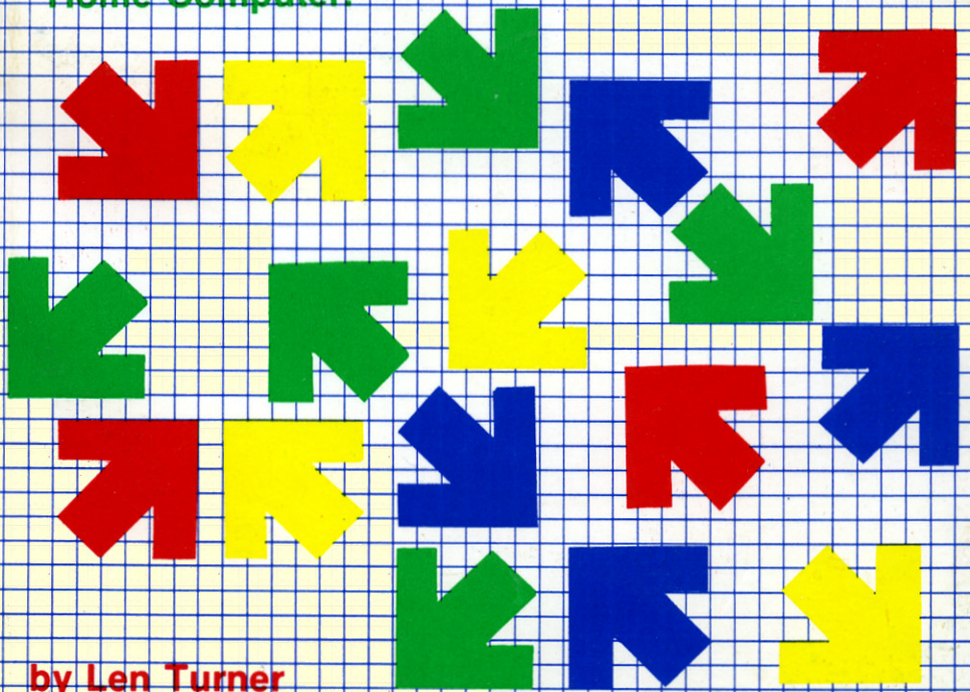
Texas Instruments Home Computer Graphics Programs

An introduction to graphics plus three-dozen ready-to-run programs in BASIC for the TI-99/4A Home Computer.



by Len Turner

Texas Instruments Home Computer Graphics Programs

Programming Books by Len Turner

101 Programming Tips & Tricks for the Texas Instruments TI-99/4A Home Computer

36 Texas Instruments TI-99/4A Programs for Home, School & Office

Texas Instruments Computer Program Writing Workbook

Texas Instruments Home Computer Games Programs

Texas Instruments Home Computer Graphics Programs

Texas Instruments Home Computer Graphics Programs

by Len Turner

ARCsoft Publishers

WOODSBORO, MARYLAND

FIRST EDITION
FIRST PRINTING

ISBN 0-86668-031-4

Preface

Color video graphics is the most exciting artform to come along in this century. It also is the most useful tool in business and education since the printing press.

If a picture is worth a thousand words, a moving picture is worth a thousand still pictures. The video-graphics picture, even when frozen as a single still image, is equivalent to a moving picture because of the ability of the computer to make instantaneous changes, to correct and progress, to add and eliminate.

Businessmen, teachers and students have been heavily impressed with the ability of graphics to transmit endless streams of useful data quickly. Millions have thrilled to the avante garde forms being devised by artists.

All programs in this book were written and thoroughly tested with a Texas Instruments TI-99/4A, a versatile microcomputer system with a small lightweight configuration and a flexible version of the BASIC programming language.

The total number of applications to which the Texas

Instruments home, personal and business microcomputers can be put is limited only by the scope of the imagination. In this book, we have created 38 practical new sets of applications programs for your use. It is hoped that you will, by using these 38 programs, learn how to make your TI computer work for you. You will be able to gain an understanding of how programs work in the computer and how to build on these 38 easy-to-use pieces of software to make your computer do even more work as your understanding grows.

This book, as well as all published by *ARCsoft Publishers*, is written for newcomers, novices and first-timers, as well as for advanced users of microcomputers. Our intention has been to provide easy-to-type-in-and-run programs for the Texas Instruments TI-99/4A, TI-99/2, Compact 40, and other TI personal, home and business microcomputers. You type these programs into your computer and it does the rest. You do not have to be a program writer to use this book!

This volume is a companion book to *101 Programming Tips & Tricks for the Texas Instruments TI-99/4A Home Computer, 36 Texas Instruments TI-99/4A Programs for Home, School & Office, Texas Instruments Computer Program Writing Workbook,* and *Texas Instruments Home Computer Games Programs.*

—Len Turner

Table of Contents

Making Things Move

Borders, Boxes & Billboards

Appendix

Introduction

Computer graphics! The most exciting element of the new world of microcomputers. After word games and number crunching, making colorful video displays can be great fun. And now you can make your own with the 38 programs in this book.

These programs are very useful in themselves. They also make good starting points for further development as you learn more and more about how to program your own computer. You can learn a great deal about how BASIC programs are organized and how they work simply by typing in these programs. Use these fun and practical programs and, then, modify them and expand them to suit your needs as your interests grow.

These programs are designed to be typed into your computer, via its keyboard console, just as you find the programs printed here in this book. No other programming is needed. We assume you have read the owner's manuals and instructional pamphlets which came with your computer and accessories. You know how to hook up the console to the TV modulator/connector and to any

other accessories you may have purchased. You know how to type the programs into your TI computer. You *do not have to be a programmer* to use these pieces of software. Just type them in, as you find them here, and run them. They will work!

These programs do not require tape or disk, unless you choose to save them on those media. These programs are so easy to type in you can save this book and retype them whenever you wish to rerun a program.

Computer printouts

To make sure there are no errors in these programs, we have written and tested each and every program on our own TI-99/4A *and* printed every one on a TI-99 line printer. The hardcopy printout from that line printer is reproduced directly in this book!

The TI computer operated the printer and listed these programs. No human hands came between the computer and these listings so no re-typing or proofreading errors have been introduced. You should find these programs run exactly as reproduced here.

If, after typing in a program from this book, you get an error message from your TI computer, check the handy list of TI BASIC words and other TI info in the Appendix at the back of this book. Then compare your typed program lines with the program lines reproduced in this book.

Undoubtedly, you will find you have made a typing error in entering the program lines into your TI. However, should you find an error in a program in this book, please call it to the attention of the author by sending a postcard or letter to him in care of *ARCsoft Publishers*, P.O. Box 132, Woodsboro, MD 21798 USA. The author will appreciate being able to make any necessary corrections to future editions of this book.

Making things move

This book has been organized into seven convenient sections plus an appendix.

The first three sections are this *Introduction* plus *The BASIC Language* and *Computer Graphics.*

The second three sections compose the main portion of the book and are the reason the book exists. They are

Sketches, Graphs & Stuff including 18 graphics programs; *Making Things Move* with 12 exciting graphics programs; and *Borders, Boxes & Billboards* eight interesting graphics programs.

The appendix includes BASIC words and other useful information which you will want to keep at hand.

Try them all. They're great fun to run. And they are especially designed to be short so you won't have to spend hours typing in one program.

Endless running

Many of the programs in this book will continue to run until you command them off manually via the CLEAR function. You may stop any run, at any time, by use of the CLEAR function.

The function key is in the lower right corner of the console keyboard and is labeled FCTN. Press and hold FCTN and press the number 4 key in the upper row of keys. The combination of FCTN and 4 creates the CLEAR instruction to the computer.

This CLEAR function is the same as what is called BREAK in other microcomputers.

Here is an example of how the CLEAR function works in the TI computer. Type in this brief two-line program. Type in line 10 and press ENTER. Then type in line 20 and press ENTER. This will lodge the complete program in program memory. Here is the program:

```
10 PRINT "XYZ"
20 GOTO 10
```

After you have the program stored in program memory, type in RUN and press ENTER to start the operation. The computer will do as instructed. It will print the letters XYZ repeatedly. In fact, it will go on forever until you stop the action.

To stop the run, press and hold the FCTN key. While holding FCTN down, press the number 4 key. This is the CLEAR function. It will stop the computer run. Try it.

REMarks

As you read through all of the programs in this book, you will notice few REM, or remarks, statements. The

author's training in writing BASIC-language computer programs included an emphasis on brevity and saving of memory space. A sharp editing pencil was in order—and still is!

REMarks and explanations in software are out. Honing, fine tuning, and waste trimming are in. Use of coding-form program-writing worksheets is encouraged. Such worksheets can be found in the *Texas Instrument Program Writing Workbook* published by *ARCsoft Publishers*. Your objective always should be to make the most efficient use of available memory.

Always remember: even though they may be headed toward the same goal, no two programmers will write the exact same list of BASIC instructions, or program lines, from scratch. As you load these various programs into your TI computer, one at a time, you'll make modifications to suit your personal needs and interests. For instance, exact wording of PRINT statements can be changed. Or two or more programs can be combined into one grand scheme. Your applications may vary.

If you want to load more than one of these programs into your TI computer at the same time, be sure to use different sets of line numbers for different programs.

Computer programmers today generally mix the use of the two words, ENTER and RETURN. They are used to mean the same thing. In this case, we mean the ENTER key on the right side of the console keyboard.

Other computers

These programs will run on any computer which is set up to be programmed in BASIC. However, to run these on machines other than ones using TI BASIC as found in the TI-99/4A, you may have to make slight modifications to program lines. Graphic commands, especially, will differ elsewhere. Also use of multiple-statement lines, using the colon (:), is quite different in most other forms of BASIC.

Refer to the owner's manual which came with your non-TI personal computer. Compare its version of BASIC with TI BASIC.

Also, if you use a non-TI microcomputer, such things as line numbering, spacing, logical tests, multiplication

symbols, print statements and other instructions may differ.

Standalone vs. subroutine

All of the programs in this book can be used as portions of larger lists of instructions to your computer. That is, they can be written in as GOTO or GOSUB objects. To do so, make appropriate changes to the first line (usually numbered 10 in this book) and the last line of each program.

If you create a subroutine, remember that every GOSUB must have a RETURN. RETURN must be the last line of each subroutine.

If you work one of these programs into a larger set of instructions, be especially careful of your memory (variable) names or labels. They must agree with, and fit into, those you are using in the main program. Also, be careful of line numbers. No two programs can occupy the exact same set of line numbers.

If you want to load more than one of these programs into your TI computer at the same time, be sure to use different sets of line numbers.

Learning programming

These programs are written to be typed into your TI computer just as you find them here with no programming needed. We assume you know how to turn on your computer and how to go about typing in a program. Many of the programs and much of the programming advice in this book will, in fact, also be of interest to old-timers in the program-writing game since we have presented many powerful new twists aimed at making your computer do more work more quickly.

Use this book to stimulate your thinking about how to approach various software problems and projects. Use it to get good ideas for new and different approaches to all of your programming goals. As you grow and develop as a program writer, modify these programs and make your computer do even more.

Happy programming!

The BASIC Language

Texas Instruments personal computers are practical, useful, fun, even exciting. But writing programs can be a drag unless you know BASIC, the most popular software language. In this book we will introduce you to the T.I. BASIC in an easy-to-understand explanation of the most-used words.

We also provide a graphics grid for use in creating graphics designs for use on the T.I. computer.

Texas Instruments BASIC

Before writing programs for your T.I. be sure you have thoroughly read and understand the owner's manual which came with your computer. It will tell you how to turn the machine on, how to hook up the accessories, how to type in new programs for the computer to run.

The introduction to T.I. BASIC which we offer here will allow you to understand the most elementary workings of your computer and its program language.

The graphics grid is designed for your use in creating

new and different video art on the TV monitor of your T.I. personal computer. Visualize the squares on the grid as if they were the dots you can turn on and off on the face of your T.I. TV picture tube. No matter whether you are working low-resolution, medium-resolution or high-resolution graphics, the sheet will allow you to plan in advance your charts, graphs and other drawings.

This will be a straight-forward introduction to programming. We assume you have tried to read the owner's manual which came with your computer. You know how to turn it on. You know that pushing its buttons can't break it. Don't be afraid to experiment. We'll show you how to make it work for you.

However, the knowledge of BASIC which you will gain from this book will be applicable to *any* microcomputer, minicomputer, or main-frame computer using the BASIC language. And all of today's popular microcomputers use BASIC.

Our simple down-to-earth instruction will help you quickly understand how to talk to your computer and make it do what you want.

The name of the language is BASIC. That stands for *Beginner's All-purpose Symbolic Instruction Code.* What does that mean? Well, you know *beginner.* That's you. *All-purpose* means it's generally useful for lots of different things. *Symbolic* reflects the fact that the comupter uses symbols to receive *instructions* from you. That is, symbols like the word PRINT or IF or THEN or FOR or NEXT. The symbols mostly are words you already know. *Code* is a buzz-word used by programmers to mean instructions to a computer.

So, you can translate *Beginner's All-purpose Symbolic Instruction Code* to mean "You use familiar words to tell your computer how to do just about anything."

BASIC was invented at Dartmouth College in the 1960s to be used by students, beginners, novices, newcomers, to computers and programming. It's very much like everyday English, as you'll see as we go through this book. We'll point out the familiar look-alike words which have

meanings you already know and understand. Words like *end, for, go, to, if, then, list, new, next, step, print, return, run, stop,* and others.

Building on what you already know, we'll show you how the computer receives your instructions and uses them to do what you want.

Universal BASIC

We will use what we consider the most-universal form of BASIC, simplified so it is applicable to just about any contemporary computer—large or small. These words, when used to instruct a computer, would be understood by just about any hardware. Be sure to check your owners' manual to see how its BASIC words differ (if they do) from those we use here.

Keep your owners' manual handy as you type in and run programs. You may need it to make sure you are properly turning on your equipment.

Please remember, no two programmers write identical programs from scratch. Even when working toward the same goal, different writers will create different logic patterns. If your program doesn't exactly match a suggestion in this book, yours still may be correct.

Assuming your program runs and gets the required result, judgment of writing quality should be made on brevity, quickness of running time, and organizational clarity. It's always best to write as few lines as possible. The faster a computer completes its work, the better. And instructions should appear to flow in a logical order so they can be followed by others who might read your writing.

What's Inside Your Computer

There are four main areas : the *input* keyboard, the tiny *microprocessor,* a hulking *memory,* and the *output* display.

Processor, input, output and memory are the important parts of any computer. There are many accessory sections but those four are where the most- interesting activity occurs.

Input and output, often abbreviated as I/O, allow a computer to receive work orders from its operator, to receive information or data for use during a work period, and to send out messages and work results to the operator.

Through the keyboard, an operator gives the computer a list of instructions for carrying out one or more jobs. That list, or *program* of action, is followed by the computer whenever told to do so. It does not have to be acted upon immediately. The program can be remembered for later action.

To achieve some of its work goals, the computer must have additional information or *data.* That information also is typed in through the keyboard.

So, the keyboard has two functions: sending in programs of instructions and sending in additional data.

The output display might be a television set or a TV-like *monitor* or a larger electric typewriter. The output display has one main duty: showing messages and work results to you.

Memory

The convenience of a computer would be lost if we had to send in instructions, one at a time, and await action after each instruction. The beauty of the beast lies in its ability to memorize a long list of instructions and then, upon later command, execute those orders. The computer has a memory to store its various lists of instructions. It is called *program memory* and it can hold more than one complete program at a time.

At the same time, things would be slowed considerably if each extra piece of information has to be keyed in repeatedly every time the computer needed it. The computer can accept data one time and then store it away for repeated use later. To keep such extra information on hand, the computer has *data memory.*

Imagine 26 boxes labeled A through Z. The contents of the boxes can be changed. Some contain something. Some contain nothing. All are *variable* in that their contents can be changed.

Consider each box to be a single memory location, identified by its label A or B or C on through Z.

Strings

The boxes can contain either *numerical* information or words composed of combinations of letters, symbols and even numbers. Such a word is thought of as a *string* of data. Whenever one of our memory location boxes is storing a word, it is a *string variable.* If it holds only numbers, with no letters or other keyboard symbols, it is a *numerical variable.*

The quantity of letters, symbols and numbers which can be tied together in a string and stored in one memory location is limited. In larger desktop computers, one string in one memory location can hold hundreds of characters. But in some computers, one string is limited to seven characters.

This limitation applies only to string variables, not to numerical variables. Here are some examples of what variables contents might look like:

String Variables	**Numerical Variables**
JIM	86
@#$%ABC	1234567890
1/12/83	22.66
BIRTHDY	1

The program writer must keep track of which kind of variable is being used in a particular memory location. For example, if you store a word in location B and then try to use that data in a math problem, an *error* will occur and you'll get a message from your computer.

Only when you have numerical information stored in a memory location can you use that data for math.

One way programmers keep such things straight is by labeling string variables with a dollar sign ($). The dollar sign means *string* and should be read as "string."

Empty boxes

If we were to put a number in A we would label it A. If we were to put a word in A we would label it A$.

By the way, you can change the contents of the various boxes during the running of a program. A location can go from empty to full or from full to empty. Or a full location can have its value changed. A program can be written so the computer will continually check memory locations to see what has been stored there.

Obviously, when we say a memory location is empty we mean it has nothing in it. In effect, it has a big fat zero inside. As a matter of fact, if you were to look at the contents of an empty variable, you would see that it contains a zero. If you ask the computer to show you the contents of an empty memory location, the output display will show Ø if it is a numerical variable. If it is a string variable with nothing stored inside, the display will show nothing. Not even a zero. It will be blank.

You write in data memory by setting the data location letter equal to the value you want to write in it. For instance:

A = 1234

The value on the right is transferred into the storage location on the left.

Program memory

Now you know how to write information in data memory, and recall it. How about writing in program memory?

Your computer is built to use the BASIC program language. BASIC requires each line of a program to start

with a *line number.* Here's a typical three-line program. Notice the numbers at the beginning of each line:

```
10 CALL CLEAR
20 A$="WORD"
30 PRINT A$
```

The computer needs those line numbers to be able to follow your instructions in sequence. It knows that line 20 comes after line 10 and line 30 comes after line 20. Here's the same program with different line numbers:

```
 5 CALL CLEAR
21 A$="WORD"
189 PRINT A$
```

This program will run just the same as the first one. The line numbers are in the same sequence and the commands within each line are the same.

It is possible to write a program which uses every single step of program memory!

NEW

The command NEW erases everything stored in program memory, no matter how many different programs you have there.

The processor

Be an electronic mouse inside the computer again. Notice the master-controller in charge of everything. That's the microprocessor. *Micro* means small. *Processor* means it follows instructions in manipulating data to do work. It's not very big but it sure is powerful!

The processor is a very logical worker, dutifully going about its business in a proper order, carrying out instructions, doing work.

Built into the processor are instructions for how to handle its chores. As it follows that internal set of instructions, it knows how to follow your external set of instructions and do the work you want done.

To make a long story short, the processor takes information from memory, does something with it, and then

either returns data to memory or displays it as output for you to read. It is able to do this many, many times each second and that's why we love the microprocessor!

Suppose you tell the microprocessor to fetch the contents of memory location B. It looks in there and finds WORD there. It *reads* that word, leaves the original behind in memory location B, and takes the information about what is in B away to work with it. The processor actually has a tiny memory inside itself so it can remember what it read in B.

If we instruct the processor to store something in memory location C, it *writes* data to that memory location. When it writes in that memory location, it destroys whatever was there before. For example, suppose we have the number 1234 stored in memory location C. As a result of an operation, we instruct the microprocessor to store the number 6789 in memory location C. It will put 6789 into C and we will lose the original number, 1234, forever.

Remember: reading destroys nothing but writing replaces old information with new.

In carrying out activities, the processor follows exactly the set of instructions you gave it as a *program.* It can't do anything else. If you make a mistake, it makes a mistake. If your work was perfect, its work will be perfect.

Program language

A program is composed of sets of alphabet letters which the processor understands as *words.* A complete set of such words makes up a *language.* BASIC is a language composed of words such as GO, TO, FOR, NEXT, IF, THEN, STEP, PRINT, RETURN, INPUT, PAUSE, WAIT, SET, STOP, END, SAVE, LOAD, GET, PUT, RUN, LIST, NEW and many others.

Since our computers are so very small, they have been given only the very best, most useful, of these words.

The more extensive the BASIC vocabulary, the more flexible the writer can be in creating programs. The total

number of BASIC words invented to date is well over 500. You have the best of these in your computer.

It's easy to see why BASIC is the most popular computer language today. It's most like everyday English and, therefore, most readily used.

Writing and Running Programs

Writing programs means creating line lists of instructions and storing them, one at a time, in program memory.

Running means having the computer recall those sets of instructions, one line at a time, and do them.

RUN

Let's put an instruction in program memory and then run it.

RUN is an instruction to the computer to start at the lowest program line number and begin executing commands it finds there.

You can make the computer start its run at a different line number by typing that line number immediately after the word RUN. For instance, to start at line 100, type:

RUN 100

The computer will skip over any program lines with numbers less than 100.

REMarks

Suppose you were to write a very long, 50-line program of instructions for your computer. You might forget what each line was to accomplish. You need some way to put information in program memory which won't be acted upon by the computer during a run. Information such as notes to yourself so that when you list your program you can recall what the various parts of the program were supposed to do. These notes to yourself, and for other programmers to read, are called *remarks.* The REM command

is used. Anything in a program line after REM will be ignored by the computer during a run. For example:

```
10 REM PRINT "NAME"
20 PRINT "WORD"
```

Type in this program and run it. You'll see that the computer has ignored, or skipped over line 10 and done line 20. Anything on a line after REM is ignored.

REMarks are good for notes but very wasteful of memory. And we don't have much memory to spare in the computer. Use REM infrequently!

BREAK

What to do when your computer goes *blitzo!*

BREAK is used whenever you need to stop a RUN dead in its tracks. It's your panic button.

STOP

But suppose you want the program to STOP automatically at some point in a run? Use the STOP command. Write it into your program as one line.

END

You can, at your option, tell the computer a program has ended. Use the END command.

Input and Output

Input means giving the computer something to store in memory, whether data or program.

Output means displaying messages and work results for you to see.

INPUT

Information can be permanently placed in memory when you write a program. That is, data will actually be part of the program as written. This fixed information could look like this:

```
10  A$="WORD"
```

Whenever you run the program the computer will always start with the memory that WORD is the data in memory location A$.

But, suppose you want the computer to pick up changeable data during a run? Use the INPUT function. Try this program:

```
10  A$="IT IS"
20  INPUT "WHAT IS THE WORD":B$
30  PRINT A$;B$
```

When you run this program, the computer starts at line 10 and stores the string IT IS in memory location A. At line 20, the computer displays the question, WHAT IS THE WORD, and waits for a reply. You type in any string of characters in reply to give your answer to the computer. The computer stores your answer in B$. Then, the computer moves on to line 30 where it recalls the contents of memory locations A$ and B$ and prints them on the display.

The TI-99/4A does not permit multiple statements on one line, as other computers might. The colon, used by other computers to set apart multiple statements on one line, is used for something else in the TI-99/4A.

The colon (:) is used to attach the variable (memory location) name to the end of an INPUT statement. For example, here the prompt in the INPUT statement is the word "NAME" and the memory location is N.

```
10 INPUT "NAME":N
```

Let's see how INPUT works when you want to collect numerical data. It works the same. Try this short program:

```
10  Q=111
20  INPUT "PICK A NUMBER":N
```

```
30  R=Q+N
40  PRINT N;"PLUS ";Q;" = ";R
```

Here, line 10 puts the value 111 into memory location Q. Line 20 displays the message, PICK A NUMBER, and awaits your response. Whatever number you select, key it in. The computer will store your number in memory location N.

Line 30 does the math work for you by adding. It recalls that 111 was stored in Q and your number was stored in N. It adds those two values to get a new total. The total is stored in memory location R. The program moves on to line 40.

At line 40 the computer prints the results in sentence form. Try it with several different numbers. It's fun!

Suppose your number were 59. The program result, after printing line 40, would look like this:

```
59 PLUS 111 = 170
```

You don't have to use the message part of the INPUT function if you don't want to. For instance:

```
10  INPUT N
20  INPUT P
30  PRINT N
40  PRINT P
```

This program allows the computer to take in your numerical data and store it in memory locations N and P and then print the values on the display. The computer will start at the lowest line number, as usual, line 10. Since no message has been supplied, the computer will display only a question mark (?). The ? tells you the computer wants some information. Try it on your computer.

PRINT

You already have used the PRINT output command but here's some further information.

PRINT causes a message to be displayed on the computer's display. The printed message consists of whatever is contained within the quotation marks following the PRINT command. For instance:

```
10  PRINT "I LIKE ICE CREAM"
```

The computer reproduces exactly what you place between the quotes, including blank spaces. Try it in your computer. Now, type in this program:

```
10  PRINT "I LIKE ICE CREAM"
20  PRINT "DO YOU?"
```

These PRINT messages need not be in the same line as the PRINT command, by the way. Rather, you can store a message in data memory and recall it for PRINTing. For example:

```
10  N = 1234.56789
20  PRINT N
```

The computer, at line 10, stores the number 1234.56789 in memory location N. At line 20, the computer recalls the value of N and prints it on its display. Here's another example:

```
10  G$ = "WORD"
20  PRINT G$
```

Here the computer stores the string of characters, WORD, in memory location G. At line 20 it recalls G$ and prints it.

Here's an even more complex program:

```
10  A = 6
20  B = 7
30  C = 2
40  D = A + B + C
50  PRINT D
```

The computer stores the number 6 in memory location A; the number 7 in location B; and 2 in C. At line 40 it recalls

the values in A, B and C and adds them together. The result of that addition is stored in D. Line 50 recalls the contents of D and prints the number on the display. Try it.

The Real Computer Power!

When folks talk about a computer having power, they often are referring to its ability to make decisions. And its looping ability. And its jumping ability. These capacities, when combined, make for some very powerful computing ability.

FOR/NEXT/STEP

You already know *loops* are fun but we need a way to control them to put them to a useful purpose. Here's one way:

```
10  FOR L = 1 TO 100
20  PRINT  L
30  NEXT L
40  PRINT "END OF COUNT"
```

Lines 10 and 30 create a FOR/NEXT loop. A FOR/NEXT loop probably is the most frequently used of the super-power BASIC commands.

In this program, line 10 actually contains a built-in counter which advances the value stored in L by one every time the program reaches line 30. In fact, until the count reaches 100, line 30 causes the program to jump back to line 10. When the value in L reaches 100, then and only then will the FOR/NEXT loop let the action drop on down to line 40. Here's a variation:

```
10  FOR A = 10 TO 100
20  PRINT  A
30  NEXT A
```

The memory location used in the loop can be any of those available to you in your computer.

Unless you tell it otherwise, the count will step up by ones. Try this change:

```
10  FOR X=2 TO 40 STEP 2
20  PRINT X
30  NEXT X
```

Here the count goes up by twos. Try this program to make the computer count down by ones:

```
10  FOR J=100 TO 1 STEP -1
20  PRINT J
30  NEXT J
```

The computer starts at 100 and counts down to 1, and then stops. Very convenient. Very powerful!

The STEP statement is not used unless you want increments other than +1. Minus numbers after STEP will cause the computer to count down in numbers while positive numbers will cause it to count up. Now make the computer take some giant steps:

```
10  FOR R=999 TO 1 STEP -100
20  PRINT R
30  NEXT R
```

The computer counts down by hundreds. At that rate, it doesn't take very long to run out of numbers.

Sometimes you need a time delay in the middle of a program as it is running. The loop can be used to create such a time delay.

```
10  FOR N=1 TO 999
20  NEXT N
```

Get a stopwatch and keep an eye on the running time for the program. Line 10 is a FOR/NEXT loop all on one line, without any output during the loop. The computer merely counts internally up to 999 and then moves on.

How long does it take such a loop to run its course? Use a stopwatch to time it. A nice long delay! Now try counting to 100:

```
10  FOR N=1 TO 100
20  NEXT N
```

How long is the delay?

```
10  FOR N=1 TO 10
20  NEXT N
```

Counting only to 10 reduces the delay.

```
10  FOR N=1 TO 5
20  NEXT N
```

Counting only to 5 makes things happen even more quickly.

```
10  FOR N=1 TO 3
20  NEXT N
```

Why is a one-second loop useful? Well, maybe you would like to turn your computer into a clock!

Here's a simple timer, for starts:

```
10 CALL CLEAR
20  T=T+1
30  FOR N=1 TO 3
40  NEXT N
50  PRINT T;"SECONDS"
60  GOTO 20
```

This is a crude clock. You can adjust its speed by changing the number 3 in line 30. It will count seconds until you stop it with the BREAK key.

Can you figure out why it takes a bit longer for the first display, 1 SECONDS, to appear? Because the computer uses up time as it works it way through lines 10, 20 and 30. You planned on it using up time at line 30 but you may have overlooked the amount of time it takes to carry out the instructions at line 10 and line 20.

IF/THEN

Did we say earlier the computer has the ability to make decisions? Yes! The IF/THEN statement is an important part of the decision-making process.

IF something happens or is true, THEN and only then will something else happen. IF nothing happens or something is not true, THEN nothing will happen. The IF/THEN test is one of the superpowers of the computer. IF something is true, THEN some action is taken. That action can be a GOTO jump to a new line.

IF/THEN statements must end with a line number, for example:

```
10 IF A=B THEN 200
```

Try this brief program. In it, the computer displays 10 if you

type in the number 10 in reply to its question. If you type any other number, it will display zero:

```
10  CALL CLEAR
20  INPUT "WHAT IS THE NUMBER":N
30  IF N=10 THEN 100
40  PRINT "0 "
50  PRINT
60  PRINT
70  GOTO 20
100 PRINT N
110 GOTO 50
```

At line 10, the entire video screen is cleared. At 20, the computer asks you to type in a number. You type in a number and press ENTER. At line 30, the computer makes a decision. It does this by testing the value you placed in memory location N at line 20. If N is found to be equal to 10 then program action jumps to line 100. If not, action drops to line 40 where the computer prints a zero. Lines 50 and 60 cause the computer to print two blank lines and then line 70 pushes action back to line 20 where the question is repeated.

GOTO

You know that the computer does your list of BASIC instructions by following line numbers. First it does line 10, then line 20, etc. But, suppose you want the computer to do things in a different order. Maybe you would like it to *jump* over a group of lines. Or skip down to a different part of the program. This ability to *branch* out and around some lines to do other lines is an important power in the computer. It involves the GOTO and GOSUB statements.

GOTO means "go to a line." The GOTO statement must include the destination where you wish the program to go. For example:

```
GOTO 100
```

When the computer finds a GOTO statement, it immediately leaves the list, searches for and finds the destination line, and reenters operations at that point. Here's a small example:

```
10 GOTO 30
20 PRINT "NAME"
30 PRINT "WORD"
```

In this program, the computer starts at line 10 where it immediately finds a command to GOTO line 30. It skips down the list until it finds line 30. At line 30 it resumes doing what you asked. It prints WORD. In this case, the instruction in line 20 never gets done.

You can jump backward and forward within the program. Here's an example:

```
 10 INPUT "ENTER A NUMBER":A
 20 INPUT "ENTER ANOTHER NUMBER":B
 30 GOTO 100
 40 PRINT"THE TOTAL IS ";T
 50 GOTO 10
100 T=A+B
110 GOTO 40
```

Again the program starts running at the lowest line number, line 10. At line 10 it asks you for a number which it stores in memory location A. At line 20 it asks for another number which it puts in B.

At line 30 it finds an order to branch down to line 100 which it does. When it finds line 100 it does the instruction in line 100. It recalls the contents of A and B and adds them together, storing the total in T. Having completed line 100, it moves on down to line 110.

At line 110 the computer finds your instruction to jump back up to line 40. Doing that, it finds at line 40 an instruction to print THE TOTAL IS and the value in T. Putting that message on the display, it goes on to line 50.

At line 50 it comes upon your command to go up to line 10. It does that, thereby starting the entire process over again. The computer will go through this elaborate loop as long as you are willing to keep giving it numbers.

GOTO is, in fact, one of the most-used words in the BASIC language. Our programs are strewn with such jumps.

GOSUB/RETURN

Often you will need to repeat the exact same set of instructions at different points in a program. You could type the required program lines into the program each time they are needed. Or you can type them once and make the program jump to them when needed.

Typing of repeating sequences wastes your program-writing time, and, more importantly, wastes program memory space. It's easier for you and uses less memory when you create one *subroutine* to be repeatedly used by the computer.

Why not use a GOTO statement to get to a subroutine? The answer lies in the RETURN from the subroutine. If you were to use GOTO to get to a subroutine from several different places in a program, the designation of where to return to after completion of the subroutine would be long and clumsy. GOSUB was invented to take care of just that problem.

A subroutine is a small program which you can imagine as being set aside from the main program. A subroutine can be used as often as you like while running the main program. Each time a subroutine is completed, the computer automatically returns to the line in the main program immediately following the line from which it earlier had left the main program. Here's a small example:

```
10  A=555
20  GOSUB 100
30  PRINT T
40  END
100 T=A+1
110 RETURN
```

The main program is contained in lines 10, 20, 30 and 40. The subroutine is lines 100 and 110. The jump to the subroutine is the instruction in line 20. Note that it contains the destination line number. The return from the subroutine is from line 110 to line 30.

At line 10, we assign the value 555 to memory location A. At line 20, we ask the computer to branch to the subroutine at line 100.

At line 100 the computer finds an instruction to recall the value of A and add one to that value. The new total is stored in memory location T.

The program moves on to line 110 where it finds RETURN. That instruction, which must *always* be at the end of a subroutine, tells the program to jump back to the line immediately following the line where it left the main program. In this case, the program left the main routine at line 20 so RETURN will kick it back to line 30.

At line 30 the computer finds a command to recall the contents of T and to display it. It does that and moves on to line 40. At line 40 it finds the END command and ceases operations.

Why an END in line 40? Because you need to make sure the subroutine is entered only from the GOSUB instruction. After line 30, without an END in line 40, the program would automatically move from line 30 to the next available higher line number which is 100. At line 100 it would enter the subroutine. At line 110 it would find a RETURN which did not come from a GOSUB and an error message would occur.

Just as a GOSUB must have a RETURN, the RETURN statement must come after a GOSUB.

The computer has a tiny private "scratchpad" bit of memory within itself where it writes temporary notes to itself. When it executes a GOSUB command, it makes note of the line number from which it left the main program. Later, when it finds a RETURN, it refers to its scratchpad to see where it left the main program. It determines the next available program line after that exit point and re-enters the main body of the program at that point.

If the computer encounters a RETURN without having left the main program via GOSUB, it won't be able to find a "where to" note on its scratchpad and will send you an error message. You don't want error messages so you prevent the computer from getting into subroutines by means other than GOSUB jump commands.

Computer Graphics

Your personal computer is a system with four major parts: input, processor, memory and output.

Processor and *memory* are the innards, the brain which does the internal work you ask for.

Input is composed of the various parts of the equipment which allow you talk to the computer, to send in information for the memory to store and for the processor to work on. Input includes the typewriter-style keyboard, a tape, a disk, etc.

Output is the equipment available for the computer to talk back to you, to report the results of work you asked it to do. Output includes the video display screen, a line printer, or other devices.

This is concerned with a special use of one piece of output equipment, the video display screen. We hope you will learn from these pages how to make the computer display useful pictures on the face of the video tube.

When you turn the power on, the computer knows how to operate because the manufacturer has written

software and inserted it into the computer's innards. That internal program is *system software*.

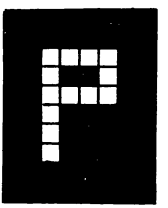
The computer can go beyond its basic internal housekeeping functions to do real-world jobs you ask of it because you write additional programs for it to follow. Your added instructions are *applications software*.

This , then, will show you how to write applications software especially to create pictures on the video display.

You hear a lot of talk, these days, about various types of *resolution*. Some graphics are said to be *low-resolution*. Some are *high-resolution*. There is a middle ground which could be thought of as *medium resolution*. What's the difference?

Low vs. high resolution

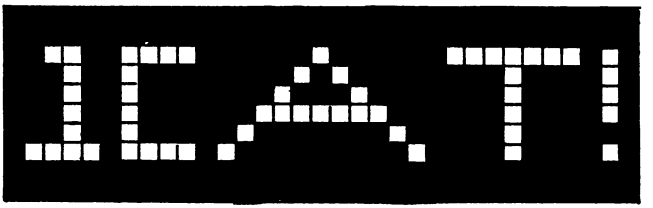
Letters, symbols, numbers, entire words, pictures, charts, graphs, anything displayed on the face of your TV screen or video display monitor is created as a series of lighted dots against a dark background. Imagine your TV screen as a large grid of tiny square rectangles like a piece of graph paper. Suppose you wanted to create the letter P on that grid, as in this approximate drawing:



The overall screen is dark. The light spots, when viewed together, create the image of the letter P. Your education leads you to see the letter P rather than an assortment of 13 white spots against a black background.

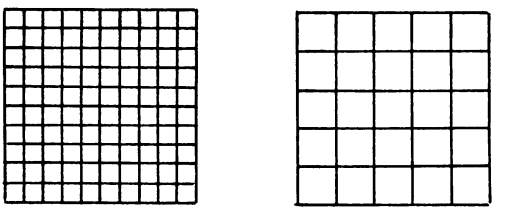
To create the letter P on the face of your TV, the computer lights several small rectangular dots in a pattern you recognize as P. The same for the letters C and A and

T, the number 1 or the symbol we call an exclamation point or any others you can think of:



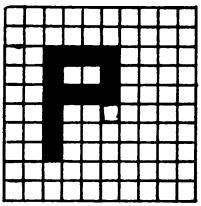
The size of the face of a TV set is fixed, but it is possible to make the lighted dots larger or smaller. The smaller the dot, the more dots we can squeeze onto the face of the video screen. Like creating graph paper with ever-smaller squares, the more dots we squeeze onto the face of the video tube, the less likely you are to be able to see any one dot.

Fewer dots filling a screen mean each dot is bigger, more easily seen. More dots filling a screen mean smaller dots, each less easily seen. For example, look at these two grids. Each is the same size. But one has twice as many small squares in it.

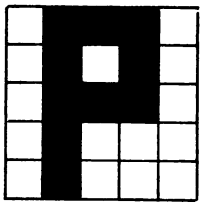


Let's try our letter P in each of two grids. The P on the left, below, contains more dots. We'll call it "high resolution" since it has a higher number of dots in the same space.

The P on the right contains fewer dots. We'll call it "low resolution" since it contains a lower number of dots in the same space:



High Resolution



Low Resolution

If we had a P with more dots than in our low-resolution P, but with fewer dots than in our high-resolution P, we would have a medium-resolution P.

All information transmitted to you from the computer on the video screen is created the same way, as a pattern of lighted dots.

Text vs. graphics mode

Text mode is used for common letters, numbers, symbols, words, formulas and other kinds of frequently-used English-language communication. In the text mode, the computer calls upon data imbedded in its permanent memory to create the patterns of lighted dots we will recognize as letters of the alphabet or numbers or symbols.

The quantities and descriptions of those patterns of lighted dots are previously established inside the computer and beyond your control. Call for the letter A and you'll always get the same A. You cannot make that text-mode A short-legged or fatter or slimmer. In text mode, an A is an A is an A...

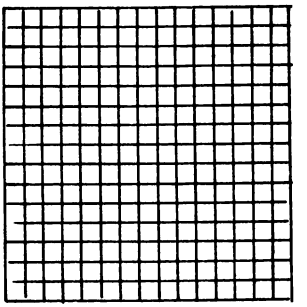
Graphics mode, on the other hand, is your own personal sketch pad. You can draw shapes and sizes of all sorts of characters and figures to suit your own desires.

When you turn power on, your computer wakes up in the text mode. Many of the BASIC words you use in programs automatically create text displays. For instance, use of the PRINT instruction makes a text display.

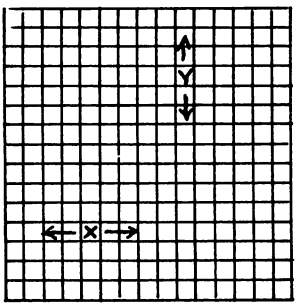
Video graph paper

Remember we said the TV screen can be imagined as having a grid like graph paper? Well, like graph paper you

can precisely locate one spot on the face of the screen by counting rows and columns. Here's a grid:



Now, suppose we thought of all the horizontal rows as X and the vertical columns as Y. We might think of lines moving across the TV screen as moving in the X direction and lines moving up and down the screen as moving in the Y direction.



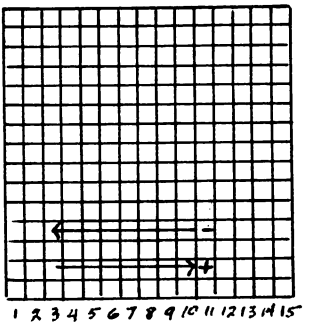
The upper left corner of the screen is position X = 1 and Y = 1. There are 32 columns across the screen and 24 rows down the screen. Thus, X can range from 1 to 32 and Y from 1 to 24.

Count the dots across the grid. Start on the left and count toward the right. As you move toward the right hand side of the grid you get more and more dots. The number of dots is increasing. Each new dot adds one to the total. Each new dot is *plus* one.

Now move backward, right to left. Each new dot sub-

tracts one from the total previously counted. Each is *minus* one.

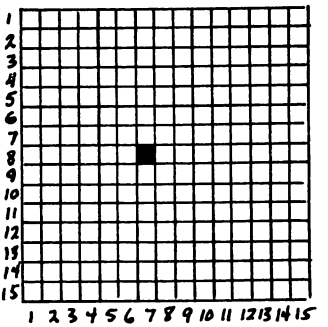
To move left to right, then, add one to the value of X. To move right to left, subtract one from the value of X.



Similarly, to go up or down the screen, the value of Y changes.

You will note that the position where X,Y is 1,1 is in the upper left hand corner of the grid. What would the lower left hand corner be? Since it is in the fifteenth position for both X and Y it would be 15,15.

Any position on the screen can be located as an X,Y point. For instance 1,1 or 15,15 or 7,8. Where is 7,8?



These X and Y values are used in the CALL GCHAR, CALL HCHAR and CALL VCHAR commands in the TI-99/4A.

Sketches, Graphs & Stuff

Drawing Sketches

Now you can draw lines, rules, diagrams, maps, charts, boxes—anything you can imagine—on the face of your color TV set. Use the Computer keyboard as your pen and its video output as your ink.

Lines 50 to 390 accept your up, down, right, or left commands, as U, D, R, or L. No other letters will work. Line 400 draws your lines.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 R=1
40 C=1
50 CALL KEY(0,Z,X)
60 IF X=0 THEN 50
70 IF Z=85 THEN 90
80 GOTO 150
90 R=R-1
100 IF R<1 THEN 120
110 GOTO 140
120 R=1
130 GOTO 50
140 GOTO 400
150 IF Z=68 THEN 170
160 GOTO 230
170 R=R+1
180 IF R>24 THEN 200
190 GOTO 220
200 R=24
210 GOTO 50
220 GOTO 400
230 IF Z=82 THEN 250
240 GOTO 310
250 C=C+1
260 IF C>32 THEN 280
270 GOTO 300
280 C=32
290 GOTO 50
```

```
300 GOTO 400
310 IF Z=76 THEN 330
320 GOTO 390
330 C=C-1
340 IF C<1 THEN 360
350 GOTO 380
360 C=1
370 GOTO 50
380 GOTO 400
390 GOTO 50
400 CALL HCHAR(R,C,128)
410 GOTO 50
```

Color Bar Graph Generator

The bar graph generated by this program can have up to 20 bars. The lengths of the bars are limited to the range of zero to 23. You type in the letter X to end the input loop. The bars are numbered sequentially, from top to bottom, starting with number one.

Program Listing

```
10 CALL CLEAR
20 DIM R$(20)
30 DIM NM$(20)
40 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
50 PRINT "WHAT IS THE TITLE"
60 INPUT "OF THE CHART?  ":T$
70 N=N+1
80 PRINT
90 INPUT "BAR LENGTH (1-23)?  ":R$(N)
100 IF R$(N)="X" THEN 160
110 IF VAL(R$(N))>23 THEN 130
120 GOTO 150
130 PRINT "OOPS, TOO LONG, TRY AGAIN"
140 GOTO 90
150 GOTO 180
160 R$(N)="0"
170 GOTO 210
180 NM$(N)=STR$(N)
190 IF N+1=21 THEN 210
200 GOTO 70
210 CALL CLEAR
220 CALL SCREEN(5)
230 FOR W=1 TO 13
240 CALL COLOR(W,16,5)
250 NEXT W
260 LT=LEN(T$)
270 TP=INT((32-LT)/2)
280 PRINT TAB(TP);T$
290 PRINT
300 FOR K=1 TO N
310 PRINT NM$(K);" ";
```

```
320 NR=VAL(R$(K))
330 PRINT TAB(4);" ";
340 FOR J=1 TO NR
350 PRINT CHR$(128);
360 NEXT J
370 PRINT
380 NEXT K
390 CALL KEY(0,Z,X)
400 IF X=0 THEN 390
410 FOR P=1 TO 20
420 R$(P)=""
430 NM$(P)=""
440 N=0
450 NEXT P
460 CALL CLEAR
470 CALL SCREEN(4)
480 FOR W=1 TO 13
490 CALL COLOR(W,2,1)
500 NEXT W
510 GOTO 10
```

Draw Bar Graphs

Drawing graphs on the video screen are a popular form of communication today. This program establishes a bar graph on the computer display.

We have selected the business-like example, shown here, to demonstrate how you go about setting up a bar graph on the TV screen.

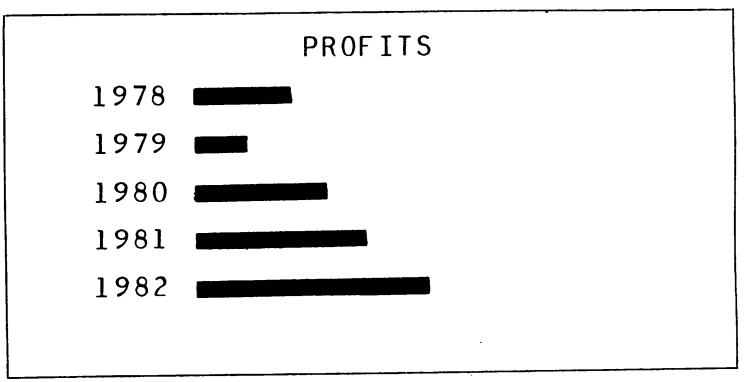
After a run, the computer awaits your press of any key to do another.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 INPUT "1978 PROFITS? ":A
40 IF A>23 THEN 30
50 INPUT "1979 PROFITS? ":B
60 IF B>23 THEN 50
70 INPUT "1980 PROFITS? ":C
80 IF C>23 THEN 70
90 INPUT "1981 PROFITS? ":D
100 IF D>23 THEN 90
110 INPUT "1982 PROFITS? ":E
120 IF E>23 THEN 110
130 CALL CLEAR
140 PRINT TAB(11);"PROFITS"
150 PRINT
160 PRINT
170 PRINT "1978 ";
180 FOR L=1 TO A
190 PRINT CHR$(128);
200 NEXT L
210 PRINT
220 PRINT
230 PRINT "1979 ";
240 FOR L=1 TO B
250 PRINT CHR$(128);
260 NEXT L
270 PRINT
280 PRINT
```

```
290 PRINT "1980 ";
300 FOR L=1 TO C
310 PRINT CHR$(128);
320 NEXT L
330 PRINT
340 PRINT
350 PRINT "1981 ";
360 FOR L=1 TO D
370 PRINT CHR$(128);
380 NEXT L
390 PRINT
400 PRINT
410 PRINT "1982 ";
420 FOR L=1 TO E
430 PRINT CHR$(128);
440 NEXT L
450 PRINT
460 CALL KEY(0,Z,X)
470 IF X=0 THEN 460
480 GOTO 10
```

Sample Run



Aztec Art

Our program reminded us of *Aztec* artwork.

Program Listing

```
10 CALL CLEAR
20 CALL SCREEN(11)
30 FOR X=10 TO 1 STEP -1
40 FOR N=1 TO 12
50 CN=INT(126*RND)
60 IF CN<33 THEN 50
70 R=12-(X*SIN(N/6*3.14))
80 C=16-(X*COS(N/6*3.14))
90 CALL HCHAR(R,C,CN)
100 NEXT N
110 NEXT X
120 GOTO 120
```

Aztec Art II

Program Listing

```
10 CALL CLEAR
20 RANDOMIZE
25 WW$="FFFFFFFFFFFFFFFF"
27 CALL CHAR(128,WW$)
30 FOR X=10 TO 1 STEP -1
40 FOR N=1 TO 12
70 R=12-(X*SIN(N/6*3.14))
80 C=16-(X*COS(N/6*3.14))
90 CALL HCHAR(R,C,128)
100 NEXT N
110 NEXT X
120 GOTO 120
```

Checkerboard

Program Listing

```
10 CALL CLEAR
20 W$="00000000000000"
30 B$="FFFFFFFFFFFFFFFF"
40 CALL CHAR(128,W$)
50 CALL CHAR(129,B$)
60 CALL COLOR(13,2,16)
100 FOR Y=4 TO 19
110 FOR X=8 TO 23
120 CALL VCHAR(Y,X,128)
140 NEXT X
150 NEXT Y
200 FOR Y=5 TO 19 STEP 2
210 FOR X=9 TO 23 STEP 2
220 CALL VCHAR(Y,X,129)
225 CALL VCHAR(Y-1,X-1,129)
230 NEXT X
240 NEXT Y
300 GOTO 300
```

Reverse Checkerboard

Program Listing

```
10 CALL CLEAR
20 W$="FFFFFFFFFFFFFFFF"
30 B$="0000000000000000"
40 CALL CHAR(128,W$)
50 CALL CHAR(129,B$)
60 CALL COLOR(13,2,16)
100 FOR Y=4 TO 19
110 FOR X=8 TO 23
120 CALL VCHAR(Y,X,128)
140 NEXT X
150 NEXT Y
200 FOR Y=5 TO 19 STEP 2
210 FOR X=9 TO 23 STEP 2
220 CALL VCHAR(Y,X,129)
225 CALL VCHAR(Y-1,X-1,129)
230 NEXT X
240 NEXT Y
300 GOTO 300
```

Clock Exercise

More round graphics on your rectangular picture tube.

Program Listing

```
10 CALL CLEAR
20 FOR N=1 TO 12
30 IF N>3 THEN 60
40 CX=49
50 GOTO 70
60 CX=32
70 CN=N+45
80 IF CN<49 THEN 100
90 GOTO 110
100 CN=CN+2
110 R=12-(10*SIN(N/6*3.14))
120 C=16-(10*COS(N/6*3.14))
130 CALL HCHAR(R,C-1,CX)
140 CALL HCHAR(R,C,CN)
150 NEXT N
160 GOTO 160
```

Blue Fence Builder

Program Listing

```
10 CALL CLEAR
20 D$="FFFF81818181FFFF"
40 CALL CHAR(128,D$)
70 FOR Y=1 TO 24
80 FOR X=1 TO 32
110 CALL COLOR(13,5,16)
120 CALL VCHAR(Y,X,128)
130 NEXT X
140 NEXT Y
200 GOTO 200
```

Multicolor Fence Builder

Program Listing

```
10 CALL CLEAR
20 D$="FFFF81818181FFFF"
40 CALL CHAR(128,D$)
70 FOR Y=1 TO 24
80 FOR X=1 TO 32
90 C=INT(16*RND)+1
100 IF C<2 THEN 90
105 IF C=4 THEN 90
110 CALL COLOR(13,C,4)
120 CALL VCHAR(Y,X,128)
130 NEXT X
140 NEXT Y
200 GOTO 200
```

Window Twinklers

Well, what would you call them?

Program Listing

```
10 CALL CLEAR
20 FC=INT(17*RND)
30 IF FC<3 THEN 20
40 CALL COLOR(1,FC,5)
50 CH=INT(38*RND)
60 IF CH<33 THEN 50
70 C=INT(33*RND)
80 IF C<1 THEN 70
90 R=INT(25*RND)
100 IF R<1 THEN 90
110 CALL HCHAR(R,C,CH)
120 GOTO 20
```

Five-Item Checklist

The computer asks you to give it the names of five items. It then prints them in a column with small check-off boxes alongside.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FF8181818181FF")
30 DIM I$(5)
40 INPUT "FIRST ITEM:  ":I$(1)
50 INPUT "SECOND ITEM: ":I$(2)
60 INPUT "THIRD ITEM:  ":I$(3)
70 INPUT "FOURTH ITEM: ":I$(4)
80 INPUT "FIFTH ITEM:  ":I$(5)
90 PRINT
100 PRINT
110 PRINT
120 PRINT
130 PRINT "CHECKLIST"
140 FOR L=1 TO 9
150 PRINT CHR$(126);
160 NEXT L
170 PRINT
180 FOR L=1 TO 5
190 PRINT CHR$(128);" ";I$(L)
200 NEXT L
210 CALL KEY(0,Z,X)
220 IF X=0 THEN 210
230 GOTO 10
```

Sample Run

```
☐ PENCIL
☐ PAPER
☐ INK
☐ PEN
☐ NOTEBOOK
```

- [ ] RADIO
- [ ] TELEVISION
- [ ] NEWSPAPER
- [ ] MAGAZINE
- [ ] BOOKS

- [ ] DOG
- [ ] CAT
- [ ] HORSE
- [ ] COW
- [ ] SHEEP

- [ ] RAIN
- [ ] SNOW
- [ ] SLEET
- [ ] HAIL
- [ ] SLUSH

- [ ] HAT
- [ ] COAT
- [ ] GLOVES
- [ ] SCARF
- [ ] BOOTS

Backward Writer

Give the computer a message of up to 99 characters. The computer will print the message backward. Then press R to repeat the same message. Or press W to start over with a new message.

The computer is simply amazing!

Program Listing

```
10 CALL CLEAR
20 DIM X$(100)
30 PRINT "TYPE A MESSAGE"
40 INPUT A$
50 IF LEN(A$)>100 THEN 70
60 GOTO 90
70 PRINT "OOPS, TOO LONG, TRY AGAIN"
80 GOTO 30
90 L=LEN(A$)
100 CALL CLEAR
110 FOR J=L+1 TO 1 STEP -1
120 X$(J)=SEG$(A$,J,1)
130 PRINT X$(J);
140 NEXT J
150 FOR Q=1 TO 10
160 PRINT
170 NEXT Q
180 PRINT "PRESS R TO REPEAT BACKWARD"
190 PRINT "PRESS M TO SEE ORIGINAL"
200 PRINT "PRESS W TO WRITE NEW"
210 PRINT "PRESS Q TO QUIT"
220 CALL KEY(0,Z,X)
230 IF X=0 THEN 220
240 IF Z=82 THEN 100
250 IF Z=77 THEN 300
260 IF Z=87 THEN 10
270 IF Z=81 THEN 370
280 CALL CLEAR
290 GOTO 180
300 CALL CLEAR
310 PRINT A$
```

```
320 FOR G=1 TO 10
330 PRINT
340 NEXT G
350 GOTO 180
360 GOTO 10
370 CALL CLEAR
380 PRINT "BYE BYE"
```

Show The Colors

This demonstrator program shows each of the available screen colors slowly. As they pass in review, watchers learn what is available. For a longer time of display of each color, increase the number 500 in line 40.

Program Listing

```
10 CALL CLEAR
20 FOR C=3 TO 16
30 CC=C
40 FOR T=1 TO 500
50 NEXT T
60 IF C>9 THEN 80
70 GOTO 90
80 CC=CC-10
90 X=32
100 IF C>9 THEN 120
110 GOTO 130
120 X=49
130 CALL SCREEN(C)
140 CALL HCHAR(1,15,X)
150 CALL HCHAR(1,16,CC+48)
160 NEXT C
170 GOTO 20
```

Snowfall

White flakes sprinkle down the screen, over and over—until you press the BREAK key. It may be useless but it's a lot of fun to watch!

Program Listing

```
10 CALL CLEAR
20 CALL SCREEN(5)
30 CALL COLOR(2,16,5)
40 R=INT(25*RND)
50 IF R<1 THEN 40
60 C=INT(32*RND)
70 IF C<1 THEN 60
80 CALL HCHAR(R,C,42)
90 GOTO 40
```

Flashing Graphics Cursor

You can make any one spot on the face of your television set, or video-display tube, dance or glitter with color using this program.

Use this flashy little indicator to spot whatever you like on the graphics screen. Change the location of the cursor spot by changing the two 12s in line 50.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 FOR C=3 TO 16
40 CALL COLOR(13,C,4)
50 CALL HCHAR(12,12,128)
60 NEXT C
70 GOTO 30
```

Screen Filler

Some say it looks like Outer Space. Maybe a view of Earth from out there? Whatever, it makes a fun, colorful display.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 CALL SCREEN(10)
40 C=INT(33*RND)
50 IF C<1 THEN 40
60 R=INT(25*RND)
70 IF R<1 THEN 60
80 CALL HCHAR(R,C,128)
90 GOTO 40
```

Super Slot-0

Oh, those evil slot machines! They're just popping up everywhere. Even inside my TI Computer.

As with all the programs used as examples in this book, simply type this one in and RUN it. The computer will display, on your video screen, the name of this program and some simple instructions.

Like any good slot machine, when you pull the handle it displays some objects. If you get no two alike, you lose. If you get two alike among the three objects, you win small. If all three are the same, you win big.

To simulate pulling the slot machine's lever arm, press the ENTER key on the keyboard.

One difference in our Slot-O game, the display is entirely at random. No one pushes a secret button under the table to make certain items pop up.

Get out your funny-money from that old Monopoly game, gather up your friends, and let's have some fun.

Program Listing

```
10 CALL CLEAR
20 GOSUB 500
30 PRINT
40 PRINT
50 PRINT
60 GOSUB 200
70 PRINT "***** ***** ***** *****"
80 PRINT "* ";A$;" * * ";B$;" * * "
   ;C$;" * * ";D$;" *"
90 PRINT "***** ***** ***** *****"
100 PRINT
105 PRINT
110 PRINT "TO PULL THE LEVER,"
120 INPUT "PRESS ENTER":KY$
130 GOTO 10
200 GOSUB 400
210 A$=CHR$(X)
220 GOSUB 400
230 B$=CHR$(X)
240 GOSUB 400
```

```
250 C$=CHR$(X)
260 GOSUB 400
270 D$=CHR$(X)
280 GOSUB 400
400 R=INT(5*RND)
410 IF R<1 THEN 400
420 IF R=1 THEN 800
430 IF R=2 THEN 900
440 IF R=3 THEN 1000
450 IF R=4 THEN 1100
460 RETURN
500 PRINT "**********************"
510 PRINT "* SUPER T.I. SLOT-O *"
520 PRINT "**********************"
530 RETURN
800 X=35
810 GOTO 460
900 X=36
910 GOTO 460
1000 X=37
1010 GOTO 460
1100 X=38
1110 GOTO 460
```

Sample Run

```
**********************
* SUPER T.I. SLOT-O *
**********************


***** ***** ***** *****
* $ * * # * * $ * * # *
***** ***** ***** *****


TO PULL THE LEVER,
PRESS ENTER
```

```
**********************
* SUPER T.I. SLOT-O *
**********************

***** ***** ***** *****
* # * * $ * * % * * & *
***** ***** ***** *****

TO PULL THE LEVER,
PRESS ENTER
```

Making Things Move

Moving Illusion

Beware! This constantly-moving image may drive you batty.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 CALL SCREEN(9)
40 FOR L=1 TO 24
50 PRINT TAB(L);CHR$(128)
60 NEXT L
70 GOTO 40
```

Super Moving Illusion

If you liked Tip Number 84 above, you'll love this one! Here the background color changes as well as the color of the lines. Very striking!

Program Listing

```
10 RANDOMIZE
20 CALL CLEAR
30 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
40 SC=INT(16*RND)
50 IF SC<3 THEN 40
60 CALL SCREEN(SC)
70 CL=INT(17*RND)
80 IF CL<3 THEN 70
90 CALL COLOR(13,CL,SC)
100 FOR L=1 TO 24
110 PRINT TAB(L);CHR$(128)
120 NEXT L
130 GOTO 40
```

Circling Dot

More round graphics on your rectangular picture tube.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 CALL COLOR(1,9,6)
40 FOR N=1 TO 12
50 R=12-(10*SIN(N/6*3.14))
60 C=16-(10*COS(N/6*3.14))
70 CALL HCHAR(R,C,128)
80 FOR T=1 TO 75
90 NEXT T
100 CALL CLEAR
110 NEXT N
120 GOTO 40
```

Dot Thrower

Program Listing

```
10 CALL CLEAR
20 D$="0000"
25 RANDOMIZE
30 CALL CHAR(128,D$)
40 CALL COLOR(13,4,2)
45 PRINT "DOT THROWER"
50 PRINT "**********"
55 PRINT
60 PRINT "TO THROW DOTS"
65 PRINT "ACROSS THE SCREEN"
70 PRINT "PRESS THE SPACE BAR"
75 PRINT
80 PRINT "PRESS ONCE FOR ONE DOT"
85 PRINT "OR HOLD DOWN FOR MANY DOTS"
90 N=N+1
95 GOSUB 200
100 CALL VCHAR(P,Q,128)
110 GOTO 90
200 CALL KEY(0,Z,X)
210 IF X=0 THEN 200
220 IF Z=32 THEN 300
230 GOTO 200
300 P=INT(23*RND)+1
310 Q=INT(31*RND)+1
320 IF N>1 THEN 340
330 CALL CLEAR
340 RETURN
```

Multicolor Dot Thrower

Program Listing

```
10 CALL CLEAR
20 D$="0000"
30 RANDOMIZE
40 CALL CHAR(128,D$)
45 PRINT "DOT THROWER"
50 PRINT "***********"
55 PRINT
60 PRINT "TO THROW COLORFUL DOTS"
65 PRINT "ACROSS THE SCREEN"
70 PRINT "PRESS THE SPACE BAR"
75 PRINT
80 PRINT "PRESS ONCE FOR ONE DOT"
85 PRINT "OR HOLD DOWN FOR MANY DOTS"
90 N=N+1
95 GOSUB 200
100 CALL COLOR(13,4,C)
110 CALL VCHAR(P,Q,128)
120 GOTO 90
200 CALL KEY(0,Z,X)
210 IF X=0 THEN 200
220 IF Z=32 THEN 300
230 GOTO 200
300 P=INT(23*RND)+1
310 Q=INT(31*RND)+1
320 C=INT(16*RND)+1
330 IF C<2 THEN 320
340 IF C=4 THEN 320
350 IF N>1 THEN 370
360 CALL CLEAR
370 RETURN
```

Square-Hole Thrower

Program Listing

```
10 CALL CLEAR
20 D$="FFFF81818181FFFF"
30 RANDOMIZE
40 CALL CHAR(128,D$)
45 PRINT "DOT THROWER"
50 PRINT "**********"
55 PRINT
60 PRINT "TO THROW COLORFUL DOTS"
65 PRINT "ACROSS THE SCREEN"
70 PRINT "PRESS THE SPACE BAR"
75 PRINT
80 PRINT "PRESS ONCE FOR ONE DOT"
85 PRINT "OR HOLD DOWN FOR MANY DOTS"
90 N=N+1
95 GOSUB 200
100 CALL COLOR(13,C,4)
110 CALL VCHAR(P,Q,128)
120 GOTO 90
200 CALL KEY(0,Z,X)
210 IF X=0 THEN 200
220 IF Z=32 THEN 300
230 GOTO 200
300 P=INT(23*RND)+1
310 Q=INT(31*RND)+1
320 C=INT(16*RND)+1
330 IF C<2 THEN 320
340 IF C=4 THEN 320
350 IF N>1 THEN 370
360 CALL CLEAR
370 RETURN
```

Making Things Move

Movement on the computer display screen is an illusion. As in any television picture, the turning on and turning off of dots in a pattern across a screen can seem to provide motion to an object drawn on the face of the tube.

There are a number of ways to get the look of motion. Let's send a dot across the screen:

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 FOR C=2 TO 32
40 CALL HCHAR(12,C,128)
50 FOR T=1 TO 25
60 NEXT T
70 CALL CLEAR
80 NEXT C
90 FOR C=31 TO 1 STEP -1
100 CALL HCHAR(12,C,128)
110 FOR T=1 TO 25
120 NEXT T
130 CALL CLEAR
140 NEXT C
150 GOTO 30
```

Reversing Delivery Truck

Program Listing

```
10 CALL CLEAR
20 LET A$="00F8E8F8FFFF44"
25 LET B$="0000000000000000"
26 LET C$="001F171FFFFF22"
27 FOR X=1 TO 28
30 CALL CHAR(128,A$)
40 CALL COLOR(13,2,4)
50 CALL VCHAR(12,X,128)
72 CALL CHAR(129,B$)
74 CALL COLOR(13,4,4)
80 CALL VCHAR(12,X,129)
83 NEXT X
90 FOR X=28 TO 1 STEP -1
100 CALL CHAR(128,C$)
110 CALL COLOR(13,2,4)
120 CALL VCHAR(12,X,128)
130 CALL CHAR(129,B$)
140 CALL COLOR(13,4,4)
150 CALL VCHAR(12,X,129)
160 NEXT X
170 GOTO 27
```

Chase The Cat

Program Listing

```
10 CALL CLEAR
20 A$="0000000000808080"
30 B$="0000000000030303"
40 C$="FF81BDBD81FFFFFF"
50 D$=C$
60 E$="1F1F1F1F1F1F1F1F"
70 F$="F8F8F8F8F8F8F8F8"
80 G$="1F1F1F1F1F1FE0E0"
90 H$="F8F8F8F8F8F80707"
100 CALL CHAR(128,A$)
110 CALL CHAR(129,B$)
120 CALL CHAR(130,C$)
130 CALL CHAR(131,D$)
140 CALL CHAR(132,E$)
150 CALL CHAR(133,F$)
160 CALL CHAR(134,G$)
170 CALL CHAR(135,H$)
180 GOSUB 400
190 FOR X=1 TO 27
200 Y=12
210 CALL COLOR(13,2,4)
220 CALL VCHAR(Y,X,128)
230 CALL VCHAR(Y,X+1,129)
240 CALL VCHAR(Y+1,X,130)
250 CALL VCHAR(Y+1,X+1,131)
260 CALL VCHAR(Y+2,X,132)
270 CALL VCHAR(Y+2,X+1,133)
280 CALL VCHAR(Y+3,X,134)
290 CALL VCHAR(Y+3,X+1,135)
295 IF X=27 THEN 310
300 CALL CLEAR
310 NEXT X
320 FOR T=1 TO 200
330 NEXT T
340 GOTO 180
400 PRINT "PRESS THE SPACE BAR"
410 PRINT "TO MAKE THE CAT"
```

```
420 PRINT "RUN ACROSS THE SCREEN"
430 CALL KEY(0,Z,X)
440 IF X=0 THEN 430
450 CALL CLEAR
460 RETURN
```

Colorful Hopping Pussycat

Program Listing

```
10 CALL CLEAR
20 A$="0000000000808080"
30 B$="0000000000030303"
40 C$="FF81BDBD81FFFFFF"
50 D$=C$
60 E$="1F1F1F1F1F1F1F1F"
70 F$="F8F8F8F8F8F8F8F8"
80 G$="1F1F1F1F1F1FE0E0"
90 H$="F8F8F8F8F8F80707"
100 CALL CHAR(128,A$)
110 CALL CHAR(129,B$)
120 CALL CHAR(130,C$)
130 CALL CHAR(131,D$)
140 CALL CHAR(132,E$)
150 CALL CHAR(133,F$)
160 CALL CHAR(134,G$)
170 CALL CHAR(135,H$)
180 C=INT(16*RND)+1
190 X=INT(26*RND)+1
200 Y=INT(21*RND)+1
210 CALL COLOR(13,C,4)
220 CALL VCHAR(Y,X,128)
230 CALL VCHAR(Y,X+1,129)
240 CALL VCHAR(Y+1,X,130)
250 CALL VCHAR(Y+1,X+1,131)
260 CALL VCHAR(Y+2,X,132)
270 CALL VCHAR(Y+2,X+1,133)
280 CALL VCHAR(Y+3,X,134)
290 CALL VCHAR(Y+3,X+1,135)
300 FOR T=1 TO 200
310 NEXT T
320 CALL CLEAR
330 GOTO 180
```

Cannon Target Shoot

Program Listing

```
10 CALL CLEAR
20 D$="FFFFFFFFFFFFFFFF"
25 G$="0000000000000000"
30 CALL CHAR(128,D$)
35 CALL CHAR(129,G$)
40 CALL COLOR(13,5,4)
50 FOR L=1 TO 19
55 PRINT CHR$(32);
60 NEXT L
65 PRINT "TARGET"
70 FOR L=1 TO 9
75 PRINT
80 NEXT L
100 PRINT CHR$(32);CHR$(32);"CANNON"
110 FOR L=1 TO 11
120 PRINT
130 NEXT L
140 FOR Y=9 TO 15
150 CALL VCHAR(Y,2,128)
160 NEXT Y
170 FOR X=2 TO 12
180 CALL VCHAR(9,X,128)
190 NEXT X
200 FOR Y=9 TO 15
210 CALL VCHAR(Y,12,128)
220 NEXT Y
230 FOR X=12 TO 2 STEP -1
240 CALL VCHAR(15,X,128)
250 NEXT X
260 FOR X=13 TO 15
270 FOR Y=11 TO 13
280 CALL VCHAR(Y,X,128)
290 NEXT Y
300 NEXT X
310 FOR Y=3 TO 12
320 FOR X=16 TO 32
330 CALL VCHAR(Y,25,128)
```

```
340 CALL VCHAR(12,X,128)
342 IF Y=12 THEN 346
344 GOTO 350
346 IF X=25 THEN 400
350 CALL VCHAR(Y,25,129)
360 CALL VCHAR(12,X,129)
370 NEXT X
380 NEXT Y
390 GOTO 500
400 CALL COLOR(13,11,4)
405 FOR XX=24 TO 26
410 FOR YY=11 TO 13
420 CALL VCHAR(YY,XX,128)
430 CALL VCHAR(YY,XX,129)
440 NEXT YY
450 NEXT XX
460 CALL COLOR(13,5,4)
470 X=32
480 GOTO 350
500 GOTO 310
```

Winking Man

Program Listing

```
10 CALL CLEAR
20 W$="FFFFFFFFFFFFFFFF"
30 B$="0000000000000000"
40 CALL CHAR(128,W$)
50 CALL CHAR(129,B$)
60 CALL COLOR(13,2,4)
70 N=1
100 FOR X=2 TO 31
110 Y=1
120 CALL VCHAR(Y,X,128)
140 NEXT X
200 FOR Y=1 TO 24
210 X=31
220 CALL VCHAR(Y,X,128)
230 NEXT Y
300 FOR X=31 TO 2 STEP -1
310 Y=24
320 CALL VCHAR(Y,X,128)
330 NEXT X
400 FOR Y=24 TO 1 STEP -1
410 X=2
420 CALL VCHAR(Y,X,128)
430 NEXT Y
500 FOR X=4 TO 10
510 FOR Y=3 TO 9
520 CALL VCHAR(Y,X,128)
530 NEXT Y
540 NEXT X
600 FOR X=22 TO 29
610 FOR Y=3 TO 9
620 CALL VCHAR(Y,X,128)
630 NEXT Y
640 NEXT X
700 FOR Y=11 TO 13
710 FOR X=14 TO 17
720 CALL VCHAR(Y,X,128)
730 NEXT X
```

```
740 NEXT Y
800 FOR X=5 TO 28
810 FOR Y=19 TO 20
820 CALL VCHAR(Y,X,128)
830 NEXT Y
840 NEXT X
850 CALL VCHAR(18,4,128)
860 CALL VCHAR(18,29,128)
900 IF N=1 THEN 1100
1000 FOR Y=3 TO 9
1010 FOR X=22 TO 29
1020 CALL VCHAR(Y,X,128)
1030 NEXT X
1040 NEXT Y
1050 N=1
1060 FOR T=1 TO 150
1070 NEXT T
1080 GOTO 900
1100 FOR Y=9 TO 3 STEP -1
1110 FOR X=29 TO 22 STEP -1
1120 CALL VCHAR(Y,X,129)
1130 NEXT X
1140 NEXT Y
1150 N=0
1160 GOTO 1060
```

Borders, Boxes & Billboards

Box The Screen

Here's how to draw a box around the graphics display-screen area on your TV monitor. Lines 30 and 50 draw the vertical sides of the box while lines 20 and 40 draw the horizontal bottom and top. Line 60 is a freeze-frame loop to hold the picture so you can see it.

Program Listing

```
10 CALL CLEAR
20 CALL HCHAR(1,1,64,32)
30 CALL VCHAR(1,32,64,24)
40 CALL HCHAR(24,1,64,32)
50 CALL VCHAR(1,1,64,24)
60 GOTO 60
```

Screen Border

Program Listing

```
10 CALL CLEAR
20 D$="FFFF81818181FFFF"
40 CALL CHAR(128,D$)
50 CALL COLOR(13,7,16)
100 FOR X=2 TO 31
110 Y=1
120 CALL VCHAR(Y,X,128)
130 NEXT X
200 FOR Y=1 TO 24
210 X=31
220 CALL VCHAR(Y,X,128)
230 NEXT Y
300 FOR X=31 TO 2 STEP -1
310 Y=24
320 CALL VCHAR(Y,X,128)
330 NEXT X
400 FOR Y=24 TO 1 STEP -1
410 X=2
420 CALL VCHAR(Y,X,128)
430 NEXT Y
500 GOTO 500
```

Flashing Screen Border

Program Listing

```
10 CALL CLEAR
20 D$="FFFF81818181FFFF"
40 CALL CHAR(128,D$)
100 FOR X=2 TO 31
110 Y=1
115 GOSUB 600
120 CALL VCHAR(Y,X,128)
130 NEXT X
200 FOR Y=1 TO 24
210 X=31
215 GOSUB 800
220 CALL VCHAR(Y,X,128)
230 NEXT Y
300 FOR X=31 TO 2 STEP -1
310 Y=24
315 GOSUB 600
320 CALL VCHAR(Y,X,128)
330 NEXT X
400 FOR Y=24 TO 1 STEP -1
410 X=2
415 GOSUB 800
420 CALL VCHAR(Y,X,128)
430 NEXT Y
500 GOTO 100
600 IF INT(X/2)=(X/2)THEN 700
610 CALL COLOR(13,7,16)
620 RETURN
700 CALL COLOR(13,16,7)
710 RETURN
800 IF INT(Y/2)=(Y/2)THEN 900
810 CALL COLOR(13,7,16)
820 RETURN
900 CALL COLOR(13,16,7)
910 RETURN
```

Box Score

To dress up scores during and at the end of a game program, use this method of putting those scores in a box. The box around the score will highlight it and jazz up your video display.

Program Listing

```
10 CALL CLEAR
20 INPUT "PLAYER'S NAME:   ":N$
30 INPUT "PLAYER'S SCORE: ":S
40 PRINT
50 S$=STR$(S)
60 LN=LEN(N$)+LEN(S$)
70 LT=LN+14
80 FOR Z=1 TO LT
90 PRINT "*";
100 NEXT Z
110 PRINT
120 PRINT "* ";N$;"'S SCORE: ";S$;" *"
130 FOR Z=1 TO LT
140 PRINT "*";
150 NEXT Z
160 FOR Z=1 TO 10
170 PRINT
180 NEXT Z
190 GOTO 20
```

Sample Run

```
PLAYER'S NAME:   HELEN
PLAYER'S SCORE:  88

**********************
* HELEN'S SCORE: 88 *
**********************
```

```
PLAYER'S NAME:   SAM
PLAYER'S SCORE:   98765

**********************
* SAM'S SCORE: 98765 *
**********************
```

Blackboard

This program appears to draw a blackboard on the screen. You can write messages on it, draw football plays, create art or words etc. on this electronic chalkboard.

Program Listing

```
10 CALL CLEAR
20 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
30 CALL SCREEN(9)
40 FOR C=10 TO 22
50 FOR R=5 TO 19
60 CALL HCHAR(R,C,128)
70 NEXT R
80 NEXT C
90 GOTO 90
```

Centered Boxed Titles

Here's how to dress up your program titles.

After you type in and RUN the program, press any key on your keyboard to do more.

Program Listing

```
10 CALL CLEAR
20 INPUT "WHAT IS THE TITLE?  ":T$
30 LT=LEN(T$)
40 IF LT>20 THEN 60
50 GOTO 80
60 PRINT "OOPS, TOO LONG, TRY AGAIN"
70 GOTO 20
80 LB=LT+6
90 SP=((32-LB)/2)-1
100 AS$="*"
110 CALL CLEAR
120 PRINT TAB(SP-1);" ";
130 FOR L=1 TO LB
140 PRINT AS$;
150 NEXT L
160 PRINT TAB(SP);"** ";T$;" **"
170 PRINT TAB(SP-1);" ";
180 FOR L=1 TO LB
190 PRINT AS$;
200 NEXT L
210 FOR L=1 TO 15
220 PRINT
230 NEXT L
240 CALL KEY(0,Z,X)
250 IF X=0 THEN 240
260 GOTO 10
```

Flashing Message Border

Program Listing

```
10 CALL CLEAR
20 B$="FFFF81818181FFFF"
30 CALL CHAR(128,B$)
35 REM *****MESSAGE STORED IN M$ AT
   LINE 50
40 PRINT "TYPE A MESSAGE OF UP"
45 PRINT "TO 20 CHARACTERS IN LENGTH"
50 INPUT M$
55 LM=LEN(M$)
60 IF LM>20 THEN 50
65 CALL CLEAR
70 TS=INT((32-LM)/2)-1
80 FOR L=1 TO TS
82 PRINT CHR$(32);
84 NEXT L
86 PRINT M$
88 FOR L=1 TO 11
90 PRINT
92 NEXT L
100 FOR X=TS TO TS+LM+5
110 Y=9
115 GOSUB 600
120 CALL VCHAR(Y,X,128)
130 NEXT X
200 FOR Y=9 TO 15
210 X=TS+LM+5
215 GOSUB 800
220 CALL VCHAR(Y,X,128)
230 NEXT Y
300 FOR X=(TS+LM+5)TO TS STEP -1
310 Y=15
315 GOSUB 600
320 CALL VCHAR(Y,X,128)
330 NEXT X
400 FOR Y=15 TO 9 STEP -1
410 X=TS
415 GOSUB 800
```

```
420 CALL VCHAR(Y,X,128)
430 NEXT Y
500 GOTO 100
600 IF INT(X/2)=(X/2)THEN 700
610 CALL COLOR(13,7,16)
620 RETURN
700 CALL COLOR(13,16,7)
710 RETURN
800 IF INT(Y/2)=(Y/2)THEN 900
810 CALL COLOR(13,7,16)
820 RETURN
900 CALL COLOR(13,16,7)
910 RETURN
```

Flashing Program Title

Program Listing

```
10 CALL CLEAR
20 B$="FFFF81818181FFFF"
30 CALL CHAR(128,B$)
40 REM *****PROGRAM TITLE OF UP TO 22
   CHARACTERS STORED IN PT$ IN LINE 50
50 PT$="PROGRAM TITLE"
55 LM=LEN(PT$)
60 IF LM>22 THEN 50
65 CALL CLEAR
70 TS=INT((32-LM)/2)-1
80 FOR L=1 TO TS
82 PRINT CHR$(32);
84 NEXT L
86 PRINT PT$
88 FOR L=1 TO 11
90 PRINT
92 NEXT L
100 FOR X=TS TO TS+LM+5
110 Y=9
115 GOSUB 600
120 CALL VCHAR(Y,X,128)
130 NEXT X
200 FOR Y=9 TO 15
210 X=TS+LM+5
215 GOSUB 800
220 CALL VCHAR(Y,X,128)
230 NEXT Y
300 FOR X=(TS+LM+5)TO TS STEP -1
310 Y=15
315 GOSUB 600
320 CALL VCHAR(Y,X,128)
330 NEXT X
400 FOR Y=15 TO 9 STEP -1
410 X=TS
415 GOSUB 800
420 CALL VCHAR(Y,X,128)
430 NEXT Y
```

```
500 GOTO 100
600 IF INT(X/2)=(X/2)THEN 700
610 CALL COLOR(13,5,11)
620 RETURN
700 CALL COLOR(13,11,5)
710 RETURN
800 IF INT(Y/2)=(Y/2)THEN 900
810 CALL COLOR(13,5,11)
820 RETURN
900 CALL COLOR(13,11,5)
910 RETURN
```

Appendix

Appendix A: BASIC Words

ABS	absolute value
ASC	ASCII number of string's first character
ATN	trig arctangent
BREAK	stops program run
BYE	leaves BASIC
CALL CHAR	redefines ASCII characters
CALL CLEAR	erases video display
CALL COLOR	sets video colors
CALL GCHAR	finds video location contents
CALL HCHAR	places character on video screen
CALL JOYST	joystick input
CALL KEY	find keypress; like INKEY$
CALL SCREEN	change video screen color
CALL SOUND	causes tones, noise
CALL VCHAR	places character on video screen
CHR$	changes number to character
CLOSE	shuts a file
CON	same as CONTINUE
CONTINUE	resume run after BREAK
COS	trig cosine
DATA	stores numbers, letters in program
DEF	user-defined function
DELETE	removes program or file
DIM	dimensions an array
DISPLAY	PRINT
EDIT	displays line for changing
END	concludes program execution
EOF	end of file
EXP	exponential value e^x
FOR	FOR/NEXT loop
GOSUB	move to subroutine
GOTO	move to line number
IF	IF/THEN decision maker
ELSE	IF/THEN/ELSE decision maker
INPUT	takes in info
INPUT#	takes in info from external file
INT	finds integer of number
LEN	changes character to number

LET	optional; assign value to variable
LIST	display program lines
LOG	natural logarithm
NEW	empty the program memory
NEXT	FOR/NEXT loop
NUM	same as NUMBER
NUMBER	automatic line numbering
OLD	load from mass storage device to memory
ON	ON/GOSUB or ON/GOTO
OPEN	establish a file
OPTION BASE	lowest array subscript
POS	position of a substring
PRINT	output to display or external device
RANDOMIZE	shuffle random number generator
READ	finds DATA lines and moves info
REC	record number; with PRINT
REM	remarks
RES	same as RESEQUENCE
RESEQUENCE	renumber program lines
RESTORE	resets DATA/READ
RETURN	go back to main program after GOSUB
RND	random number generator
RUN	start program execution
SAVE	copy program to external file
SEG$	finds substring
SGN	find whether number is positive or negative
SIN	trig sine
SQR	square root
STEP	FOR/NEXT loop increment control
STOP	ends program run
STR$	changes number to string
TAB	control PRINT or DISPLAY location
TAN	trig tangent
THEN	IF/THEN decision maker
TO	FOR/NEXT loop
TRACE	debug program lines
UNBREAK	remove breakpoints
UNTRACE	cancels TRACE
VAL	changes string to number

Appendix B: Character Sets

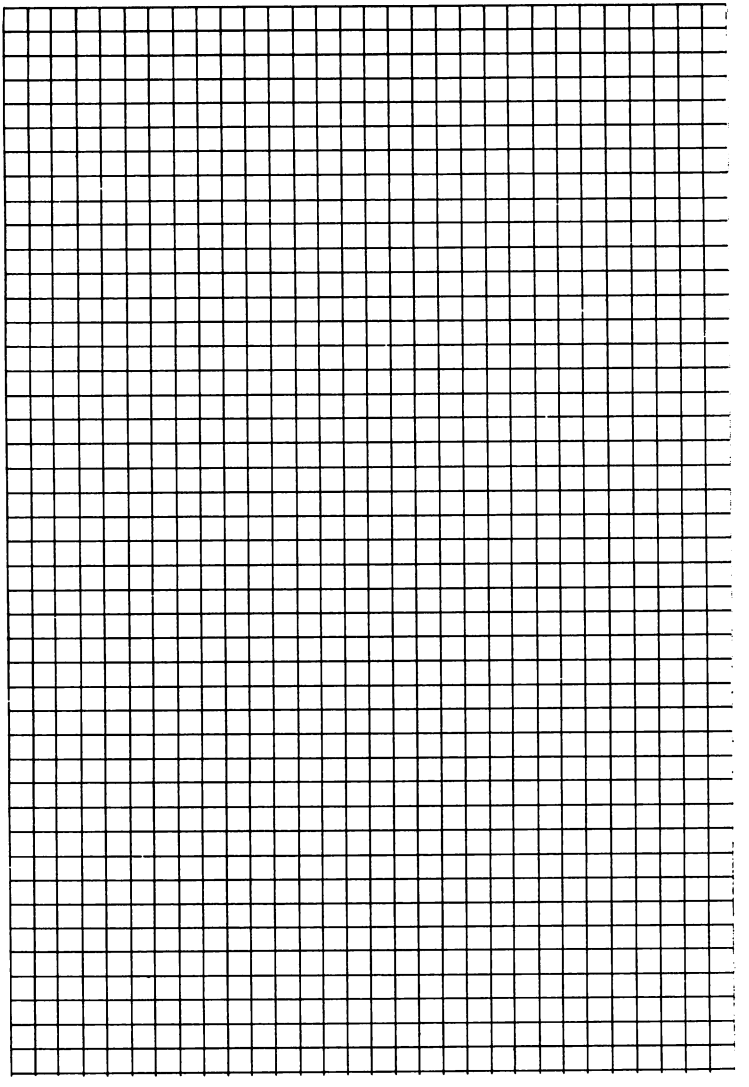
Set	ASCII Code
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159

Appendix C: Color Codes

Number	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

Appendix D: Graphics Grid

Use the handy grid form to layout your on-screen designs. Imagine the vertical direction as the Y axis and the horizontal direction as the X axis. You may substitute larger graph paper for this function. Use 10-per-inch division paper.



Computer books from ARCsoft Publishers At Your Bookstore

For the Texas Instruments TI-99/4A Home Computer:

Texas Instruments Home Computer Games Programs
Len Turner $8.95 ISBN 0-86668-032-2

Texas Instruments Home Computer Graphics Programs
Len Turner $9.95 ISBN 0-86668-031-4

101 Programming Tips & Tricks for the Texas Instruments TI-99/4A
Len Turner $8.95 ISBN 0-86668-025-X

36 Texas Instruments TI-99/4A Programs for Home, School & Office
Len Turner $8.95 ISBN 0-86668-024-1

Texas Instruments Computer Program Writing Workbook
Len Turner $4.95 ISBN 0-86668-812-9

For the Commodore computers:

Commodore 64 Computer Programs for Beginners
Howard Adler $8.95 ISBN 0-86668-033-0

101 Programming Tips & Tricks for the VIC-20 and Commodore 64
Howard Adler $8.95 ISBN 0-86668-030-6

34 VIC-20 Computer Programs for Home, School & Office
Howard Adler $8.95 ISBN 0-86668-029-2

VIC-20 Computer Program Writing Workbook
Howard Adler $4.95 ISBN 0-86668-811-0

For the TRS-80 Model 100 and other portable computers

44 Programs for the TRS-80 Model 100 Portable Computer
Jim Cole $8.95 ISBN 0-86668-034-9

For the TRS-80 Color Computer and TDP-100 computers:

Color Computer Graphics
Ron Clark $9.95 ISBN 0-86668-012-8

101 Color Computer Programming Tips & Tricks
Ron Clark $7.95 ISBN 0-86668-007-1

55 Color Computer Programs for Home, School & Office
Ron Clark $9.95 ISBN 0-86668-005-5

55 MORE Color Computer Programs for Home, School & Office
Ron Clark $9.95 ISBN 0-86668-008-X

The Color Computer Songbook
Ron Clark $7.95 ISBN 0-86668-011-X

TRS-80 Color Computer Program Writing Workbook
Ron Clark $4.95 ISBN 0-86668-816-1

My Buttons Are Blue and Other Love Poems
Edited by Ron Clark $4.95 ISBN 0-86668-013-6

For the ATARI computers:

101 ATARI Computer Programming Tips & Tricks
Alan North $8.95 ISBN 0-86668-022-5

31 New ATARI Computer Programs for Home, School & Office
Alan North $8.95 ISBN 0-86668-018-7

ATARI Computer Program Writing Workbook
Alan North $4.95 ISBN 0-86668-814-5

For the TIMEX/Sinclair 1000, Sinclair ZX-81 and MicroAce:

Practical TIMEX/Sinclair Computer Programs for Beginners
Edward Page $7.95 ISBN 0-86668-027-6

101 TIMEX 1000/Sinclair ZX-81 Programming Tips & Tricks
Edward Page $7.95 ISBN 0-86668-020-9

TIMEX/Sinclair Computer Games Programs
Edward Page $7.95 ISBN 0-86668-026-8

37 TIMEX 1000/Sinclair ZX-81 Programs for Home, School & Office
Edward Page $8.95 ISBN 0-86668-021-7

TIMEX/Sinclair Computer Program Writing Workbook
Edward Page $4.95 ISBN 0-86668-810-2

For the Apple and Franklin ACE computers:

101 APPLE Computer Programming Tips & Tricks
Fred White $8.95 ISBN 0-86668-015-2

33 New APPLE Computer Programs for Home, School & Office
Fred White $8.95 ISBN 0-86668-016-0

APPLE Computer Program Writing Workbook
Fred White $4.95 ISBN 0-86668-813-7

For the TRS-80, Sharp and Casio Pocket Computers:

Practical PC-2/PC-1500 Pocket Computer Programs
Jim Cole $7.95 ISBN 0-86668-028-4

Pocket Computer Programming Made Easy
Jim Cole $8.95 ISBN 0-86668-009-8

99 Tips & Tricks for the New Pocket Computers
Jim Cole $7.95 ISBN 0-86668-019-5

101 Pocket Computer Programming Tips & Tricks
Jim Cole $7.95 ISBN 0-86668-004-7

Murder In The Mansion and Other Computer Adventures — 2nd Edition
Jim Cole $6.95 ISBN 0-86668-501-4

50 Programs in BASIC for Home, School & Office — 2nd Edition
Jim Cole $9.95 ISBN 0-86668-502-2

50 MORE Programs in BASIC for Home, School & Office
Jim Cole $9.95 ISBN 0-86668-003-9

Pocket Computer Program Writing Workbook
Jim Cole $4.95 ISBN 0-86668-817-X

35 Practical Programs for the CASIO Pocket Computer
Jim Cole $8.95 ISBN 0-86668-014-4

Texas Instruments Home Computer Book IV $9.95

Texas Instruments Home Computer Graphics Programs

by Len Turner

If you have a Texas Instruments TI-99/4A Home Computer and want to do video graphics with it, you must have this handy volume of introduction and more than three-dozen complete ready-to-run programs in BASIC.

You will find 38 complete programs, ready to be typed in and run immediately. The programs run exactly as you find them in this book. They require no modification, no rewriting, no programming skill. All you need to know is how to turn on your TI-99/4A Home Computer and type these programs into it. The computer does all the rest!

You also will find, in this exciting new guide, an introduction to computer video graphics, how it works, how to use it. The simple down-to-earth language used in this introduction will make video graphics clear to you. The learn-by-doing programs will let you actually get hands-on experience running video graphics programs on your own home computer.

Sections in this book include *Sketches, Graphs & Stuff, Making Things Move,* and *Borders, Boxes & Billboards.* Among the 38 instant-use programs in this book are on-the-tube sketchers, bar graph generators, checkerboards, walls, fences, delivery trucks, cats, cannons, winking men, flashing borders, fancy screens, colorful message boards, window twinklers, flashing cursors, dot throwers, boxed titles and even a low-res slot machine!

An outstanding book of graphics programs for your home computer, you will be pleased to find the easy-to-understand explanation of graphics an added bonus in this guidebook. It covers all the exciting elements you need to know to get started writing your own graphics programs for your home computer. The elementary explanation of graphics as presented here is applicable to most home computers so, even if you don't own a TI-99/4A, you will be able to understand how graphics work. Read these programs and learn how to create your own programs. Weave these programs into larger sets of instructions you are writing for your home computer. Use them to put fun into your creations.

ARCsoft Publishers
Woodsboro, Maryland

ISBN 0-86668-031-4