CAT. NUMBER
26-2101

USER'S MANUAL FOR LEVEL 1

Radio Shack

TRS-80

MICRO COMPUTER SYSTEM

CUSTOM MANUFACTURED IN U.S.A. BY RADIO SHACK A DIVISION OF TANDY CORPORATION

LIMITED WARRANTY

Radio Shack warrants for a period of 90 days from the date of delivery to customer that the computer hardware described herein shall be free from defects in material and workmanship under normal use and service. This warranty shall be void if the computer case or cabinet is opened or if the unit is altered or modified. During this period, if a defect should occur, the product must be returned to a Radio Shack store or dealer for repair. Customer's sole and exclusive remedy in the event of defect is expressly limited to the correction of the defect by adjustment, repair or replacement at Radio Shack's election and sole expense, except there shall be no obligation to replace or repair items which by their nature are expendable. No representation or other affirmation of fact, including but not limited to statements regarding capacity, suitability for use, or performance of the equipment, shall be or be deemed to be a warranty or representation by Radio Shack, for any purpose, nor give rise to any liability or obligation of Radio Shack whatsoever.

EXCEPT AS SPECIFICALLY PROVIDED IN THIS AGREEMENT, THERE ARE NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE AND IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS OR BENEFITS, INDIRECT, SPECIAL, CONSEQUENTIAL OR OTHER SIMILAR DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR OTHERWISE.

IMPORTANT NOTICE

ALL RADIO SHACK COMPUTER PROGRAMS ARE DISTRIBUTED ON AN "AS IS" BASIS WITHOUT WARRANTY

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

NOTE: Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.

A Personal Note from the Author

This is not a conventional book. There are plenty of good conventional books, and plenty that are not so good.

This book is written specifically for people who don't know anything about computers, and who don't want to be dazzled by fancy footwork from someone who does. It is written to teach you how to use your Radio Shack TRS-80 computer and start you on a fast track to becoming a competent programmer. To that end, every fair and unfair, conventional and unconventional, flamboyant and ridiculous technique I could think of was used. I want you to have fun with your computer! I don't want you to be afraid of it, because there is nothing to fear.

The only restraints put on this book were good taste and a genuine attempt not to insult your intelligence. Beyond that, it contains no "snow jobs", no efforts to impress or intimidate you, and no attempt to sell you anything except the idea that computers are just not all that hard to learn to use.

Sit back, relax, read slowly as though savoring a good novel, and above all, let your imagination wander. I'll supply you with all the routine facts and techniques you need. The real enjoyment begins when your imagination starts the creative juices flowing and the computer becomes a tool in your own hands. You become its master - not the other way around. At that point it evolves from just a box of parts into an extension of your personality.

Enjoy your new computer!

Dr. David A. Lien
San Diego - 1977

FIRST EDITION
SECOND PRINTING — 1978

Table of Contents

This User's Manual and You

This Manual has been written for the average person who has no experience with a Computer. We've deliberately kept our style light and humorous (some may even say it's corny!) . . . we think this will make your learning experience fun.

(And why **shouldn't** learning be fun . . . ?)

The Manual is organized in three basic sections:

A. 26 Chapters which introduce you to various capabilities of the Computer; in small enough bites so you won't choke. These Chapters include numerous little check points and examples (as we get deeper into the book the examples get deeper).
At the end of the Chapters we've given some Exercises – to give you a chance to try out your knowledge ON YOUR OWN.

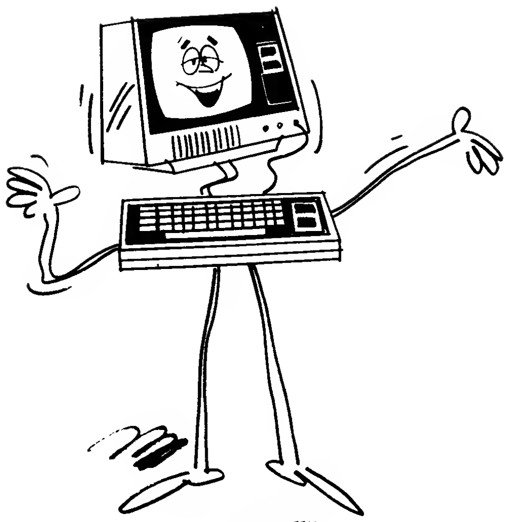
B. A section with sample answers to the Exercises in each Chapter. You can see how you make out with your attempts at programming.

C. A section with some **User's Programs** – some good examples of interesting and practical programs (some for fun, some for business, some for education, etc.).

We've also included some helpful information in an APPENDIX.

The Manual is written in a style where the Computer assists you in learning (educators might like to call it "Computer Assisted Instruction" . . . we'll try to avoid trying to impress you with that type of fancy wording).

So, on you go – and we hope you have as much fun with this book as we did preparing it (we had some headaches too . . . hope you don't have any of those).

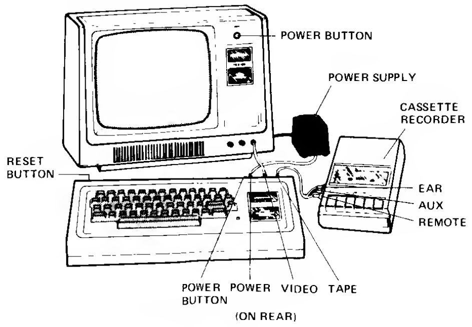


SETTING UP THE SYSTEM

Carefully unpack the system. Remove all packing material. Be sure you locate all cables, papers, tapes, etc. Save the packing material in case you need to transport the system.

Connecting the Video Display and Keyboard:

1. Connect the power cord from the Video Display to a source of 120 volts, 60 Hz AC power. Note that one prong of the AC plug is wider than the other — the wide prong should go into the widest slot of the AC socket.
   NOTE: If you use an AC extension cord, you may not be able to plug the Display's power cord in. Do not attempt to force **this wide prong into the extension cord**; use a wall outlet if at all possible.
2. Connect the power cord of the Power Supply to a source of 120 volts, 60 Hz AC power.
3. Connect the gray cable from the front of the Video Monitor to the **VIDEO** jack on the back of the Keyboard Assembly. **Take care to line up the pins correctly** (the plug fits only one way).
   NOTE: Before the next step, be sure the **POWER** switch on the back of the Keyboard is off (button out).
4. Connect the gray cable from the Power Supply to the **POWER** jack on the back of the Keyboard Assembly. **Again, take care to mate the connection correctly**.



Connecting the Cassette Recorder:

NOTE: You do not need to connect the Cassette Recorder unless you plan to record programs or to load taped programs into the TRS-80.

1. Load batteries into the CTR-41 as described in the Manual. Or make connections for 120 volt AC power.
2. Connect the short cable (DIN plug on one end and 3 plugs on the other) to the **TAPE** jack on the back of the Keyboard Assembly. **Be sure you get the plug to mate correctly**.
3. The 3 plugs on the other end of this cable are for connecting to the CTR-41.
   A. Connect the **black** plug into the EAR jack on the side of the CTR-41. This connection provides the output signal from the CTR-41 to the TRS-80 (for loading Tape programs into the TRS-80).
   B. Connect the larger **gray plug** into the AUX jack on the CTR-41. This connection provides the recording signal to record programs from the TRS-80 onto the CTR-41's tape.
   Also, plug the Dummy Plug (provided with the CTR-41) into the MIC jack (this disconnects the built-in Mic so it won't pick-up sounds while you are loading tapes).
   NOTE: Be sure you always use the Dummy Plug when loading programs onto tape (Recording).

Dummy Plug

   C. Connect the smaller gray plug into the REM jack on the CTR-41. This allows the TRS-80 to automatically control the CTR-41's motor (turn tape motion on and off for recording and playing tapes).

Notes On Using The Recorder

There are a number of things you should be aware of as you use the Cassette Tape System: (Some of this will be covered in greater detail in Chapter 9 . . . but some of you can't wait till then . . . *can you!)*

1. To Play a tape (load a taped program into the TRS-80), **you must have the CTR-41's Volume control set to 7 to 8**. Then press the CTR-41's PLAY key and then type CLOAD on the TRS-80 and **ENTER** this command. This will start the tape motion. An * will appear on the top line of the Monitor; a second * will blink, indicating the program is loading. When loading is done, the TRS-80 will automatically turn the CTR-41 off and flash READY on the screen. You are then ready to RUN the program (type in and hit **ENTER**).

2. To Record a program from the TRS-80, press the CTR-41's RECORD and PLAY keys simultaneously. Then type CSAVE on the TRS-80 and **ENTER** this command. When the program has been recorded, the TRS-80 will automatically turn the CTR-41 off and flash READY on the screen. Now you have your program on tape (it still is in the TRS-80 also). Many computer users make a second or even a third recording of the tape, just to be sure they have a good recording.
   NOTE: To load the full 4K of RAM in the TRS-80 takes less than 3 minutes of tape. Short programs will take only a few seconds of tape.
3. Use the CTR-41's Tape Counter to aid you in locating programs on tapes.
4. For best results, use Radio Shack's special 10 minute Computer Tape Cassettes (especially designed for recording computer programs). If you use standard audio tape cassettes, be sure to use top quality, such as Realistic SUPERTAPE. Keep in mind that audio cassettes have lead-ins on both ends (blue non-magnetic mylar material) — **you can not record on the leader portion of the tape.** Advance the tape past the leader before recording a program.
5. When you are not going to use a CTR-41 for loading or recording programs, do not leave RECORD or PLAY keys down (press STOP).
6. To REWIND or FAST-Forward a cassette, you must disconnect the plug from the REM jack (with REM jack connected, the TRS-80 controls tape motion).
7. If you want to save a taped program permanently, break off the erase protect tab on the cassette (see CTR-41 Manual).
8. Do not expose recorded tapes to magnetic fields. Avoid placing your tapes near the Power Supply.
9. To check if a tape has a program recorded on it, you can disconnect the plug from the EAR jack (also disconnect the REM plug so you can control the CTR-41 with the keys) and Play the tape; you'll hear the program material from the speaker.

TURNING THE SYSTEM ON

Turn on the Video Display by pressing the **POWER** button. Turn on the TRS-80 Keyboard by pressing the **POWER** button on the back (next to the POWER jack); the red LED just to the right of the Keyboard should light up and the screen should show READY . Adjust C (contrast) and B (brightness) controls on the front of the Display for the sharpest display. Set Brightness so the background is gray and the words are white. Do not set Brightness too high.

If Display does not show READY , press the Keyboard's **POWER** switch off and on again.

NOTE: There is a Reset button inside a door at the left rear of the Keyboard assembly. This Reset button can be used to unlock a looping program or if the TRS-80 does not turn off a cassette or in other such abnormal program situations.

One More Thought –

You're all ready now, right? Well, maybe. But let's just prepare you for the TRS-80 and Manual with one more thought . . .

How do you "talk" to a Computer? In Binary Numbers? In Electronics (is there such a language . . .)? In English . . .?

Well, we use a simplified form of English — it's called the BASIC Language (Beginners All-purpose Symbolic Instruction Code). (There are lots of other "computer languages", but this is the easiest.) This Manual covers Radio Shack's LEVEL I BASIC.

As you go through this Manual you'll learn the different words of this simple computer language — and how to punctuate *(VERY IMPORTANT)* — and how to apply all of it for fun and practical benefit. It's an easy language to learn — but remember, **you've got to use the language that the TRS-80 understands** (we'll be giving you some examples of wrong language use and you'll see what happens).

Chapter 1

Computer Etiquette

From the moment you turn it on, the TRS-80 follows a well-defined set of rules for coping with you, the "master." This makes it an especially easy computer to use. To a large extent, all you have to do is say the right thing (via the keyboard) at the right time. Of course, there are lots of "right things" to say; putting them together for a purpose is called **programming**.

This set of rules is permanently stored in the Computer in two programs, called the monitor and the interpreter.

In this chapter we're going to start a conversation with the TRS-80 by teaching it a few simple social graces. At the same time, you'll be learning the fundamentals of computer etiquette. You'll even write, wonder of wonders, your first TRS-80 computer program!

Getting READY

1. Connect the keyboard-computer, Video Display and Power Supply as explained in the previous section. Plug Video Display and Power Supply into 12Ø-volt AC outlets.
2. Press POWER button on Video Display and the back of the Keyboard. Give the video tube a few seconds to warm up.
3. READY
   >— should appear in the upper left corner of the screen. Press the **ENTER** key several times to produce a column of READY messages. The Computer is trying to tell you something: "I'm ready — it's your turn to do something!"

ENTER is a monitor command. It tells the Computer to take a look at whatever you've typed on the screen. In step 3, you didn't type anything, so the Computer just comes back with another READY.

To make sure you start off with a clean slate — erasing all traces of prior programs or tests — type NEW and press **ENTER**. The Computer will respond by erasing the screen and printing

```
READY
>—
```

at the top of the screen.

Now type in P.M. and **ENTER**. This is a test to see that the Computer powered up properly. The display should read:

```
P.M.
3583
```

Hit [P] key, [.] key, [M] key and [.] key. Don't use shift key — letters are always capital for TRS-80.

If you have 8K of memory, the number should be 7679. With 16K of memory, it should be 15871.

If the number is not 3583, turn the Computer off, using the pushbutton on the right rear corner of the keyboard. Wait about 1Ø seconds and turn it on again. Repeat the test and verify that the number is 3583.

It's good practice to perform this simple test whenever you turn on the TRS-80. Always type NEW and **ENTER** before performing the test. As for what the test tests — we'll wait a few chapters for that!

"EXTRA CAREFUL"

Just What Is a Computer Program?

A program is a sequence of instructions that the Computer stores until we command it to follow (or "execute") those instructions. Programs for the TRS-80 are written in a language called BASIC — and that should give you an idea of how easy it is to learn!

Let's write a simple one-line program to let the TRS-80 introduce itself. First be sure the last line on the screen shows a >, which we call the "prompt". This is the Computer's way of saying, "Go ahead — do something!" Now type the following line, **exactly** as shown:

```
1Ø PRINT "HELLO THERE. I AM YOUR NEW TRS-8Ø MICROCOMPUTER!"
```

Do **not** hit **ENTER** key yet!

You don't have to use the SHIFT key to get a capital letter — that's the only kind of letter the TRS-80 uses. However, some of the keys do have two characters printed on them. Use the SHIFT key to get the upper characters — like the " marks and the exclamation point(!).

If you made a mistake, don't worry — it's much easier to correct typing errors on the TRS-80 than it is on a regular typewriter. No rubber erasers or white paint to fuss with! Just use the backspace key ←. Each time you press this key, the rightmost character will be erased. If your error was at the beginning of the line, you'll have to erase your way back to that point and then retype the rest of the line.

See the little "dash" (–) that moves across the screen as you type in a letter? This is the "cursor". It lets you know exactly where the next character you type will be printed on the screen. Pushing the space bar moves the cursor along one space, without printing anything.

Now go back and examine **VERY CAREFULLY** what you have typed:

1. Did you enclose everything after the word PRINT in quotation marks?
2. Are there any extra quotation marks?

If everything's okay, you can press **ENTER**. The > prompt will reappear. The Computer is telling you, "Fine — what's next?"

If you press **ENTER** a second time, the screen will read:

```
READY
```

This is reassuring, but not necessary — as long as the bottom item on the screen is the >prompt, you know it's "your turn."

If It's Too Late

If you find an error after you've typed a line and pressed **ENTER**, you cannot use the ← backspace key to correct it. Instead, retype the **entire** line correctly. As soon as you **ENTER** the line, it will replace the incorrect one. This is because both of them share the same starting number (in this case, 1Ø).

"Allow me to introduce myself."

Now we'll tell the Computer to execute our program. The BASIC command for this is simple: RUN. So type RUN and press **ENTER** . If you made no mistakes, the display will read:

```
HELLO THERE. I AM YOUR NEW TRS-8Ø MICROCOMPUTER!
```

If this isn't what you got, go back and try it again. If RUN still doesn't produce the greeting, there's something wrong in your program. Type NEW to clear it out and type in the one-line program again.

If it did work — let out a yell! "HEY MA, IT WORKS!" This is very important, because now that you have tasted success with a computer, it may be the last you are heard from in some time.

Note that the word PRINT was not displayed, nor were the quotation marks. They are part of the program's instructions and we didn't intend for them to be printed.

Type the word RUN again and hit **ENTER** .

Type RUN to your heart's content, watching the magic machine do as it's told, over and over. When you feel you've really got the hang of all this, get up and stretch, walk around the room, look out the window — the whole act. Because you'll soon get hooked and you won't want to take time for such things later on.

Whether you're typing in a program, or giving direct commands like RUN, you've got to hit **ENTER** to tell the Computer to take a look at what you've typed and act accordingly.

Learned in Chapter 1

Commands	Statements	Miscellaneous
BREAK	PRINT	> prompt
ENTER		— cursor
NEW		← backspace key
RUN		" " quotation marks

We'll put a list like this at the end of each chapter. Use it as a checkpoint to make sure you didn't miss anything.

Maybe you're wondering what's the difference between BASIC commands and BASIC statements. Commands are executed as soon as you type them in and press **ENTER** . Statements are put in to programs and are only executed after you type the command RUN .

Special message for people who can't resist the urge to play around with the computer and skip around in this book. *(There always are a few!)* **It's possible to "lose control" of the Computer, so that it won't give you a READY message when you press ENTER. To regain control, just press BREAK , then ENTER . If that doesn't work, find the Reset button inside the left rear corner of the TRS-80 and push it. There!**

As you exercise your TRS-80, you'll note that with SHIFT you get some symbol/characters that are not used with LEVEL I (Eg. ^ [ ]) although they can be inside a print statement.

Notes:

Chapter 2

How To Expand A Program

You now have a program in the Computer. (If you turned it off between lessons, fire it up again and type in line 1Ø from Chapter 1.) It's only a one-liner, but let's expand it by adding a second line. In BASIC, every line in the program must have a number, and the program is executed in order from the smallest number to the largest. Type:

```
2Ø PRINT "YOU CALLED, MASTER. DO YOU HAVE A COMMAND?"
```

Have you noticed that we use Ø for the number zero — so you can distinguish between the letter and number. The Video Display does it this way — so we'll do the same throughout the Manual.

Check it carefully — especially the quote marks, then

```
RUN ENTER
```

If all was correct, the screen will read:

```
HELLO THERE. I AM YOUR NEW TRS-8Ø MICROCOMPUTER!
YOU CALLED, MASTER. DO YOU HAVE A COMMAND?
```

If it ran OK, answer the question by typing

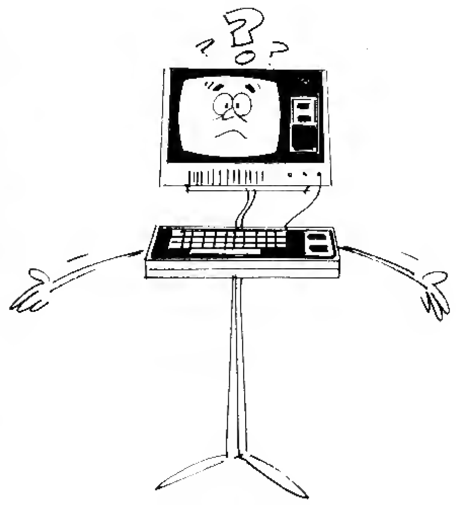
```
YES ENTER
```

Oh — sorry about that! It "bombed", didn't it? The screen said,

```
WHAT?
```

This error message is the result of a built-in troubleshooter which lets you know when you've said the wrong thing (or the right thing at the wrong time), The WHAT? message on the screen says, "No-no, dummy — the program you wrote doesn't have any way for me to accept an answer just because it asked a question" — or words to that effect.

A later lesson will cover another error message. Meanwhile, if you get a `WHAT?`, `HOW?` or `SORRY`, go back and examine the program for an error. Your "YES" answer here was used purposely to show an error message. Later on, we'll program the Computer to accept a "YES" or "NO" answer and act on it.



"WHAT?"

And the Program Grows

It is customary, traditional (and all that) to space the lines in a program 1∅ numbers apart. Note that your two-line program has the numbers 1∅ and 2∅. The reason . . . it's much easier to modify a program if you leave room to insert new lines in-between the old ones. There is no benefit to numbering the lines more closely (like 1,2,3,4). **Don't do it.**

Look at the Video Display. Let's decide we'd rather not have the two lines so close together, but would like to have space between them. Type in the new line:

```
15 PRINT ENTER
```

Then

```
RUN ENTER
```

It should now read:

```
HELLO THERE. I AM YOUR NEW TRS-8∅ MICROCOMPUTER!

YOU CALLED, MASTER. DO YOU HAVE A COMMAND?
```

Looks neater, doesn't it? But what about line 15??? It says PRINT. PRINT what???? Well – print nothing. That's what followed PRINT , and that's just what it printed. But in the process of printing nothing it automatically activated the carriage return, and inserted a space between the printing ordered in lines 1∅ and 2∅. So *that's* how we insert a space.

Didn't that room between lines 1∅ and 2∅ come in handy?

Another important statement is REM, which stands for REMARK. It is often convenient to insert REMarks into a program. Why? So you or someone else can refer to them later, to help you remember complicated programming details, or even what the program's for and how to use it. It's like having a scratch-pad or notebook built-in to your program. When you tell the Computer to execute the program by typing RUN and **ENTER** , it will skip right over any numbered line which begins with the statement REM. The REM statement will have no effect on the program. Insert the following:

```
5 REM *THIS IS MY FIRST COMPUTER PROGRAM* ENTER
```

then

```
RUN ENTER
```

The run should read just like the last run, totally unaffected by the presence of line 5. Did it?

You might be wondering why the asterisks(*) in line #5? The answer is . . . they're just for decoration: *let's give this operation some class!* Remember, anything that is typed on a line following REM is ignored by the Computer.

Well, this programming business is getting complicated and I've already forgotten what is in our "big" program. How can we get a listing of what our program now contains? Easy. A new BASIC command. Type

```
LIST ENTER
```

The screen should read:

```
 5 REM *THIS IS MY FIRST COMPUTER PROGRAM*
10 PRINT "HELLO THERE. I AM YOUR NEW TRS-80 MICROCOMPUTER!"
15 PRINT
20 PRINT "YOU CALLED, MASTER. DO YOU HAVE A COMMAND?"
```

You can call for a LIST any time the prompt appears on the screen.

Where is the END of the program?

The end of a program is, quite naturally, the last statement you want the Computer to execute. Most computers require you to place an END statement after this point, so the computer will know it's finished. But with your TRS-80, an END statement is optional — you can put it in or leave it out. Remember though, if you want to run your BASIC programs on fussier computers, you'll probably need the END statement.

When we get into more complex programs, you'll want to use END statements to force the Computer to stop at specified points — so actually, END comes in very handy even with the TRS-80.

Let's take a close look at END. By the rules governing its use, most dialects of BASIC **which require** END insist that it be the last statement in a program, telling the computer "That's all, folks." By tradition, it is given the number 99, or 999, or 9999 (or larger), depending on the largest number the specific computer will accept. Your RADIO SHACK computer accepts Line numbers up to 32767.

Let's add an END statement to our program.
Type in:

```
99 END ENTER
```

then

```
RUN ENTER
```

The sample run should read:

```
HELLO THERE. I AM YOUR NEW TRS-80 MICROCOMPUTER!

YOU CALLED, MASTER. DO YOU HAVE A COMMAND?
```

"Why didn't the word END print?" **Answer**: Because nothing is printed unless it is the "object" of a PRINT statement. So how could we get the Computer to print THE END at the end of the program execution? Think for a minute before reading on.

```
98 PRINT " THE END"
```

This will work if line #98 is the last PRINT statement in your program.

Erasing Without Replacing

Just for fun, let's move the END statement from line 99 to the largest usable line number, 32767. This requires two steps.

The first is to erase line 99. Note that we're not just making a change or correcting an error in line 99 — we want to completely eliminate it from the program. Easier done than said: Type:

```
99
```

Then **ENTER**

The line is erased. How can we be sure? Think about this now. Got it??? Sure — "pull" a LIST of the entire program by typing

```
LIST ENTER
```

The screen should show the program with lines 5, 1∅, 15, 2∅ and 98. .99 should be gone. Any entire line can be erased the same way.

The second step is just as easy. Type

```
32767 END ENTER
```

. . . and the new line is entered. Pull a listing of the program to see if it was. Was it??? Now RUN the program to see if moving the END statement changed anything. Did it??? It shouldn't have.

Other Uses for END

Move END from #32767 to line #17, then RUN. What happened? It ENDed the RUN after printing line 1∅ and a space. RUN it several times.

Now move END to line 13 and RUN . Then to line 8 and RUN. Do you see the effect END has, depending where it is placed (even temporarily) in a program?

Another Error Message

Let's cause a different error message to appear. Move the optional END statement from line 8 to line 5ØØØØ. The Computer should come back with an error message

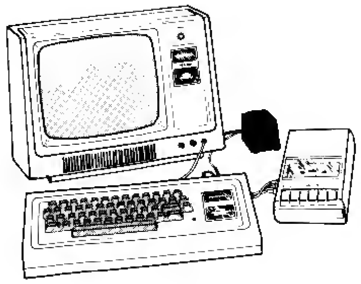
```
HOW?
```

It is saying "I am very patient with you humans and will obey your every command as long as it is within my ability. Line numbers above 32767 are beyond my ability, so just HOW do you expect me to obey?" Pretty smart, this computer.

In general, a HOW? message means, "I understand your instructions, but they're asking me to do something that's impossible." The WHAT? error message, on the other hand, means, "I don't understand your instruction — either the grammar is wrong or you're using words that aren't in my vocabulary."

Learned in Lesson 2

Commands	Statements	Miscellaneous
LIST	PRINT (Space)	Error Messages
	REM	WHAT?
	END	HOW?
		Line Numbering



Notes:

Chapter 3

"But Can It Do Math?"

Yes, it can. Basic arithmetic is a snap for the TRS-80. So are highly complex math calculations — when you write special programs to perform them. (More on this later.)

LEVEL I BASIC uses the four fundamental arithmetic operations, plus a fifth which is just a modifications of two of the others.

1. Addition, using the symbol **+**
2. Subtraction, using the symbol **—**
   (See - nothing to this. Just like grade school. I wonder whatever happened to old Miss . . . Well, ahem – anyway)
3. Multiplication, using the special symbol *
   (Oh drat, I knew this was too easy to be true!)
4. Division, using the symbol **/**
   (Well, at least it's simpler than the old ÷ symbol)
5. Negation (meaning "multiply-times-minus-one"), using the symbol **—**

Of course, we also need that old favorite, the equals sign (=). But wait - - the BASIC language is particular about how we use this sign! Math expressions (like 1 + 2 * 5) can only go on the right-hand side of the equals sign; the left-hand side is reserved for the "variable name". This is the name we give to the result of the math expression. (This all may seem a little strange, but it's really quite simple, as you'll discover in the next few pages.)

Now that wasn't too bad, was it? Be careful. You **cannot** use an "X" for multiplication. Unfortunately, a long time ago a mathematician decided to use "X", which is a letter, to mean multiply. We use letters for other things, so it's much less confusing to use a "*" for multiplication. Confusion is one thing a computer can't tolerate.

So, to computers, "*" is the only symbol which means multiply. After using it a while, you, too, may feel we should do away with X as a symbol for multiplication.

Putting all this together in a program is not difficult, so let's do it. First, we have to erase the "resident program" from the Computer's memory.

"Resident program" is computer talk for "what's already in there".

Type the command

```
NEW ENTER
```

then type

```
LIST ENTER
```

to check that there's nothing left in memory. The Computer should come back with a simple >.

Putting the Beast to Work

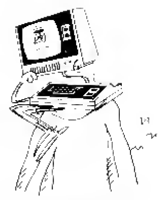
We will now use the Computer for some very simple problem-solving. That means using equations — *oh – panic.* But then, an equation is just a little statement that says what's on one side of the equals sign amounts to the same as what's on the other side.

That can't get too bad (it says here).

We're going to use that old standby equation,

"Distance traveled equals Rate of travel times Time spent traveling."

If it's been a few years, you might want to sit on the end of a log and contemplate that for awhile.



"HMMM"

To shorten the equation, lets choose letters (called variables) to stand for the three quantities. Then we can rewrite the equation as a BASIC statement acceptable to the TRS-80:

```
40 D = R * T
```

Remember, we have to use the * for multiplication.

What's that 40 doing there? That's the program line number. Remember, every step in a program has to have one. We chose 40, but another number would have done just as well. The extra spaces in the line are there just to make the equation easier for us to read; the TRS-80 ignores them. Later, when you write very long programs, you'll probably want to eliminate extra spaces, because they take up memory space. For now, they may be helpful, so leave them in.

Here's what line 40 means to the Computer: "Take the values of R and T, multiply them together, and assign the resulting value to the variable D. So until further notice, D is equal to the result of R times T."

We could not reverse the equation and write, R*T = D. This would have no meaning for the Computer. Remember, the left hand side of the equation is reserved for variable names (whichever letter we choose). The right hand side is the place to put math expressions involving numbers, operators, and known variables.

We can use any of the 26 letters from A through Z to identify the values we know as well as those we want to figure out. Whenever you can, it's a good idea to chose letters that remind you of the things they stand for — like the D, R, and T of the Distance, Rate, Time equation.

To further complicate this very simple example, we will point out now that there's an optional way of writing the equation, using the BASIC statement LET:

```
40 LET D = R * T
```

This use of LET reminds us that making D equal R times T was our choice, rather than an eternal truth like 1 + 1 = 2. Some computers are fussy, and always require the use of LET with programmed equations. Your TRS-80 says, "Have it your way".

Okay — let's complete the program.

Assume:

Distance (in miles) = Rate (in miles per hour) multipled by Time (in hours). How far is it from Boston to San Diego if a jet plane traveling at an average speed of 500 miles per hour makes the trip in 6 hours?

(Yes, I know you can do that one in your head but that's not the point!)

Type in the following:

```
10 REM * DISTANCE, RATE, TIME PROBLEM * ENTER
20 R = 500 ENTER
30 T = 6 ENTER
40 D = R * T ENTER
```

Check the program carefully, then

```
RUN ENTER
```

Hum de dum . . . . ho-hum. . . . . (this sure is a slow computer).

```
READY
```

All it says is READY . *The Computer doesn't work!*

Yes it does. **It worked just fine**. The Computer multiplied 5ØØ times 6 just like we told it, and came up with the answer of 3ØØØ miles. But we forgot to tell it to give *us* the answer. Sorry about that.

Yes, yes . . . we know the distance from Boston to San Diego is closer to 2500 miles — but we took a quick detour via Bermuda (besides, 3000 is an easier number to be working with)!

Can you finish this program without help? It only takes one more line. Give it a good try before reading on for the answer. That way, the answer will mean more to you. (Hint: We've already used PRINT to print messages in quotes. What would happen if we said 5Ø PRINT "D"? . . . No, we want the **value** of D, not "D" itself. Hmmm, what happens when we get rid of the quotes?)

DON'T READ BEYOND THIS POINT UNTIL YOU'VE WORKED ON THE ABOVE EXERCISE!

Look in Part B of this Manual for an answer for this 1st Exercise. Also some notes and ideas.

Well, the answer of 3ØØØ is correct, but its "presentation" was no more inspiring than the printout from a hand calculator. This inevitably leads us back to where we first started this foray into the unknown — the PRINT statement.

Note that we said in line 5Ø PRINT D. There were no quotes around the letter D like we had used before. The reason is simple but fairly profound. If we want the Computer to print **the exact words** we specify, we enclose them in quotes. If we want it to print the **value** of a variable, in this case D, we leave the quotes off. That simple message is worth serious thought before continuing on.

Did you think seriously about it?! . . . Then on you go!

Now suppose we want to include both the value of something *and* some exact words on th same line. Pay attention, as you will be doing more and more program design yourself, and PRINT statements give beginners more trouble than any other single part of computer programming. Type in the following:

```
50 PRINT "THE DISTANCE (IN MILES) IS", D  ENTER
```

(REMEMBER: Typing in a statement with a line number that already is in use erases the original statement entirely — and that's what we want to do here.)

Then

```
RUN ENTER
```

The display should appear:

```
THE DISTANCE (IN MILES) IS          3000
```

How about that! The message enclosed in quotes is printed exactly as we specified, and the letter gave us the value of D. The comma told the Computer that we wanted it to print two separate items on the same line. We can tell it to print up to four items on the same line, simply by inserting commas between them.

With this in mind, see if you can change line 50 so the computer finishes the program with the following message:

```
THE DISTANCE IS    3000                                MILES.
```

Break up the quoted message into two parts, and put the variable in between them on the PRINT line.

```
50 PRINT "THE DISTANCE IS", D, "MILES."
```

Now what about all that extra space on the printout line? The reason for it is that the computer divides up the screen width into four zones of 15 characters each. When a PRINT statement contains two or more items separated by commas, the computer automatically prints the items in different print zones. Automatic zoning is a very convenient method of outputting tabular information, and we'll explore the subject further later on.

It's possible to eliminate all that extra space in the output from our Distance, Rate, Time program. Retype the last version of line 50, substituting semi-colons (;) for commas throughout the line.

(Careful — don't replace the period with a semi-colon.)

```
RUN ENTER
```

The display should appear:

```
THE DISTANCE IS 3ØØØ MILES.
```

Look carefully at program line 5Ø. There's no unused space between the S in IS, the D, and the M in MILES. But in the printout on the display, there is a space between IS and 3ØØØ, and another space between 3ØØØ and MILES. How come?

Reason: A semicolon automatically inserts one space between the two items it is separating. As you do more programming, this point will become important.

WHEW!

Well, we have already covered more than enough commands, statements and math operators to solve myriads of problems.

Math operators? — they're the =, +, −, * and / symbols we talked about earlier.

Now let's spend some time actually writing programs to solve problems. There is no better way to learn than by doing, and everything covered so far is fundamental to our success in later Chapters. So don't jump over these exercises — it's the best way to get you into the thick of programming. You'll find sample answers in Part B, along with further comments.

EXERCISE 3-2: Write a program which will find the time required to travel by jet plane from San Diego to Boston, if the distance is 3ØØØ miles and the plane travels at 5ØØ MPH.

EXERCISE 3-3: If the circumference of a circle is found by multiplying its diameter times π, (3.14) write a program which will find the circumference of a circle with a diameter of 35 feet.

EXERCISE 3-4: If the area of a circle is found by multiplying π times the square of its radius, write a program to find the area of a circle with a radius of 5 inches.

EXERCISE 3-5: Your checkbook balance was $225. You've written three checks (for $17, $35 and $225) and made two deposits ($4Ø and $2ØØ). Write a program to adjust your old balance based on checks written and deposits made, and print out your new balance.

Remember, you can use any of the 26 letters, not just D, R and T (they were just convenient for our problem).

Learned in Chapter 3

Statements	Math Operators	Miscellaneous
LET	=	,
	+	;
	—	A-Z variables
	*	
	/	

Chapter 4

Are There More Stars or Grains of Sand?

In this mathematical world we are blessed with very large and very small numbers. Millions of these and billionths of those. To cope with all this, your Computer uses "exponential notation", or "standard scientific notation" when the number sizes start to get out of hand. The number 5 million (5,ØØØ,ØØØ), for example, can be written "5E+Ø6". This means, "the number 5 followed by six zeros."

If an answer comes out "5E-Ø6", that means we must shift the decimal point, which is after the 5, six places to the left, inserting zeroes as necessary. Technically, it means 5 X 10^{-6}, or 5 millionths, (.ØØØ,ØØ5). It's really pretty simple once you get the hang of it, and a lot easier to keep track of numbers without losing the decimal point. Since the Computer insists on using it with very large and very small numbers, we can just as well get in the good habit, too.

Type NEW before performing the following exercises.

EXERCISE 4-1: If one million cars drove ten thousand miles in a certain year, how many miles did they drive altogether that year? Write and run a simple program which will give the answer.

EXERCISE 4-2: Changes lines 2Ø and 3Ø in the Car Miles Solution program (from Exercise 4-1) to express the numbers written there in exponential notation, or SSN (Standard Scientific Notation). Then RUN it.

Or technically, 5*10^6, which is 5 times ten to the sixth power:

5*1Ø*1Ø*1Ø*1Ø*1Ø*1Ø

Now you can see the value of scientific notation!

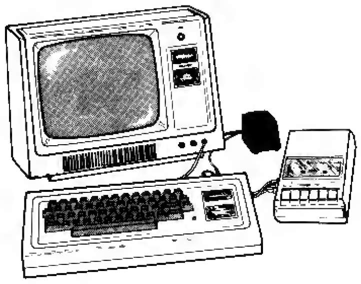
In our BASIC, that's 5/1Ø/1Ø/1Ø/1Ø/1Ø/1Ø

Didn't forget the **ENTER** did you? Up till now we've been reminding you that you have to enter each line or command — but from now on, we'll assume you've got that little routine matter down pat.

Learned in Chapter 4

Miscellaneous

E−notation



(E stands for "exponent" and in our case it refers to the exponent of 10 — ie, the number of zeros to the right or left of the main number.)

Chapter 5

Using ( ) and the Order of Operations

Parentheses play an important role in computer programming, just as in ordinary math. They are used here in the same general way, but there are important exceptions.

1. In BASIC, parentheses can enclose operations to be performed. Those operations which are within parentheses are performed before those not in parentheses.
2. Operations buried deepest within parentheses (that is, parentheses inside parentheses) are performed first.
3. When there is a "tie" as to which operations the Computer should perform first after it has removed all parentheses, it works its way along the program line from left to right doing the multiplication and division. It then starts at the left again and performs the addition and subtraction.

NOTE: INT. RND and ABS functions are performed before multiplication and division. (We haven't talked about these yet, but just to be complete . . .)

4. A. problem listed as (X) (Y) will NOT tell the Computer to multiply. X * Y is for multiplication.

Example: To convert temperature in Fahrenheit to Celsius (Centigrade), the following relationship is used:

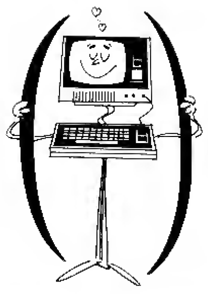
The Fahrenheit temperature equals 32 degrees plus nine-fifths of the Celsius temperature.

Or, maybe you're more used to the simple formula —

$$F^\circ = \frac{9}{5} \text{ X } C^\circ + 32$$

Assume we have a Celsius temperature of 25°. Type in this program and RUN it.

```
10 REM * CELSIUS TO FAHRENHEIT CONVERSION *

20 C = 25

30 F = (9/5)*C + 32

40 PRINT C; "DEGREES CELSIUS =" ;F;"DEGREES FAHRENHEIT."
```



"FRIENDS."

If you want to be sure your problems are calculated correctly, use ( ) around operations you want performed first.

Recall the old memory aid, "My Dear Aunt Sally"? In math you are supposed to do Multiplication and Division first (from left to right), then come back for Addition and Subtraction (left to right). The TRS-80 uses the same sequence.

Sample Run:

```
25 DEGREES CELSIUS = 77 DEGREES FAHRENHEIT.
```

First notice that line 4Ø consists of a PRINT statement followed by four separate expressions — two variables and two groups of words in quotes called "literals" or "strings".

Remember what the semi-colons are for?

Next, note how the parentheses are placed in line 3Ø. With the 9/5 secure inside, we can multiply its quotient times C, then add 32.

Now, remove the parentheses in line 3Ø and RUN again. The answer comes out the same. Why?

1. On the first pass, the Computer started by solving all problems within parentheses, in this case just one (9/5). It came up with (but did not print) 1.8. It then multiplied the 1.8 times the value of C and added 32.
2. On our next try, without the parentheses, the Computer simply moved from left to right performing first the division problem (9 divided by 5), then the multiplication problem (1.8 times C), then the addition problem (adding 32). The parentheses really made no difference in our first example.

Next, change +32 to 32+ and move it to the front of the equation in line 3Ø. Run it again, without parentheses.

Did it make a difference in the answer? Why not?
Answer: Execution proceeds from left to right, multiplication and division first, then returns and performs addition and subtraction. This is why the 32 was **not** added to the 9 before being divided by 5. *Very important!* If they had been added, we would of course have gotten the wrong answer.

EXERCISE 5-1: Write and run a program which converts 65° Fahrenheit to Celsius. The rule tells us that "Celsius temperature is equal to five-ninths times what's left after 32° is subtracted from the Fahrenheit temperature."

$$C^\circ = (F^\circ - 32) \times \left[\frac{5}{9}\right]$$

EXERCISE 5-2: Remove the first set of parentheses in the #5-1 answer and run again.

EXERCISE 5-3: Replace the first set of parentheses in program line 3Ø and remove the second pair of parentheses, then RUN. Note how the answer comes out — correctly!

EXERCISE 5-4: Insert brackets in the following equation to make it correct. Write a program to check it out on the TRS-80.

3Ø - 9 - 8 - 7 - 6 = 28

Learned in Chapter 5

Miscellaneous

( )

Order of Operations

Notes:

Chapter 6

IF **you liked** Chapter 1 **through** 5, *THEN* **you're going to love the rest of this book!**

Because we're really just getting into the good stuff. Like IF-THEN and GOTO statements that let your Computer make decisions and take . . . er, executive action. But first, a few more operators . . .

Relational operators allow the Computer to compare one value with another. There are only three:

1. Equals, using the symbol =
   (How'd you guess?)
2. Is greater than, using the symbol >
3. Is less than, using the symbol<

Combining these three, we come up with three more operators:

4. Is not equal to, using the symbol< >
5. Is less than or equal to, using the symbol <=
6. Is greater than or equal to, using the symbol >=

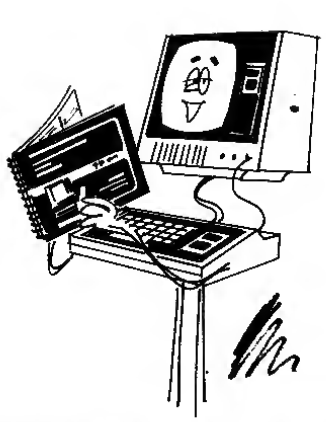
By adding these six relational operators to the four **math** operators we already know, plus new *STATEMENTs,* called IF-THEN, & GOTO, we create a powerful system of comparing and calculating that becomes the central core of everything else that follows.

The IF-THEN statement, combined with the six relational operators above, gives us the **action** part of a system of logic. Enter and RUN this program:

```
10 A = 5

20 IF A = 5 THEN 50

30 PRINT "A DOES NOT EQUAL 5."

40 END

50 PRINT " A EQUALS 5."
```

The screen should display:

```
A EQUALS 5.
```



"THEY GOTTA MAKE IT A MOVIE!"

Example: A< B means A is less than B. To help you distinguish between< and >, just remember that the smaller part of the< symbol points to the smaller of the two quantities being compared.

Now let's examine the program line by line.

Line 1∅ establishes the fact that A has a value of 5.

Line 2∅ is an IF-THEN statement which directs the Computer to go to line 5∅ IF the value of A is exactly 5, *skipping over whatever might be inbetween lines 2∅ and 5∅.* Since A does equal 5, the Computer jumps to line 5∅ and does as it says, printing A EQUALS 5. Line 3∅ and 4∅ are not used at all in this case.

Now, change line 1∅ to read:

```
1∅ A = 6
```

and RUN

The run should say:

```
A DOES NOT EQUAL 5.
```

Taking it a line at a time:

Line 1∅ establishes the value of A to be 6.

Line 2∅ tests the value of A. If A equals 5, THEN the Computer is directed to go to line 5∅. But "the test fails", that is, A does **NOT** equal 5, so the Computer proceeds as usual to the next line, line 3∅.

Line 3∅ directs the Computer to print the fact that A DOES NOT EQUAL 5. It does not tell us what the **value** of A is, only that it does not equal 5. The Computer then proceeds on to the next line.

Line 4∅ ENDs the program's execution. Without this statement separating lines 3∅ and 5∅, the Computer would charge right on to line 5∅ and print its contents, which obviously are in conflict with the contents of line 3∅. This is an example of using an IF-THEN statement with only the most fundamental relational operator, the equals sign.

Now let's see if you can accomplish the same thing by using the "does not equal" sign:

EXERCISE 6-1: Rewrite the resident program using a "does not equal" sign in line 2Ø instead of the equals sign, changing other lines as necessary, so the same results are achieved with your program as with the one in the Example.

EXERCISE 6-2: Change line 1Ø to give A the value of 6. Leave the other four lines from #6-1 as shown. Add more program lines as necessary so the program will tell us whether A is larger or smaller than 5 and RUN.

EXERCISE 6-3: Change the value of A in line 1Ø at least three more times, running after each change to ensure that your new program works correctly.

No sample answers are given since you are choosing your own values of A. It will be obvious whether or not you are getting the right answer.

The IF-THEN statement is what is known as a *CONDITIONAL branching* statement. The program will "branch" to another part of the program **on the condition that** it passes the test it contains. If it fails the test, the program simply continues to the next line.

A statement called GOTO is known as an *UNCONDITIONAL branching* statement. If we were to replace lines 4Ø and 8Ø with GOTO 99, and add line 99:

```
99 END
```

. . . whenever the Computer hit line 4Ø or 8Ø it would **unconditionally** follow orders and go to 99, ENDing the run. While your Radio Shack Computer is rather broad-minded when it comes to accepting these various BASIC dialects, many computers are not. For practice, change lines 4Ø, 8Ø and 99 as discussed above and

Note: Do Not leave a space between GO and TO (some forms of BASIC use two words — Radio Shack's BASIC wants only one word).

Did the program work OK as changed? Did you try it with several values of A? Be sure you do so! We will find many uses for the GOTO statement in the future.

Learned in Chapter 6

Statements	Relational Operators	Miscellaneous
IF-THEN-ELSE	=	Conditional branching
GOTO	>	Unconditional branching
	<	
	<>	
	<=	
	>=	

Chapter 7

It Also Talks and Listens

Begin this lesson by typing in the sample answer program to Exercise #6-2:

By now you have probably gotten tired of having to retype line 1Ø over each time you wish to change the value of A. The *INPUT* statement is a simple, faster and more convenient way to accomplish the same thing. It's a biggie, so don't miss any points.

Add the following lines to the *resident* program:

Resident — remember, that's the program that is now residing in the Computer.

```
 5 PRINT " THE VALUE I WISH TO GIVE A IS"

1Ø INPUT A
```

Now RUN

The Computer should print:

```
THE VALUE I WISH TO GIVE A IS

?_
```

See the question mark on the screen. It means, "It's your turn — and I'm waiting . . ."

Enter a number and see what happens. It should be identical to what happened when you typed in the same number earlier by **changing** line 1Ø. Run the program several more times to get the feel of the INPUT statement.

Pretty powerful, isn't it?

Let's add a touch of class to the INPUT process by retyping line 5 as follows:

```
5 PRINT " THE VALUE I WISH TO GIVE A IS";
```

Look at that line very carefully. Do you see how it differs from the earlier line 5??? It is different . . . . A semicolon has been added at the **end** of the line.

Think back a bit now. We used semicolons before in PRINT statements, but only in the middle to hook several of them together so they would print close together on the same line. In this case, we put a semicolon at the end, so the question mark from the next line wil print on the same line, rather than down there by itself. After changing line 5 as above, RUN it. It should read:

```
THE VALUE I WISH TO GIVE A IS?_
```

Please note that you cannot use a semicolon indiscriminately at the end of a PRINT statement. It is only meant to hook two lines together, both of which have printing to be done. The INPUT line prints the question mark. We shall see later where two long lines starting with PRINT can be connected together by the trailing semicolon so as to print on the same line.

Your Radio Shack TRS-80 *Interpreter* is, as has been mentioned, able to speak "The King's Basic" as well as a variety of dialects. The first of the many "short-cut" dialects we will be exploring throughout these lessons involves combining PRINT and INPUT into one statement. Change line 5 to read:

Interpreter — is the internal circuit that allows you to "talk" to the TRS-80 in English (BASIC) and it can talk to you.

Sometimes the word dialect is used when talking about the different forms of a computer language. Just as with dialects in "human" languages, there can be slight differences in word uses, etc. in BASIC. (Radio Shack's BASIC is totally compatible with the Dartmouth BASIC — the original BASIC. But we do have some handy short cuts, so we might call them a "dialect".) We'll sometimes refer to this as a shortcut and sometimes as a dialect.

```
5 INPUT " THE VALUE I WISH TO GIVE A IS" ;A
```

then delete line 1Ø by typing

```
1Ø
```

then RUN.

The results come out exactly the same, don't they? Here is what you have changed:

1. PRINT to INPUT
2. Both statements on the same line
3. Eliminated the extra line

In the long programs which you will be writing, running and converting, this shortcut will be valuable.

Up to now, all our programs have been strictly one-shot affairs. You type RUN , the Computer executes the program, prints the results (if any) and comes back with a READY . To repeat the program, you have to type in RUN again. Can you think of another way to get the Computer to execute a program two or more times?

No — don't enlarge the program by repeating its steps over and over again — that's not very creative!

We'll answer that question by upgrading our Celsius-to-Fahrenheit conversion program (Chapter 5). If you think GOTO is a powerful statement in everyday life, wait till you see what it does for a computer program!

Type NEW and the following:

```
10 REM * IMPROVED CELSIUS TO FAHRENHEIT CONVERSION PROGRAM *

20 INPUT "WHAT IS THE TEMPERATURE IN DEGREES CELSIUS";C

30 F = (9/5)*C + 32

40 PRINT C; "DEGREES CELSIUS = ";F; "DEGREES FAHRENHEIT."

50 GOTO 20
```

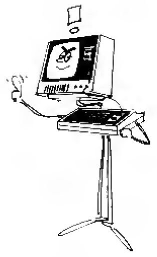
and RUN.

The Computer will keep on asking for more until you get tired or the power goes off (or some other event beyond its control). This is the kind of thing a Computer is best at — doing something over and over again. Modify some of the other programs to make them self-repeating. You'll find they're much more useful that way.

These have been 7 long and "meaty" lessons, so go back and review them all again, repeating those assignments where you feel weak. We are moving out into progressively deeper water, and it is complete mastery of these **fundamentals** that is your life preserver.

Learned in Chapter 7

Statements	Miscellaneous
INPUT and INPUT with built-in PRINT	; Trailing semicolon


"GO TO!"

You'll have to hit BREAK to get out of the program loop.

"I CAN DO THIS ALL DAY"

Notes:

Chapter 8

Two Easy Features

The Calculator Mode

Before continuing our exploration of the nooks and crannies of our Computer — acting as a **computer**, we should be aware that it also works well as a **calculator**. If you **omit** the line number before certain commands, the Computer will execute them, print the answer on the screen, then erase the command you entered. What's more, it will work as a calculator even when a computer program is loaded, **without disturbing that program**. All you need, to be in the calculator mode, is the prompt >.

Example: How much is 3 times 4? Type in

```
PRINT 3 * 4
```

. . . the answer comes back

```
12
```

Example: How much is 345 divided by 123?

```
PRINT 345/123
```

. . . the answer is

```
2.80488
```

Spend a few minutes making up routine arithmetic problems of your own, using the calculator mode to solve them. Any arithmetic expression you might use in a program can also be evaluated in the calculator mode. This includes parentheses and chain calculations like A*B*C.

Try the following problem:

```
PRINT (2/3)*(3/2)
```

The answer comes back:

```
1.0000009
```

```
  70 SET(RND(127),RND(47))
  72 N.Z
  80 I=0
1000 P.AT525, "WHOSE WOODS THESE ARE I THINK I KNOW.";
1001 GOSUB6000
1100 P.AT525," HIS HOUSE IS IN THE VILLAGE, THOUGH";
1101 GOSUB6000
1200 P.AT525," HE WILL NOT SEE ME STOPPING HERE   ";
1201 GOSUB6000
1300 P.AT525," TO WATCH HIS WOODS FILL UP WITH SNOW";
1301 GOSUB6000
1400 P.AT525," MY LITTLE HORSE MUST THINK IT QUEER ";
1401 GOSUB6000
1500 P.AT525," TO STOP WITHOUT A FARMHOUSE NEAR    ";
1501 GOSUB6000
1600 P.AT525," BETWEEN THE WOODS AND FROZEN LAKE   ";
1601 GOSUB6000
1700 P.AT525," THE DARKEST EVENING OF THE YEAR.     ";
1701 GOSUB6000
1800 P.AT525," HE GIVES HIS HARNESS BELLS A SHAKE ";
1801 GOSUB6000
1900 P.AT525," TO ASK IF THERE IS SOME MISTAKE.";
1901 GOSUB6000
2000 P.AT525,"THE ONLY OTHER SOUND'S THE SWEEP";
2001 GOSUB6000
2100 P.AT525," OF EASY WIND AND DOWNYFLAKE.    ";
2101 GOSUB6000
2200 P.AT525," THE WOODS ARE LOVELY, DARK  AND DEEP";
2201 GOSUB6000
2300 P.AT589," BUT I HAVE PROMISES TO KEEP.";
2305 I=3
2310 GOSUB6000
2400 P.AT653," AND MILES TO GO BEFORE I SLEEP.";
2405 I=6
2410 GOSUB6000
2500 P.AT717," AND MILES TO GO BEFORE I SLEEP.";
2505 I=9
2510 GOSUB6000
5000 SET(RND(127),RND(47))
5001 G.5000
6000 F.N=1TO20
6020 X=RND(127)
6030 Y=RND(47)
6070 IF Y = 24+IG.6020
6080 IF Y = 25+I G.6020
6090 IF Y = 26+I G.6020
6100 SET(X,Y)
6150 F.A=1TO20:N.A
6200 N.N
6300 RETURN
```

Termites

A malicious sense of humor helps on this one. Its avowed purpose is to demonstrate the graphic RESET (X, Y) function, turning off the "lights" in a random fashion, but it's not without other redeeming value. If you don't like to sit by the fire and watch it snow while reading good poetry, you can always watch the termites eat your house down.

```
 40 CLS
 50 F.X=1TO127
 60 F.Y=3TO47
100 SET(X,Y)
120 N.Y:N.X
```

```
170 N=5715
180 P."            SEE THE TERMITES EAT.    ONLY";
185 P.AT45," BITES LEFT!";
200 X=RND(127)
220 Y=RND(45) + 2
300 IF POINT(X,Y)=0 G.200
500 RESET(X,Y)
550 N=N-1
600 P.AT36,N;
700 IF N=0 G.999
800 G.200
999 G.999
```

Sorry

SORRY is a popular board game by Parker Brothers. This program demonstrates how to load a deck of cards into a numerical array, draw them out in a random fashion, "reshuffle" the deck after the last card is drawn, and continue drawing. You may specify how many seconds delay you wish between each drawing of the cards, allowing as much time as desired to actually move the pieces on your own SORRY board. Have fun!

```
10 REM * RANDOM GENERATOR FOR GAME OF SORRY *
11 IN. "ENTER A NUMBER FROM 1 TO 100" ;N
12 F. I=1 TO N:J=RND(32767):N.I
15 CLS
20 P." STAND BY FOR THE SHUFFLING OF THE DECK OF CARDS."
21 P.
22 P.
30 P.
40 FOR N=1 TO 45
50 READ A(N)
60 NEXT N
66 P.:P.:P.
70 Y=1
75 P. "SHUFFLING COMPLETED . . . GAME CONTINUES!"
80 B=0
90 GOTO 110
100 B=35
110 R=INT(RND(45))
120 M=A(R)
130 IF M=0 GOTO 110
140 A(R) = 0
150 T=0
160 FOR Z=1 TO 45
170 T=A(Z) + T
180 NEXT Z
185 P.T.(27)," PRESS ENTER" ;:IN. A$
190 IF T=0 GOTO210
200 GOTO 240
210 P." END OF DECK.  THE CARDS ARE BEING RESHUFFLED."
220 RESTORE
230 GOTO 30
240 IF Y < 0 G.270
250 P.TAB(10);" RED"
260 GOTO 280
270 P. TAB(40);" GREEN"
280 IF M = 13 GOTO 300
290 P. TAB(B+15);M
300 ON M GOTO 320,340,590,380,590,590,400,590,590,430,450,590,470
310 GOTO 590
320 P.TAB(B);" MAY MOVE A NEW PIECE OUT"
330 GOTO 590
340 P.TAB(B);" MAY MOVE A NEW PIECE OUT"
345 P.:P.
350 P. TAB(B+5);" DRAW AGAIN . . . . "
```

```
360 P.
370 GOTO 630
380 P.TAB(B);" MUST BACK UP 4 SPACES"
390 GOTO 590
400 P.TAB(B);" MAY SPLIT THE 7 BETWEEN"
410 P.TAB(B+3);" 2 PIECES"
420 GOTO 590
430 P.TAB(B);" MAY MOVE BACKWARDS 1 SPACE"
440 GOTO 590
450 P. TAB(B);" CAN SWAP PIECES WITH OPPONENT"
460 GOTO 590
470 P.
480 P.
490 IF B=0 GOTO 550
500 P. "  GOTCHA      <<<---<<<   <<<---<<<" ;
510 P.TAB(49);" S O R R Y !"
520 P.
530 P.
540 GOTO 590
550 P.  " S O R R Y !  >>>--->>>   >>>--->>>" ;
560 P. TAB(55);" GOTCHA !"
570 P.
580 P.
590 FOR X=1TO4
600 P. TAB(30);" *"
610 NEXT X
620 Y=Y*(-1)
630 IF Y>0 THEN 80
640 GOTO 100
650 D.1,1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,7,7,7,7,8,8
660 D.8,8,10,10,10,10,11,11,11,11,12,12,12,12,13,13,13,13
```

Automatic Ticket Number Drawer

Like to make a big splash at the next Rotary Club, Country Fair, or other ticket drawing giveaway? This program uses the random number generator to pick the lucky number(s) and eliminate charges of stuffing the ticket box, besides giving the whole affair some pizzaz. If your own number comes up and you are charged with rigging the computer, you're on your own.

```
  3 IN. "ENTER A NUMBER FROM 1 TO 100";N
  4 F.I=1 TO N: J=RND(32767):N.I.
  5 CLS
 10 REM * PICKS WINNER(S) BY DRAWING TICKET NUMBER *
 11 REM * NO MORE THAN 32767 TICKETS CAN BE SOLD *
 12 REM * BUT TICKET NUMBERS CAN RANGE TO 999999 & BEYOND *
 40 IN." THE LOWEST TICKET NUMBER IS" ;B
 50 P.
 70 IN." THE HIGHEST TICKET NUMBER IS " ;H
 80 P.
 90 E=H-B+1
 91 IF E<32768 G.110
100 P." TOO MANY TICKETS SOLD!" :END
110 IN." HOW MANY WINNERS DO YOU WANT " ;W
120 CLS
130 IF W > E G.269
140 P.
141 P.
142 P.
143 P.
180 P."*  A N D  T H E  W I N N I N G  ";
182 IF W > 1 G.185
183 P." T I C K E T  I S *"
```

```
184 G.200
185 P."T I C K E T S   A R E   *"
200 P.
205 A(0) = 0
210 FOR N = 1 TO W
220 Z = RND(E)
260 P.
262 P.TAB(12);" >----->>>  ";Z + 8 - 1
264 NEXT N
266 END
269 CLS:P.:P.:P.:P.:P.:P.:P.
270 P.TAB(8);" YOU CAN'T HAVE MORE WINNERS THAN" ;
272 P." ENTRIES - DUMMY !"
```

Craps

The game is as old as history. A testimonial to the intelligence and ingenuity of our ancient ancestors. An excellent way to demonstrate the running of twin Random Number Generators.

You don't need to know how to play the game – the computer will quickly teach you. (. . . There's one born every minute . . .)

```
  1 IN."ENTER A NUMBER FROM 1 TO 100";N
  2 F.I=1TON:J=RND(32767):N.I
 10 REM * CRAPS GAME *
 20 CLS
 30 GOSUB 300:P=N
 40 P.:P."YOU ROLLED ****";A;" AND ";B;"****"
 50 ON P GOTO 60, 120,120, 100,100,100,110,100,100,100,110,120
 60 REM * USED FOR THE ON STATEMENT IF P=1 (WHICH IT CAN'T)*
100 P." YOUR POINT IS" ;N:GOTO 130
110 PRINT " YOU WIN!" :P.:END
120 PRINT " YOU LOSE." :P.:END
130 GOSUB 300:M=N
135 P.:P."YOU ROLLED ****";A;" AND ";B;"****"
140 IF P=M THEN 110
150 IF M=7 THEN 120
160 G.130
300 A=RND(6):B=RND(6):N=A+B:RET.
310 RETURN
```

Fire When Ready, Gridley

You have probably seen this popular graphics display at your Radio Shack Store. It is very well done, and due to popular demand is printed here. Little boys of all ages are fascinated by it, and it's great for showing off your computer. You will want to keep this one on tape for fast loading.

```
CASTLE SHOT

  5 REM * CASTLE SHOT *
 10 INPUT " ENTER YOUR INITIALS" ;A$
 20 CLS
 30 Z=74
 40 FOR Y=17 TO 47
 50 FOR X=Z TO 127
 60 SET (X,Y)
 70 NEXT X
 80 IF Y<23 THEN Z=Z+2
200 NEXT Y
210 FOR X=75 TO 123 STEP 4
220 SET (X,16)
230 SET (X+1,16)
240 NEXT X
250 Q=0
300 FOR X=95 TO 125 STEP 5
```

```
310 FOR Y=47 TO 35 STEP -1
320 RESET (X,Y)
330 NEXT Y
340 NEXT X
400 FOR X=95 TO 125
410 RESET (X,34)
420 NEXT X
500 PRINT AT 688, A$;"'S CASTLE" ;
600 FOR X=73 TO 100
610 SET (X,12)
620 SET (X,13)
630 NEXT X
700 FOR X=85 TO 95
710 SET (X,14)
720 SET (X,15)
730 NEXT X
740 RESET (90,13)
750 RESET (91,13)
1000 FOR Z=1 TO 2
1010 FOR X=2 TO 14
1020 FOR Y=40 TO 43
1030 SET (X,Y)
1040 NEXT Y
1050 NEXT X
1100 FOR X=3 TO 13 STEP 2
1110 RESET (X,41)
1120 NEXT X
1130 RESET (7,43)
1140 RESET (8,43)
1190 REM---THIS WILL MAKE THE CANNON RECOIL
1200 FOR X=1 TO 100:NEXT X
1210 RESET (73,12)
1220 RESET (73,13)
1230 RESET (74,12)
1240 RESET (74,13)
1250 SET (101,12)
1260 SET (101,13)
1270 SET (102,12)
1280 SET (102,13)
1290 FOR X=1 TO 100:NEXT X
1300 SET (74,12)
1310 SET (74,13)
1320 SET (73,12)
1330 SET (73,13)
1340 RESET (102,12)
1350 RESET (102,13)
1360 RESET (101,12)
1370 RESET (101,13)

1500 FOR X=71 TO 2 STEP -1
1510 P=X-73
1520 Y=P*P/150 + 12
1530 SET (X,Y)
1540 SET (X-1,Y)
1600 RESET (X+1,Q)
1610 RESET (X,Q)
1620 Q=Y
1630 NEXT X
1640 PRINT AT 771, "KAPOW!" ;
1700 GOSUB 1900
1710 FOR X=1 TO 18
1720 RESET (X,45)
1730 RESET (X,36)
1740 RESET (X,37)
```

```
175Ø NEXT X
18ØØ NEXT Z
181Ø PRINT AT Ø
182Ø END
19ØØ FOR X=1 TO 1ØØØ
191Ø NEXT X
192Ø RETURN
```

House Security

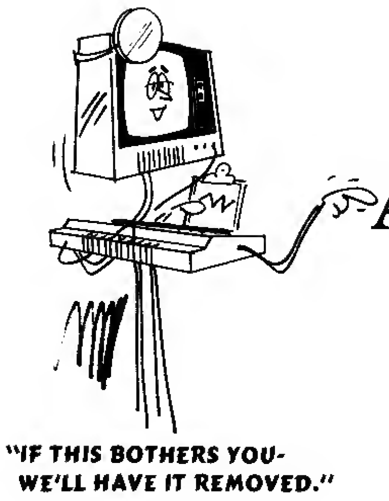
```
 1Ø REM * LOGICAL AND PROGRAM *
 15 CLS
 2Ø Y=1:N=Ø:P." PLEASE ANSWER YES OR NO TO THE FOLLOWING QUESTIONS"
 25 P.
 3Ø INPUT " IS THE FRONT DOOR LOCKED" ;A
 4Ø INPUT " IS THE BACK DOOR LOCKED" ;B
 5Ø INPUT " IS THE KITCHEN WINDOW CLOSED" ;C
 6Ø INPUT " IS THE BEDROOM WINDOW CLOSED AND LOCKED" ;D
 7Ø INPUT " IS THE GARAGE DOOR LOCKED" ;E
 75 P.:P.
 8Ø IF (A=Y)*(B=Y)*(C=Y)*(D=Y)*(E=Y) THEN 12Ø
 9Ø P." HOUSE NOT LOCKED UP FOR THE NIGHT."
 95 P.
1ØØ P." PLEASE CHECK FOR AN UNLOCKED DOOR OR WINDOW."
11Ø END
12Ø P." HOUSE SECURITY CHECK SHOWS HOUSE LOCKED UP FOR THE NIGHT."
13Ø END
```

Loan Amortization

This program provides a fully developed installment plan for the repayment of small-to-moderate size loans, such as car or home improvement loans. The program includes all instructions necessary to using it. Use it with common sense; in the last payment period, amounts may be carried out to a fraction of a cent.

Challenge: modify the program to eliminate fractional-cent payments, without changing the total amount paid as interest or principal.

```
 1Ø C=Ø:CLS:IN. "PRINCIPAL";P
 2Ø IN. "# OF PERIODS";L
 3Ø IN. "INTEREST RATE";R
 4Ø I=R/12:I=I/1ØØ
 5Ø T=1:F.X=1TOL
 6Ø T=T*(1+I):N.X:T=1/T
 7Ø T=1-T
 8Ø M=P*I/T
 85 M=INT(M*1ØØ+.5)/1ØØ
 9Ø GOS.2ØØ
1ØØ F.Z=1TOL
11Ø IFC<13G.12Ø
115 IN. "PRESS ENTER TO CONTINUE";A$:C=Ø:GOS.2ØØ
12Ø A=(INT(P*I*1ØØ+.5))/1ØØ
13Ø B=M-A:P=P-B
14Ø P.Z;:P.T.(1Ø),P;:P.T.(2Ø),M;
15Ø P.T.(3Ø),B;:P.T.(4Ø),A
16Ø C=C+1:N.Z
17Ø END
2ØØ CLS:P. "PAYMENT REMAINING MONTHLY PRINCIPAL INTEREST"
21Ø P. "NUMBER PRINCIPAL PAYMENT PAYMENT PAYMENT"
22Ø RET.
```



Appendix:

Appendix A: Subroutines

These subroutines will let you run programs which require advanced math functions not directly available in LEVEL I BASIC.

If you entered all the subroutines exactly as they're listed, you'd have less than 7ØØ bytes of memory left for your main program — not enough to do much of anything. So just enter the subroutines you need, and omit REM statements if you're still short on space.

Once you've entered a subroutine and gotten it running, save it on a Cassette. Try saving different combinations of subroutines on Cassettes: for example, make a SIN/COS/TAN cassette, a SIN/SQR cassette, an EXPONENTIATION/LOG/EXPONENTIAL/SGN cassette — whatever combinations are useful to you.

Each subroutine listing has a set of instructions in the margin. Study them closely. You'll see that some subroutines require other subroutines for internal calculations. You must enter these "auxiliary subroutines" when the instructions call for them.

Always enter 3ØØØØ END as a protective block when using subroutines. For complete information on the use of subroutines, see Chapter 25.

NOTE: Accuracy of the subroutines is less than the accuracy of LEVEL I math operators and intrinsic functions. This is due to two factors: 1. The subroutines contain many chain calculations, which tend to magnify the small error of individual operations. 2. These subroutines are only approximations of the functions they replace. In general, the subroutines are accurate to five or six decimal places over much of their allowable range, with a decrease in accuracy as the input approaches the upper or lower limits for input values.

Subroutines listed in this Appendix:

- **Square Root**
- **Exponentiation**
- **Logarithms (Natural and Common)**
- **Exponential (Powers of e)**
- **Tangent**
- **Cosine**
- **Sine**
- **ArcCosine**
- **ArcSine**
- **ArcTangent**
- **Sign**

Square Root

Computes: SQR(X), $\sqrt{X}$

Input: X, must be greater than or equal to zero

Output: Y

Also uses: W,Z internally

Other subroutines required: None

How to call: GOSUB 3ØØ3Ø

```
3ØØØØ END
3ØØ1Ø REM *SQUARE ROOT* INPUT X, OUTPUT Y
3ØØ2Ø REM ALSO USES W & Z INTERNALLY
3ØØ3Ø IF X = Ø T. Y = Ø : RET.
3ØØ4Ø IF X>Ø T. 3ØØ6Ø
3ØØ5Ø P. "ROOT OF NEGATIVE NUMBER?" : STOP
3ØØ6Ø Y=X*.5 : Z=Ø
3ØØ7Ø W=(X/Y-Y)*.5
3ØØ8Ø IF (W=Ø) + (W=Z) T. RET.
3ØØ9Ø Y=Y+W : Z=W : G. 3ØØ7Ø
```

Exponentiation

Computes: X ^{Y} (X to the Y power)

Input: X, Y. If X is less than zero, Y must be an odd integer

Output: P

Also uses: E, L, A, B, C internally. Value of X is changed.

Other subroutines required: Log and Exponential

How to call: 3∅12∅

```
30000 END
30100 REM *EXPONENTIATION* INPUT X,Y; OUTPUT P
30110 REM ALSO USES E,L,A,B,C INTERNALLY
30120 P=1 : E=0 : IF Y = 0 T. RET.
30130 IF (X<0)*(INT(Y)=Y) T. P=1-2*Y+4*INT(Y/2) : X=-X
30140 IF X<>0 T. GOS. 30190 : X=Y*L : GOS. 30250
30150 P=P*E : RET.
```

Logarithms (Natural and Common)

Computes: LOG(X) base e, and LOG(X) base 10

Input: X greater than or equal to zero

Output: L is natural log (base e), X is common log (base 10)

Also uses: A,B,C interally. Value of X is changed.

Other subroutines required: None

How to call: GOSUB 3∅19∅

```
30000 END
30170 REM *NATURAL & COMMON LOG* INPUT X, OUTPUT L,X
30175 REM OUTPUT L IS NATURAL LOG, OUTPUT X IS COMMON LOG
30180 REM ALSO USES A,B,C INTERNALLY
30190 E=0 : IF X<0 T. P. "LOG UNDEFINED AT" ;X:STOP
30195 A=1 : B=2 : C=.5
30200 IF X>=A T. X=C*X : E=E+A : G. 30200
30205 IF X<C T. X=B*X : E=E-A : G. 30205
30210 X=(X-.707107)/(X+.707107) : L=X*X
30215 L=(((.598979*L+.961471)*L+2.88539)*X+E-.5)*.693147
30220 IF ABS(L)<1E-6 T. L=0
30225 X=L*.4342945 : RET.
```

Exponential

Computes: EXP (X) (e to the X power)

Input: X

Output: E

Also uses: L,A internally. Value of X is changed.

Other subroutines required: None

How to call: GOSUB 3∅25∅

```
30000 END
30240 REM *EXPONENTIAL* INPUT X, OUTPUT E
30245 REM ALSO USES L,A INTERNALLY
30250 L=INT(1.4427*X)+1 : IF L<127 T. 30265
30255 IF X>0 T. P. "OVERFLOW " : STOP
30260 E=0 : RET.
30265 E=.693147*L-X : A=1.32988E-3-1.41316E-4*E
30275 E=(((A-.166665)*E+.5)*E-1)*E+1 : A=2
30280 IF L<=0 T. A=.5 : L=-L : IF L=0 T. RET.
30285 F. X=1 TO L : E=A*E : N. X : RET.
```

Tangent

Computes: TAN(X)

Input: X in degrees

Output: Y

Also Uses: A,C,W and Z internally. Value of X is changed.

Other subroutines required: Cosine, Sine

How to call: GOSUB 30320

```
30000 END
30300 REM *TANGENT* INPUT X IN DEGREES, OUTPUT Y
30310 REM ALSO USES A,C,W,Z INTERNALLY
30320 A=X : GOS. 30360
30330 IF ABS (Y)<1E-5 T. P. "TANGENT UNDEFINED" : STOP
30340 C=Y : X=A : GOS.30376 : Y=Y/C : RET.
```

Cosine

Computes: COS(X)

Input: X in degrees

Output: Y

Also uses: W and Z internally. Value of X is changed.

Other subroutines required: Sine

How to call: GOSUB 30360

```
30000 END
30350 REM *COSINE* INPUT X IN DEGREES, OUTPUT Y
30351 REM ALSO USES W,Z INTERNALLY
30360 W=ABS(X)/X:X=X+90:GOS.30376:IF(Z=-1)*(W=1)T.Y=-Y
30365 RET.
```

Sine

Computes: SIN(X)

Input: X degrees

Output: Y

Also uses: Z internally. Value of X is changed.

Other subroutines required: None

How to Call: GOSUB 30376

```
30000 END
30370 REM *SIN* INPUT X IN DEGREES, OUTPUT Y
30371 REM ALSO USES Z INTERNALLY
30376 Z=ABS(X)/X:X=Z*X
30380 IF X>360 T. X=X/360 : X=(X-INT(X))*360
30390 IFX>90T.X=X/90:Y=INT(X):X=(X-Y)*90:ONYG.30410,30420,30430
30400 X=X/57.29578 : IF ABS(X)<2.48616E-4 Y=0:RET.
30405 G.30440
30410 X=90-X : G. 30400
30420 X=-X : G. 30400
30430 X=X-90 : G. 30400
30440 Y=X-X*X*X/6+X*X*X*X*X/120-X*X*X*X*X*X*X/5040
30450 Y=Y+X*X*X*X*X*X*X*X*X/362880 : IF Z=-1T.Y=-Y
30455 RET.
```

ArcCosine

Computes: Arccos(S), angle whose cosine is S

Input: S, Ø<= S<= 1

Output: Y in degrees, W is in radians

Also uses: X,Z internally

Other subroutines required: ArcSine

How to call: GOSUB 3Ø5ØØ

```
30000 END
3Ø5ØØ REM *ARCCOS* INPUT S, OUTPUT Y,W
3Ø51Ø REM Y IS IN DEGREES, W IS IN RADIANS
3Ø52Ø GOS. 3Ø55Ø : Y=9Ø-Y : W = 1.57Ø796-W : RET.
```

ArcSine

Computes: ArcSin(S), angle whose sine is S

Input: S, Ø<= S<= 1

Output: Y in degrees, W in radians

Also uses: X,Y internally

Other subroutines required: None

How to call: 3Ø55Ø

```
3ØØØØ END
3Ø53Ø REM *ARCSIN SUBROUTINE* INPUT S, OUTPUT Y,W
3Ø535 REM Y IS IN DEGREES, W IS IN RADIANS
3Ø54Ø REM ALSO USES VARIABLES X,Z INTERNALLY
3Ø55Ø X=S : IF ABS(S)<=.7Ø71Ø7 T. 3Ø61Ø
3Ø56Ø X=1-S*S : IF X<Ø T. P. S; "IS OUT OF RANGE" : STOP
3Ø57Ø W=X/2 : Z=Ø
3Ø58Ø Y=(X/W-W)/2 : IF (Y=Ø)+(Y=Z) T. X=W : G. 3Ø61Ø
3Ø6ØØ W=W+Y : Z=Y G. 3Ø58Ø
3Ø61Ø Y=X+X*X*X/6+X*X*X*X*X*.Ø75+X*X*X*X*X*X*X*4.464286E-2
3Ø62Ø W=Y+X*X*X*X*X*X*X*X*X*3.Ø38194E-2
3Ø625 IF ABS(S)>.7Ø71Ø7 T. W=1.57Ø796-W
3Ø63Ø Y=W*57.29578 : RET.
```

ArcTangent

Computes: ATN(X), angle whose tangent is X

Input: X

Output: C in degrees, A in radians

Also uses: B,T internally. Value of X is changed.

Other subroutines required: Sign

How to call: GOSUB 3Ø69Ø

```
3ØØØØ END
3Ø66Ø REM *ARCTANGENT* INPUT X, OUTPUT C,A
3Ø67Ø REM C IS IN DEGREES. A IS IN RADIANS
3Ø68Ø REM ALSO USES B,T INTERNALLY
3Ø69Ø GOS. 3Ø81Ø : X=ABS(X) : C=Ø
3Ø7ØØ IF X>1 T. C=1 : X=1/X
3Ø71Ø A=X*X
3Ø72Ø B=((2.86623E-3*A-1.61657E-2)*A+4.29Ø96E-2)*A
3Ø73Ø B=((((B-7.5289E-2)*A+.1Ø6563)*A-.142Ø89)*A+.199936)*A
3Ø74Ø A=((B-.333332)*A+1)*X
3Ø75Ø IF C=1 T. A=1.57Ø796-A
3Ø76Ø A=T*A : C=A*57.29578 : RET.
```

Sign

Computes: SGN(X), the sign-component of X

Input: X

Output: To equal to −1 for X negative, ∅ for X zero, +1 for X positive

Also uses: No other variables

Other subroutines required: None

How to call: GOSUB 3∅81∅

```
3∅∅∅∅ END
3∅8∅∅ REM *SIGN* INPUT X, OUTPUT T=-1,∅ OR +1
3∅81∅ IF X<∅ T. T=-1
3∅82∅ IF X=∅ T. T=∅
3∅83∅ IF X>∅ T. T=1
3∅84∅ RET.
```

Appendix B: Cassette Data Files

The material in this Appendix is optional and yet very important. The more practical programming you do, the more you'll appreciate your TRS-80's data file capabilities. They allow you to go from the world of programming to the larger world of **data processing**.

What we mean is, you'll be able to do lots more with larger quantities of information.

Up to now we've relied on LEVEL I's 26 number variables A to Z, 876 (or less) array locations A(X), two string variables A$ and B$, and DATA lines to store the data our programs need. This leaves us with two limitations:

1. The Computer's memory may not be large enough to hold all the data we need (for example, an inventory list).
2. When we turn off the Computer, the values of A,B,A(X), etc., are lost.

Cassette data files solve both of these problems. We can save huge quantities of information on tape and retrieve them later, just as we save and reload programs. Only instead of the commands CSAVE and CLOAD, we use the special statements PRINT # and INPUT #.

To perform the exercises in this Appendix, you'll need to keep your Tape Recorder connected and set in the proper mode — RECORD, PLAY or STOP — as indicated in the text. Insert a blank cassette tape and set the tape counter to zero so you'll know where you started the data file.

Press RECORD and PLAY keys on your Recorder at the same time, then type in the following lines and RUN:

```
50 A=1:B=2:C=3

100 PRINT # A;",";B;",";C
```

Note the special punctuation required to separate each variable to be printed onto tape. **The sequence of five characters (;",";) must be inserted between every two variables in a PRINT # statement.**

This program causes three things to happen:

1. The Tape Recorder is automatically started (assuming you have it set in the RECORD mode).
2. The values of A, B and C are written onto the cassette.
3. The Recorder is automatically stopped. (You should then press STOP on the Recorder to disengage the recording head.)

You now have a permanent record which can easily be read back into the Computer. Note that the variables A, B and C are **not** written onto the tape — just the **values** of those variables (in this case, 1, 2 and 3) are stored.

To read back the data from tape, you must first press REWIND on the Recorder to rewind the tape to the point where the data file started. (You'll have to disconnect the REMote plug to gain manual control of the recorder. When you have rewound the tape to the starting point, reconnect the REMote plug.)

Type `NEW` to clear out the old program and enter these lines:

```
100 A=0: B=0: C=0

110 INPUT # A,B,C

120 PRINT "THE DATA HAS BEEN READ FROM THE TAPE."

130 PRINT "A=";A,"B=";B,"C=";C
```

Now press PLAY on the Recorder and type `RUN`.

If the data from the earlier program was stored and read properly, the Computer should display:

```
THE DATA HAS BEEN READ FROM THE TAPE

A= 1          B= 2          C= 3

READY

>-
```

Line 100 sets our variables to zero. If the data is not read properly, A, B and C will be output as zero.

Line 110 causes the Recorder to start, loading three numbers into the variables A, B and C. When the three numbers have been read, the Recorder motion is stopped.

No unusual punctuation is required to separate the variables on an INPUT # statement — just the ordinary commas.

Line 120 prints a reassuring message. This is important when the Computer is using an external device such as a Tape Recorder. Print messages are also valuable as prompting instructions to the user regarding the control of the Recorder. For example, before the Computer executes a PRINT # statement, we can have it print a message telling the user to put the Recorder in the Record mode.

Line 130 prints the data that was read from the tape.

NOTE: If the Recorder is not in the PLAY mode (with proper connections made) when it executes an INPUT # statement, the Computer will keep trying to read the tape until it gets something. You have no keyboard control of the Computer during such an input operation, so it is effectively locked-up. The only way to unlock it is to press the Reset button located in the expansion port on the left rear corner of the Keyboard. This will terminate the entire program, but will not erase it.

One last word of advice: If you PRINT # a list of, say, 1Ø values onto tape, you should INPUT # a list of 1Ø values also. If you don't match up the number of PRINT # items with the number of INPUT # items, you'll end up either losing data or going into the lock-up condition described above.

The following program demonstrates how a data file can be used to create a list of data items, process and update it. Study it carefully and think how similar programs might handle inventories, or any sequential lists.

```
  1 REM *AVG.TEMP AND HUMIDITY USING A DATA FILE*
  5 C=Ø:CLS
  7 B=Ø
 1Ø IN. "WHAT DAY OF THE MONTH IS IT";D
 2Ø IN."WHAT WAS THE TEMPERATURE TODAY";T
 3Ø IN. "WHAT WAS THE HUMIDITY";H
 4Ø IF D=1 THEN 16Ø
 5Ø P."LOAD PREVIOUS TEMPERATURES AND HUMIDITIES THIS MONTH."
 53 P."FIRST REWIND TAPE TO BEGINNING OF DATA FILE."
 55 P."THEN PRESS RECORDER'S PLAY KEY."
 6Ø IN."PRESS ENTER WHEN READY";A$
 7Ø FOR X=1 TO D-1
 8Ø INPUT # Y,Z
 9Ø B=B+Y
1ØØ C=C+Z
11Ø NEXT X
12Ø B=(B+T)/D
13Ø C=(C+H)/D
14Ø CLS:P. "THE AVERAGE TEMPERATURE IS";B
15Ø P."THE AVERAGE HUMIDITY IS";C
16Ø P.:P."NOW THE TRS-8Ø WILL WRITE"
17Ø P."TODAY'S TEMPERATURE AND HUMIDITY"
18Ø P."ONTO THE TAPE."
19Ø P."SO PRESS RECORD AND PLAY KEYS"
2ØØ P."BUT DO NOT REWIND."
21Ø IN."PRESS ENTER WHEN READY";A$
22Ø PRINT # T;",";H
23Ø P.:P."NOW TODAY'S INFO IS ADDED TO THE TAPE FILE."
24Ø P.:P."PLEASE PRESS STOP KEY ON THE RECORDER."
25Ø END
```

Line 7Ø reads back all the previous days' numbers, two at a time. When all the information is read in, the average temperature and humidity are calculated (using the current day's info as well).
Line 21Ø then writes the current day's information at the end of the list.

For a sample run of the program, assume it is the first day of the month. Enter plausible temperature and humidity figures. Continue running the program until you've got a cumulative listing for several days. Getting the feel for data files?

Suggestions for Further Use of Data Files

1. **Teaching/Testing**. Write a program that gives a multiple-choice test, for example, a vocabulary test. Include ten questions. The program should write the student's name and all ten responses onto a cassette data file. Design the program so that any number of students may take the test in sequence. Include instructions about when to use the RECORD, PLAY and STOP keys.

Write a grader program that uses the data file created above to read each student's name and responses, grade the test, and then read the next student's test. Be sure to leave time for the teacher to mark down the names and grades in his or her little black book.

2. **Inventory**. Write a program that sets up an array in which you store the following information about a group of cars;

(See Chapter 21, where this same array was developed.)

License No.	Engine Size	Color Code	Body Style

The program should then store the array in a data file.

Write another program which
1. Asks you which car you're interested in (you enter the license number).
2. Reads the data file until it comes to the correct license number.
3. Prints out all the information about that particular car.

Appendix C:
Combined Function and ROM Test

The following program puts the TRS-80 through its paces — all of them. If you're having trouble running a program, and you think it may be the Computer's fault, try this program on it. (First check to see that the Computer powered-up properly by running the `P.M.` test described in Chapter 26.)

Program execution is in three stages:

1. Function checkout (takes about 5 seconds)
2. RAM checkout (takes a few minutes)
3. Display checkout. This lets you check centering, straight-line distortion, etc. (Takes hardly any time at all — press **ENTER** to "redraw" test pattern.)

If at any point the Computer comes back with a "BREAK AT ###" (### will be a line number), you know that one of the functions isn't performing properly (ROM error). In case of a RAM error, BREAK message will be preceded by the message "RAM ERROR".

If you don't get a BREAK message (or an infinite loop), you can relax about the TRS-80 and go back to troubleshooting your program.

Type in the program VERY CAREFULLY, get it running properly, then save it on tape for later use.

```
10 IN. "TYPE 1, THEN PRESS ENTER";X

15 CLS:P.AT0; "TRS-80 FUNCTION TEST"

20 READ Y

30 DATA 2

40 RESTORE

50 READ Y

55 F.A=1TO1000:N.A

60 IFX>YSTOP
```

```
70Ø IFX>=YSTOP
80Ø IFY<XSTOP
90Ø IFY<=XSTOP
10ØØ F.X=1TO1ØSTEP2
11Ø GOTO13Ø
12Ø STOP
13Ø GOS.15Ø
14Ø GOTO16Ø
15Ø RETURN
16Ø ONXGOTO18Ø
17Ø STOP
18Ø SET(X,Y)
185 IFPOINT(X,Y)G.19Ø
187 STOP
19Ø RESET(X,Y)
2ØØ IFX<>Y-1STOP
21Ø IFY=X+1G.23Ø
22Ø STOP
23Ø Z=RND(Ø)
24Ø X=1.1:X=INT(X)
245 Y=ABS(X)/2+.5
25Ø IFY=1G.27Ø
26Ø STOP
27Ø REM EVERYTHING IS OK
```

```
290 CLS:P.TAB(5),"ALL  FUNCTIONS ARE O.K.,THE RAM TEXT IS NOW
    RUNNING."

300 A=M./4-1:B=0

310 F.Y=1TO8:Q=.5

320 F.B=1TOY:Q=Q*2:N.B

330 F.X=ØTOA:A(X)=Q:N.X

340 F.X=ØTOA:IFA(X)<>QP. "RAM  ERROR" :STOP

350 N.X

360 P.AT68,Q:N.Y:P.ATØ; "THE  RAM TEST IS COMPLETE"

370 F.A=1TO25ØØ:N.A

400 CLS:K=1

410 A$=GH

420 F.X=1TO32:P.A$;:N.X

430 F.X=1TO14:P.T.(29);A$:N.X

440 P.AT 469;

450 F.X=1TO9:P.A$;:N.X:P.

460 P.T.(21);:F.X=1TO9:P.A$;:N.X

470 P.AT96Ø;

480 F.X=1TO31:P.A$;:N.X

490 IN.B$

500 IFK>ØA$=8Ø

510 IF K<ØA$=GH

520 K=-K

530 CLS:G.42Ø
```

Interface Specifications

Cassette

Suggested Input Level for Playback from Recorder	2 V peak-to-peak at a minimum impedance of 360K ohms
Typical Computer Output Level to Recorder	800 mV peak-to-peak at 1K ohm
Remote On/Off Switching Capability	0.5 A max at 6 VDC

DIN Jack Pin Connections (See Figure 1)

1 — Remote
2 — Signal ground
3 — Remote
4 — Input from recorder's earphone jack
5 — Output to recorder's Aux or Mic jack

Video Signal

DIN Jack Pin Connections (See Figure 1)

1 — +5 VDC at 50 mA
2 — Not used
3 — Not used
4 — Video signal, 1.4 V peak-to-peak, 0.4V negative sync, 75 ohms
5 — Ground

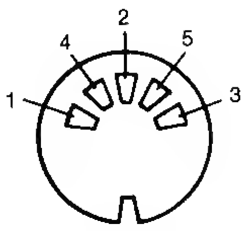


Figure 1. Pin Connections for TAPE and VIDEO DIN Jacks (viewed from rear of keyboard assembly)

Pin Connections for Expansion-Port Edge Card

(See Figure 2)

P/N	SIGNAL NAME	DESCRIPTION
1	RAS*	Row Address Strobe Output for 16-Pin Dynamic Rams
2	SYSRES*	System Reset Output, Low During Power Up Initialize or Reset Depressed
3	CAS*	Column Address Strobe Output for 16-Pin Dynamic Rams
4	A1Ø	Address Output
5	A12	Address Output
6	A13	Address Output
7	A15	Address Output
8	GND	Signal Ground
9	A11	Address Output
10	A14	Address Output
11	A8	Address Output
12	OUT*	Peripheral Write Strobe Output
13	WR*	Memory Write Strobe Output
14	INTAK*	Interrupt Acknowledge Output
15	RD*	Memory Read Strobe Output
16	MUX	Multiplexor Control Output for 16-Pin Dynamic Rams
17	A9	Address Output
18	D4	Bidirectional Data Bus
19	IN*	Peripheral Read Strobe Output
20	D7	Bidirectional Data Bus
21	INT*	Interrupt Input (Maskable)
22	D1	Bidirectional Data Bus
23	TEST*	A Logic "Ø" on TEST* Input Tri-States AØ-A15, DØ-D7, WR*, RD*, IN*, OUT*, RAS*, CAS*, MUX*
24	D6	Bidirectional Data Bus
25	AØ	Address Output
26	D3	Bidirectional Data Bus
27	A1	Address Output
28	D5	Bidirectional Data Bus
29	GND	Signal Ground
30	DØ	Bidirectional Data Bus
31	A4	Address Bus
32	D2	Bidirectional Data Bus
33	WAIT*	Processor Wait Input, to Allow for Slow Memory
34	A3	Address Output
35	A5	Address Output
36	A7	Address Output
37	GND	Signal Ground
38	A6	Address Output
39	+5V	5 Volt Output (Limited Current)
40	A2	Address Output

NOTE: *means Negative (Logical "Ø") True Input or Output

Mates with AMP P/N 88103-1 Card Edge Connector or Equivalent

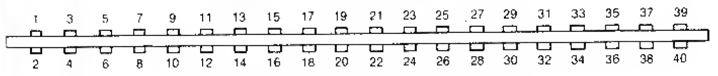


Figure 2. Connection points for Expansion-Port Edge Card (viewed from rear of keyboard assembly)

Notes:

Notes:

Notes:

Summary of LEVEL 1 BASIC

Commands	Purpose	Example	Described in Chapter(s)
NEW	Clears out all program lines stored in memory	NEW (not part of program)	1
RUN	Starts program execution at lowest-numbered line	RUN (not part of program)	1
RUN###	Starts program execution at specified line number	RUN 3ØØ (not part of program)	11
LIST	Displays the first 12 program lines stored in memory, starting at lowest numbered line. Use ↑ key to display higher-numbered lines (if any)	LIST (not part of program)	2
LIST###	Same as LIST, but starts at specified line number	LIST 3ØØ (not part of program)	11
CONT	Continues program execution when BREAK AT ### is displayed	CONT (not part of program)	11

Statements	Purpose	Example	Described in Chapter(s)
PRINT	Prints value of a variable or expression; also prints whatever is inside quotes	1Ø PRINT "A+B="; A+B	1,2,3
INPUT	Tells Computer to let you enter data from the Keyboard	1Ø INPUT A,B,C	7
INPUT	Also has built-in PRINT capability	1Ø INPUT "ENTER A"; A	7
READ	Reads data in DATA statement	1Ø READ A,B,C,A$	16
DATA	Holds data to be read by READ statement	2Ø DATA 1,2,3, "SALLY"	16
RESTORE	Causes next READ statement to start with first item in first DATA line	3Ø RESTORE	16
LET	(Optional) Assigns a new value to variable on left of equals sign	Ø LET A=3.14159	2
GOTO	Transfers program control to designated program line	1Ø GOTO 3ØØ	6

Statements	Purpose	Example	Described in Chapter(s)
IF-THEN	Establishes a test point	1ØIF A=B THEN 3ØØ	6
FOR-NEXT	Sets up a do-loop to be executed a specified number of times	1Ø FOR I=1 TO 1Ø 2Ø NEXT I	10,11 13
STEP	Specifies size of increment to be used in FOR-NEXT loops	1Ø FOR I=Ø TO 1Ø STEP 2	10
STOP	Stops program execution and prints BREAK AT ### message	1Ø IF A<B STOP	11
END	Ends program execution and sets program counter to zero	99 END	2
GOSUB	Transfers program control to subroutine beginning at specified line	1Ø GOSUB 3ØØØ	15,25
RETURN	Ends subroutine execution and returns control to GOSUB line	3Ø1Ø RETURN	15,25
ON	Multi-way branch used with GOTO and GOSUB.	1Ø ON N GOTO 3Ø,4Ø,5Ø 1Ø ON N GOSUB 3ØØØ, 4ØØØ, 5ØØØ	15

Print Modifiers	Purpose	Example	Described in Chapter(s)
AT	(Follows PRINT) Begins printing at specified location on Display	1Ø PRINT AT 65Ø, "HELLO"	22
TAB	(Follows PRINT) Begins printing at specified number of spaces from left margin	1Ø PRINT TAB (1Ø); "MONTH"; TAB (2Ø); "RECEIPTS"	12

Graphic Statements	Purpose	Example	Described in Chapter(s)
SET	Lights up a specified location on Display	1Ø SET (3Ø,4Ø)	20,22
RESET	Turns off a specified graphics location on Display	2Ø RESET (3Ø,4Ø)	20,22
POINT	Checks the specified graphics location: if point is "on", returns a 1; if "off", returns a Ø.	3Ø IF POINT (3Ø,4Ø)=1 THEN PRINT "ON"	22
CLS	Turns off all graphics locations (clears screen)	1Ø CLS	10,20

Built-In Functions	Description	Example	Described in Chapter(s)
MEM	Returns the number of free bytes left in memory	1∅ PRINT MEM	8
INT(X)	Returns the greatest integer which is less than or equal to X	1∅ I=INT (Y)	14
ABS(X)	Absolute value of X	1∅ M=ABS (A)	17
RND (∅)	Returns a random number between ∅ and 1	1∅ X=RND(∅)	19
RND(N)	Returns a random integer between 1 and N	1∅ X=RND(5∅∅)	19

Math Operators	Function	Example	Described in Chapter(s)
+	Addition	A+B	3
–	Subtraction	A–B	3
*	Multiplication	A*B	3
/	Division	A/B	3
=	Assigns value of right-hand side to variable on left-hand side	A=B	3

Relational Operators	Relationship	Example	Described in Chapter(s)
<	Is less than	A<B	6
>	Is greater than	A>B	6
=	Is equal to	A=B	6
<=	Is less than or equal to	A<=B	6
>=	Is greater than or equal to	A>=B	6
	Is not equal to	A<>B	6

Logical Operators	Function	Example	Described in Chapter(s)
*	AND	(A=3)*(A=7) "A equals 3 and A equals 7"	24
+	OR	(A=3)+(B=7) "A equals 3 or B equals 7"	24

Variables	Purpose	Example	Described in Chapter(s)
A through Z	Take on number values	A=3.14159	3
A$ and B$	Take on string values	AS=RADIO SHACK	16
A(X)	Store the elements of a one-dimensional array	A(∅)=4∅∅	21

LEVEL I Shorthand Dialect

Command/Statement	Abbreviation	Command/Statement	Abbreviation
PRINT	P.	TAB (after PRINT)	T.
NEW	N.	INT	I.
RUN	R.	GOSUB	GOS.
LIST	L.	RETURN	RET.
END	E.	READ	REA.
THEN	T.	DATA	D.
GOTO	G.	RESTORE	REST.
INPUT	IN.	ABS	A.
MEM	M.	RND	R.
FOR	F.	SET	S.
NEXT	N.	RESET	R.
STEP (after FOR)	S.	POINT	P.
STOP	ST.	PRINT AT	P.A.
CONT	C.		

RADIO SHACK TC A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U K
280-316 VICTORIA ROAD RYDALMERE, N S W 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

PRINTED IN U.S.A.