	return variable in CALL KEY instruction	18 when the fire button is pushed

Lines 120 through 125 remind you to make sure that the ALPHA LOCK key is up. (Remember that the joysticks will not work correctly if the ALPHA LOCK key is down.) These lines also remind you how to use the joystick, and they tell you how to both begin and end the program: by pressing the fire button—the red bar at the end of the joystick where the connecting wire comes out of it.

Line 126 is our old friend the KEY subprogram. When the first argument of the KEY subprogram is 1, it means two things. First, it means that if you are using a joystick, it must be joystick number one. Second, it means that if the fire button is pressed, the number 18 will be assigned to the variable RV. In other words, the fire button works just like a keyboard key—it sends its own code (18) to the return variable of the KEY subprogram.

Line 128 looks at RV, and if it is an 18, you drop through and continue the program. Otherwise, you go back to line 126. This little loop goes around thousands of times a second. What you see when you run the program is simply that nothing happens until you press the fire button on joystick number one.

Line 160 sets RV to 0. If we did not do this, RV would still be 18 when we came to line 177, and the program would end.

Lines 170 and 171 are there for a related reason. They simply delay the program. If we did not delay the program, it would be

very easy to hold the fire button down long enough at line 126 for the program to get to line 177, and the program would begin and end in the same moment.

Line 175 is the beginning of the large loop that goes around and around from here on, as long as the program is running.

The first thing that happens in this loop is that the flying saucer is displayed—at line 176.

The second thing that happens, at line 177, is that we check to see if the fire button has been pressed since the last time around the loop. If it has, we go right to line 1200, where the program ends.

The only other change in this program since we first saw it in Chapter 4, is that the flying saucer can no longer fly off of the edge of the screen. We prevent it from doing so each time it moves in any direction. This is done with one of the following groups of lines: 415-417, 515-517, 615-617, 725-728, 825-828, 915-917, 1025-1028, and 1125-1228.

Each time the saucer makes a move, the program uses one of the eight small subroutines which begin at line 400, 500, 600, 700, 800, 900, 1000 or 1100. Each of these groups begins with a REM statement that describes how it moves the saucer; for example, up and to the left or down and to the right.

Within each of these small subroutines a line such as 415 checks to see whether the saucer is about to go off the edge of the screen. Line 415, for example, checks to see if the number of the column to which the saucer is about to go is smaller than 33. If the column number is smaller than 33, the saucer will still be on the screen, since the screen is 32 columns wide. In that case the program jumps to line 420, and the loop starts over.

If the saucer is about to go off the edge, however, the program

falls through to line 417. Line 417 simply reverses what happened in line 410, the net result of which is that the value of the variable C is unchanged, and the saucer remains on the same column and does not move. Then line 420 is executed, and we are sent back to the top of the loop.

Conclusion

Do not expect to be able to absorb everything we have covered in this chapter overnight. It has contained a great deal of information, and a lot of new ideas. Go over it a few times; try out the programs; play with them—maybe change lines here and there to see what happens.

When you have digested all this, you will understand many important ideas about programming in TI BASIC. You will be ready to start writing your own programs, limited only by your own imagination.

Good Luck and Have Fun.

APPENDIX

ASCII Codes

32	(space)	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	(delete)

Color Codes

COLOR CODE	COLOR
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Gray
16	White

Character Codes

ASCII CODE	GROUP NUMBER
32-39	1
40-47	2
48-55	3
56-63	4
64-71	5
72-79	6
80-87	7
88-95	8
96-103	9
104-111	10
112-119	11
120-127	12
128-135	13
136-143	14
144-151	15
152-159	16

Editing Function Keys

FCTN E this combination of keys is used to display a program line. Type the line number and then **FCTN E.** The line will appear with the cursor over the first character on the line.

FCTN S moves the cursor to the left within a program line.

FCTN D moves the cursor to the right within a program line.

FCTN 1 erases the character under the cursor and closes the space which is left by moving everything to the right of the cursor one space to the left.

FCTN 2 initiates insert mode. In insert mode, every key press will insert a character under the cursor and move everything to the right of the cursor one space to the right. To get out of insert mode, press either **FCTN D** or **FCTN S.**

FCTN 3 erases the line completely.

FCTN 4 erases anything you have added to the line and ends the edit. To see the line, type its number followed by **FCTN E** again.

Musical Tone Frequencies

Frequency	*Note*
110	A
117	A#, B♭
123	B
131	C (low C)
139	C#, D♭
147	D
156	D#, E♭
165	E
175	F
185	F#, G♭
196	G
208	G#, A♭
220	A (below middle C)
220	A (below middle C)
233	A#, B♭
247	B
262	C (middle C)
277	C#, D♭
294	D
311	D#, E♭
330	E
349	F
370	F#, G♭
392	G
415	G#, A♭
440	A (above middle C)

Musical Tone Frequency (continued)

Frequency	*Note*
440	A (above middle C)
466	A#, B♭
494	B
523	C (high C)
554	C#, D♭
587	D
622	D#, E♭
659	E
698	F
740	F#, G♭
784	G
831	G#, A♭
880	A (above high C)
880	A (above high C)
932	A#, B♭
988	B
1047	C
1109	C#, D♭
1175	D
1245	D#, E♭
1319	E
1397	F
1480	F#, G♭
1568	G
1661	G#, A♭
1760	A

Reserved Words

ABS
APPEND
ASC
ATN
BASE
BREAK
BYE
CALL
CHR$
CLOSE
CON
CONTINUE
COS
DATA
DEF
DELETE
DIM
DISPLAY
EDIT
ELSE
END
EOF
EXP
FIXED
FOR
GO
GOSUB
GOTO
IF
INPUT
INT
INTERNAL
LEN
LET
LIST
LOG
NEW
NEXT
NUM
NUMBER
OLD
ON
OPEN
OPTION
OUTPUT
PERMANENT
POS
PRINT
RANDOMIZE
READ
REC
RELATIVE
REM
RES
RESEQUENCE
RESTORE
RETURN
RND
RUN
SAVE
SEG$
SEQUENTIAL
SGN
SIN
SQR
STEP
STOP
STR$
SUB
TAB
TAN
THEN
TO
TRACE
UNBREAK
UNTRACE
UPDATE
VAL
VARIABLE

GLOSSARY

argument: Some instructions that you give to the computer require extra information, telling the computer something about how it is supposed to obey the instruction. This extra information is called an argument.

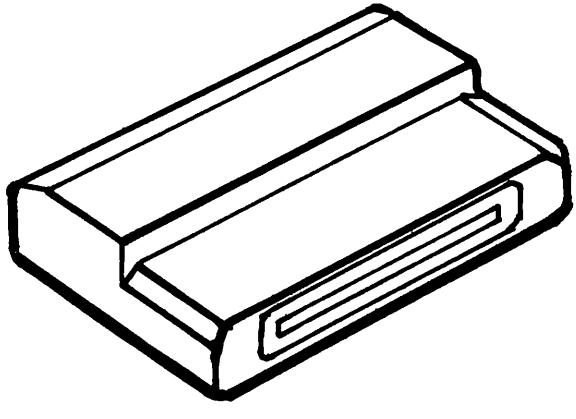
ASCII number: Every character on the keyboard is represented by a number called its ASCII code. For example, the ASCII code for A is 65. The ASCII code for B is 66, and so on.

catalog: a list of all the files on a diskette. Also to catalog a diskette means to list the files on it.

central processor: The brain of a computer, which directs and controls everything that the computer does. In the TI 99/4A the central processor is a single integrated circuit, or "chip', called a microprocessor.

column: on a computer display or print-out, a vertical line.

command module: a small plastic box which plugs into the TI 99/4A and contains a program stored in ROM.



CPU: stands for Central Processing Unit. The same thing as a central processor.

crash: computer jargon for what happens when a program stops running because something is wrong.

cursor: the flashing square or other figure that indicates where on a computer display the next typed character will go.

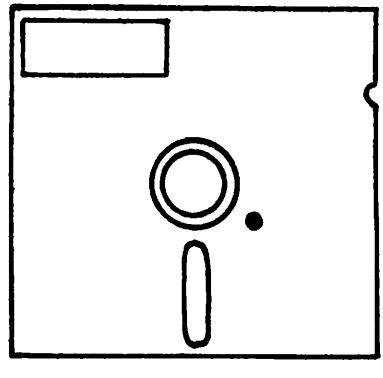
data: information used by a computer, such as programs, numbers, strings and files.

decrement: to reduce the value of a number by one.

default: a value which the computer automatically assigns to some argument, if the user does not assign a value.

deferred mode: when you give the computer several instructions as part of a program so that they are executed one after the other as the program runs rather than at the time they are typed in. Opposite of immediate mode.

disk: a thin, flat, circle of plastic, coated with iron oxide, on which a computer may record information.



diskette: a disk that is 5¼ inches in diameter.

display: pictures or text that a computer prints on a television or monitor screen.

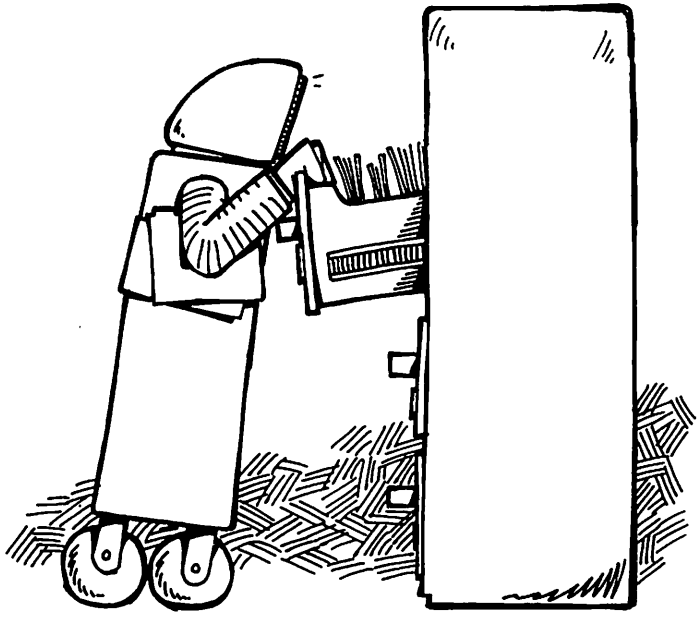
double density: refers to a disk that can hold twice as much data as a single density disk.

double sided: refers to a disk which can have data recorded on both sides.

edit: to change text or program lines by removing characters, adding new characters, moving characters from one place to another or substituting new characters for old ones.

element: If you think of an array as a row of boxes, each box in the array is called an element of that array.

file: a body of information stored in a single place by the computer. Information in a file is all related to a single subject.



floppy disk: a disk that is made of thin plastic and is flexible.

hard disk: a disk that is made of aluminum and is not flexible.

hard-sectored: a disk whose sectors are located by holes in the disk, usually one hole near the center of the disk for each sector. Opposite of soft-sectored.

immediate mode: when you give the computer individual instructions that are executed as soon as you press the ENTER key. Opposite of deferred mode.

increment: to increase the value of a number by one.

initialize: (1) to assign a value to some variable for the first time. (2) to prepare a blank disk for use.

load: to transfer data from a storage device, such as a disk drive or cassette recorder, into a computer's memory.

mass storage: devices where data is stored outside the computer, such as a cassette recorder or a disk drive.

memory: the place inside a computer where data is stored. This is different from mass storage, which is usually outside the computer.

menu: a list of choices from which you may select what you want the computer to do.

monitor: a device like a television, but meant especially for use with a computer. It is used as a television screen for the computer.

null: containing nothing. For example, a null string contains no characters.

peripheral: a device attached to the computer such as a printer or a disk drive.

program: a list of instructions which the computer follows in performing a job.

prompt: a character displayed by a computer to notify the user that an action is expected of him.

RAM: Random Access Memory. That part of a computer's memory whose contents may be changed by the user.

random access: the ability to go directly to data anywhere in a storage medium, as with a disk. The opposite of sequential access.

ROM: Read Only Memory. The part of a computer's memory that contains built-in data which cannot be changed. TI 99/4A command modules contain ROM.

routine: a short program that performs a specific task.

row: a horizontal line of characters on a display or print-out.

save: to record data onto a storage medium such as tape or disk.

sector: a disk is divided up into a number of pie-shaped sections, called sectors. See 'track'.

sequential access: when you are searching for an item of information, and you have to look past all the information that comes before it before you reach the item you seek, as with cassette tape. The opposite of random access.

single density: a disk on which you can record a normal amount of information. See double density.

single sided: when only one side of a disk can be used to record data.

soft-sectored: a diskette whose sectors are located by marks recorded on it, instead of by holes in the diskette. Opposite of hard-sectored.

storage: a place where data is kept. Data may be kept for long periods in mass storage, such as a cassette tape or disk; or it may be kept temporarily in the computer's memory while power is on.

subprogram: a small program that may be part of a larger program. A number of subprograms are built into TI BASIC, and these are run with the CALL instruction.

track: a disk is divided up into a number of concentric rings around the center of the disk. These are called tracks. Each track is divided up into a number of sectors. Each sector of each track holds the same amount of data.

variable: a place in the computer's memory where a number or string is stored.

$8.95

Computer Books

from

ENRICH/OHAUS®

THE GOOD IDEA PEOPLE

Finally, microcomputer handbooks you can understand. Written in everyday language for the beginning beginner, these handbooks...

- Start at the absolute beginning
- Provide the information to make you "Computer Literate"
- Guide you as you discover what your computer can *really* do
- Give you "hands-on" experiences so that you can write your own programs
- Put you in complete control

FUNCTIONAL TOO! The built-in easel allows these books to stand up at your computer for easy use!

Other computer books from ENRICH:

The Illustrated Computer Dictionary & Handbook $9.95

Free Software for Your ATARI $8.95
Free Software for Your Commodore $8.95
Free Software for Your Apple $8.95
Free Software for Your TI $8.95

Apple for the Beginning Beginner $8.95
ATARI for the Beginning Beginner $8.95
PET for the Beginning Beginner $8.95
TRS-80 for the Beginning Beginner $8.95
TI for the Beginning Beginner $8.95

Ask for these books wherever good books are sold!

Printed in the U.S.A.
ISBN: 0-86582-132-1