TIMEX sinclair

1500 PERSONAL HOME COMPUTER

USER MANUAL

TIMEX sinclair

1500 PERSONAL HOME COMPUTER

USER MANUAL

Charles F. Durang
Author

Judith Richland
Graphic Design

Special Acknowledgements

Charles F. Durang
Author
N. Attleboro, MA

Judith Richland
Graphic Design
Richland Design Associates
Cambridge, MA

We would like to acknowledge the many people who gave us their feedback comments and contributions:

Gregory Coffin, Ph.D. and Staff
Urban Schools Collaborative
Northeastern University
Boston, MA

Nancy Gardner
Cambridge, MA

Larry Johnson
Boston Museum School
Boston, MA

Aaron Paris & Family
Waterbury, CT

Dr. Paul D. Perreault
San Francisco Bay Area Timex Sinclair Computer Users Group
Palo Alto, CA

Serif & Sans, Inc.
Typography
Boston, MA

Joanne Spelman
Thomaston, CT

Jackie Strassberger
Belmont, MA

Susan Thomas
Intentional Education
Watertown, MA

Robert E. Troupe
Minneapolis, MN

Ellen Weinberger
Medford, MA

David Wood
Sinclair/Timex User Group
Boston Computer Society

and especially **Steven Vickers**, from whom we learned much. The Appendices A, B, C, D, and E are his work.

plus the letters from the consumers and the Timex Computer Corporation staff.

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- reorient the receiving antenna
- relocate the computer with respect to the receiver
- move the computer away from the receiver
- plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems". This booklet is available from the US Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

WARNING: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

90-Day Limited Warranty

Congratulations on the Purchase of Your Timex Sinclair Computer!

We hope you'll take the time to read the owner's manual. This will help you to use your Timex Sinclair Computer most effectively and with the greatest of pleasure.

Your new Timex Sinclair Computer, incorporating the latest electronic technology, has been manufactured under stringent quality control standards. Yet, no matter how well designed and constructed, your computer may at some time require service.

To assure that you enjoy the traditional satisfaction of owning a Timex product, Timex computer repair service offers:

- 90-DAY LIMITED WARRANTY
- LOW COST 12-MONTH SERVICE CONTRACT
- FACTORY REPLACEMENT PARTS
- RELIABLE REPAIRS
- PROMPT RETURN OF YOUR COMPUTER

The Timex Computer Club

The Timex Computer Club is an exclusive group of Timex Sinclair Computer Owners. Membership in the Timex Computer Club will allow you to increase your enjoyment of your Timex Sinclair computer. As a member, you will receive regular early notice of Timex Computer Corporation technological advances, new hardware and software products, creative programming ideas and special products and software offers. You will also be able to share computer ideas and achievements with other club members all over the country! **For enrollment see card in back of book.**

NOTE: The 90-Day Limited Warranty on your Timex Sinclair Computer is in no way affected if you choose not to send us the Purchase Information Card. However, we must have the information to enroll you in the Timex Computer Club.

90-Day Limited Warranty

Basic Coverage: This Timex Sinclair Computer is warranted to the owner for a period of 90 days from date of original purchase against defects in manufacture. This Limited Warranty is given by Timex Computer Corporation—not by the dealer from whom it was purchased.

What Timex Will Do: If a defect in manufacture of the Computer is discovered within 90 days from date of original purchase, Timex Computer Corporation will, at its option, repair or replace the defective unit.

What You Must Do: You must return the Computer, with sales receipt, indicating date of purchase, to Timex Product Service Center with a written explanation of the reason for the return. It is recommended that you include both cables, TV/Computer switch, and Power Plug with your shipment.

Return your unit, postage pre-paid to:

Timex Product Service Center
P.O. Box K
7004 Murray Street
Little Rock, AR 72203

To protect against in-transit loss, we recommend you insure your Computer.

Limitations:

THE ABOVE REMEDY IS EXCLUSIVE. TIMEX COMPUTER CORPORATION LIMITS THE DURATION OF ANY WARRANTY IMPLIED BY STATE LAW, INCLUDING THE IMPLIED WARRANTY OF MERCHANTABILITY, TO 90 DAYS FROM THE DATE OF ORIGINAL PURCHASE. TIMEX COMPUTER CORPORATION IS NOT LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGE.
This warranty gives your specific legal rights, and you may also have other rights which vary from state to state. Some states do not allow limitations on how long an implied warranty lasts, or the exclusion of limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you.

This warranty is void if the Computer has been tampered or ill-treated or if the defect is related to servicing not performed by us.

Call 1-800-24-TIMEX from 8:00 AM to 8:30 PM ET* Monday through Friday for answers to computing problems and helpful hints.

*Subject to change.

Table of Contents

Table of Contents

Getting Started

Introduction

Welcome to the world of computing. Before you plug in your new Timex Sinclair 1500, please take a moment to consider these old sayings we just made up:

1. You will enjoy computing.
2. You will find it easy as well as enjoyable.
3. You shouldn't be afraid of the computer. You are smarter than it is.
4. You will make mistakes as you learn. You will learn from your mistakes. The computer will not laugh at you.
5. Your mistakes will not do any harm to the computer. You can't break it by pushing the "wrong" button.
6. You are about to take a giant step into the future. Everyone will soon be using computers in every part of their daily lives, and you will have a head start.

You Don't Need to Learn Programming

You do not need to know how to program a computer to use the T/S 1500, any more than you need to know how to do a tuneup in order to drive a car. You may **want** to learn to program — it is not difficult and can be very enjoyable — but you can use the computer for the rest of your life without having to learn programming.

A Computer Is a Tool

A computer is a tool, like a hammer or saw — or perhaps like a food processor. Hammers and saws can generally do only one thing well. Food processors can perform different operations, and normally you can "program" them by simply pushing the proper buttons.

A computer is an information tool, and is the most versatile tool ever invented. Because it can do many things, it needs very detailed instructions to perform any particular task. These instructions are called programs, and someone has to write them — but not necessarily you.

Many Programs are Available

Once a program is written, it can be saved and used over and over again. Many of these programs are available for you to use on the T/S 1500. You can get some of them on tape cassettes and others in printed form. If a program is short, you may wish to type it in from a printed listing. Longer ones are easiest to load from a cassette tape.

The many available programs for your Timex Sinclair 1500 include games, management tools for your home or business (like balance-sheet calculations, record-keeping, accounting/bookkeeping, taxes, personal or business inventories, etc.), creative and artistic activities.

You can maintain athletic statistics, recipes, address or Christmas-card lists, prepare reports, learn and use mathematics, and do many other

things. Because Timex Sinclair computers are among the world's best-selling computers, new software (ready-to-run programs) is published every day. Your computer will someday be able to answer the telephone, place outgoing calls, bring you information from distant sources, monitor your burglar alarm, water your lawn and perform many other duties around the house.

Use the Computer to Learn

One of the most important uses of the Timex Sinclair computer is as an educational tool. Right now children are beginning to use computers in school, both in terms of learning about them and in using them to learn other subjects. Having a computer at home can help children in school whether or not the schools use computers. More and more educational programming is becoming available.

The Timex Sinclair 1500 is an easy-to-use tool for the beginner, but it is a very sophisticated computer that many scientists and engineers use. As you become more expert in computing, you'll find that you won't easily outgrow your first computer.

This Manual and Other Aids to Help You

This manual will help you get started, and will give you the help you need to use the computer with ready-to-run programs and to do some elementary programming of your own. There is a long and growing list of sources for more advanced information; check your Timex computer dealer, bookstore or library for the latest.

Your dealer may be able to answer some of your questions, and will certainly have some books and other products to help you.

Have fun as you get to know your Timex Sinclair 1500 computer!

How To Set Up Your T/S 1500

1

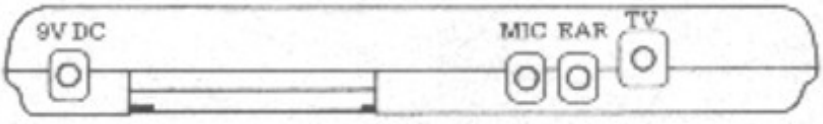
Chapter Preview

This chapter shows you how to connect the T/S 1500 to your TV and cassette recorder, and start using it right away.

Here's what you need to start using your Timex Sinclair 1500 computer, most of which came in the box with this manual:

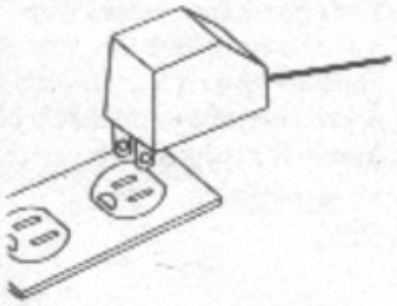
1. **The computer itself.** Only the size of a small book, it packs the computing power of a room-sized machine of just a few years ago.

   You'll see three jack sockets on the back, marked 9V DC, MIC and EAR, and a larger socket marked TV. Also on the back of the computer is a slot, where you can attach "peripheral devices" like the printer, the RAM pack for additional memory, etc.



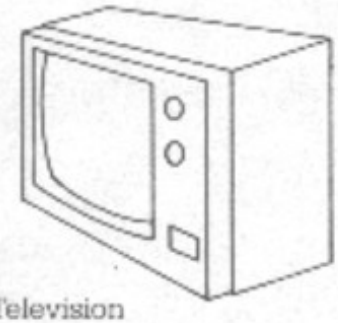
Rear view of computer

Chapter 1: How To Set Up Your T/S 1500



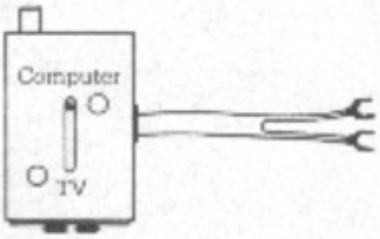
Power supply

2. **The power supply.** It is a transformer that plugs into any standard wall socket (110 volts), with a cord that plugs into the computer's 9V DC socket. The computer has no on/off switch: you turn it on by simply plugging it in with the power supply.



Television

3. **Your television set.** You can use a color or a black-and-white set but, of course, the Timex Sinclair 1500 will display in black and white only.



Transfer switch box

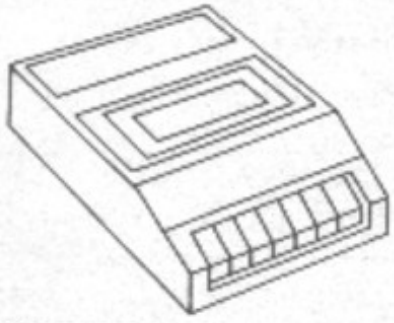
4. **A transfer switch box,** which enables you to switch between your TV antenna and the computer. If you already have a box that looks like this attached to your TV (from a video game console or another computer), you can just leave it there and keep this one as a spare.

TV cable

5. **A television cable,** several feet long, to connect the computer to your TV.

Dual audio cable

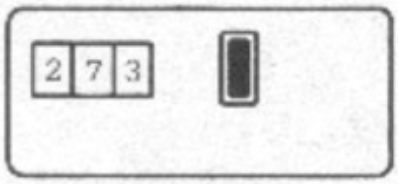
6. **A dual audio cable,** about a foot long, with miniature phone plugs at each end, to connect the T/S 1500 to your tape cassette recorder.



Cassette recorder

7. **Your cassette recorder.** You'll need one that will accept the miniature plugs in its earphone and microphone sockets. (If you already have a recorder and it doesn't fit the plugs, try taking the recorder and the cables to your local electronics shop. You will probably be able to get very inexpensive adaptors rather than buying another cassette recorder.)

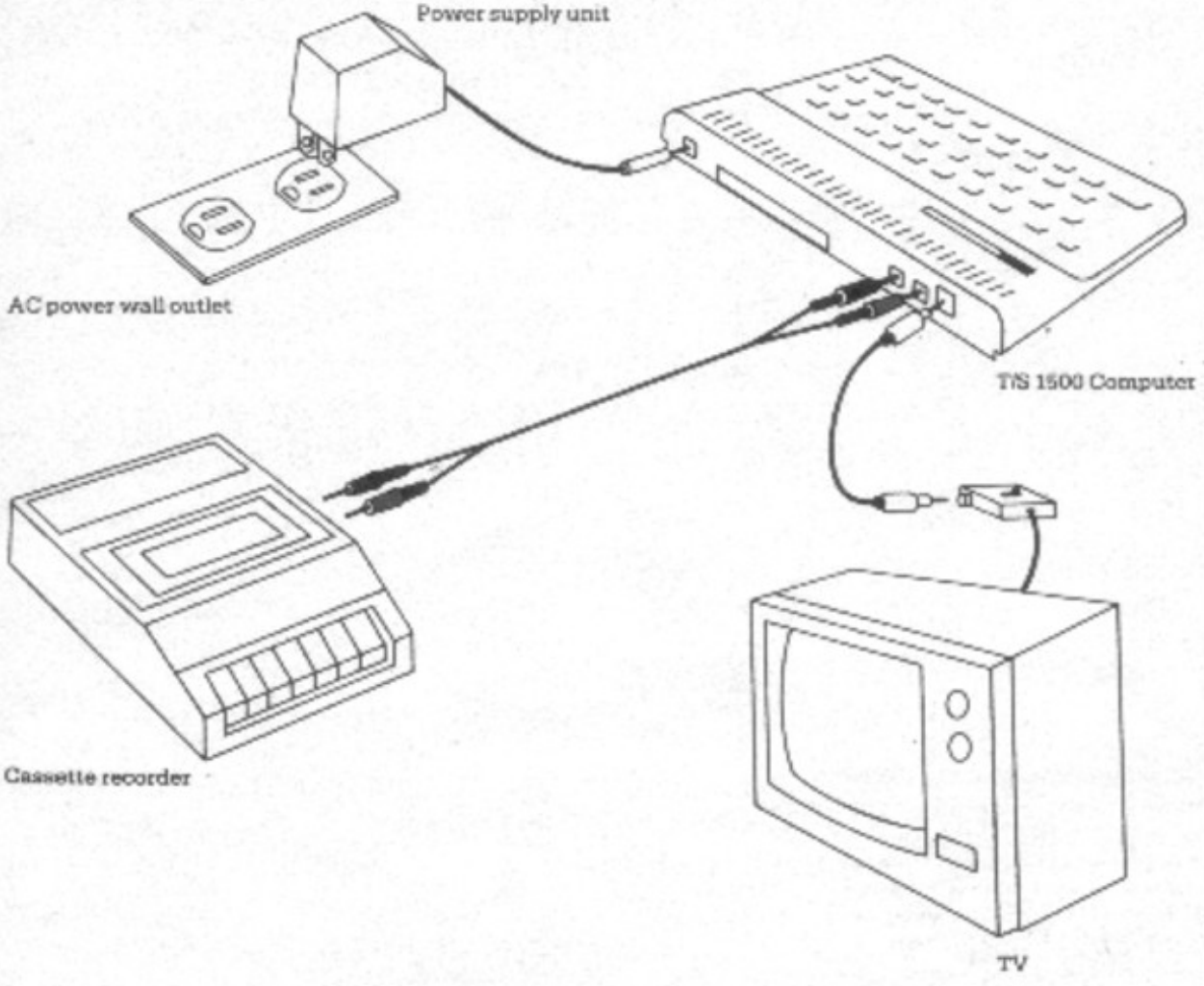
Chapter 1: How To Set Up Your T/S 1500



Tape recorder counter

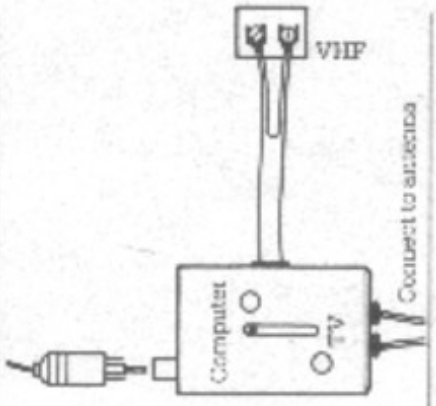
It is not necessary, but can be very helpful, if your recorder has a tape counter. You can then keep track of where on a tape a given program is located, making it easier to find them without having to wait for the computer to search an entire tape.

Now that you have all the parts identified, you can begin to put the system together. (If you are going to try some programming on your own, you can do without the recorder and skip Chapter 3 after setting up; if you have a recorder and want to use some pre-recorded software, or save on tape some of your own programming, you'll want to study that chapter.)



See next page for instructions on setting up the computer.

Chapter 1: How To Set Up Your T/S 1500



Transfer switch box to VHF terminals on TV

First, disconnect the VHF TV antenna wires from your television set (you can leave UHF wires alone).

Connect the wires from the transfer switch box to the VHF terminals on your TV instead, and then connect the antenna wires to the screws marked TV on the switch box.

Plug the connecting cable into the socket on the switch box and into the TV socket on the T/S 1500.

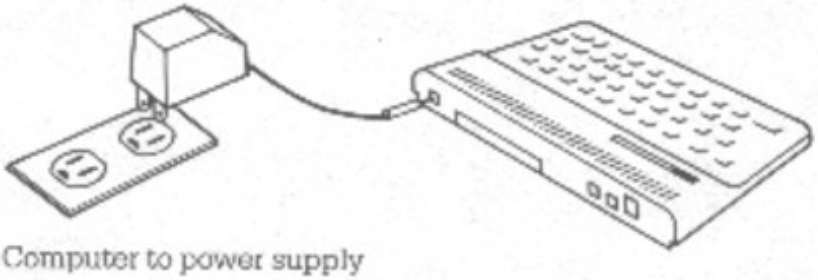
UHF/VHF matching transformer

If you have cable TV, or a 75-ohm antenna lead (the round wire, instead of the flat one), you'll need to buy a "UHF/VHF matching transformer" (or it may be called something else; again, your local electronics store can advise you) for about three dollars. You may even have to contact your cable company if their wire goes directly into your set instead of screwing on to the terminal on the back.

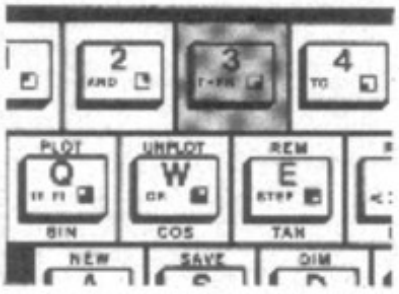
Television channels

Next, turn on the TV. Set it to channel 2 or channel 3, whichever one is not being used for broadcasting in your area.

Then, plug the power supply into the wall and into the 9V DC socket on the computer.

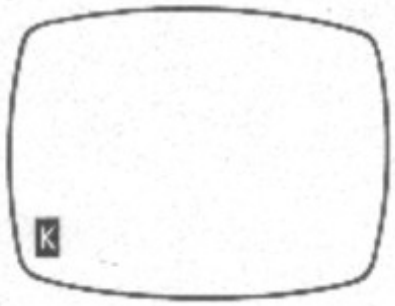


Computer to power supply



If your TV is set to channel 3, press the number 3 on the T/S 1500's keyboard. The Timex Sinclair 1500 will automatically "broadcast" to channel 2 unless "3" is the first key pressed after it is turned on. If the 3 appears on the TV screen (it may or may not, depending on how quickly you press the key after plugging in the computer), press the **ENTER** key to remove it.

Chapter 1: How To Set Up Your T/S 1500



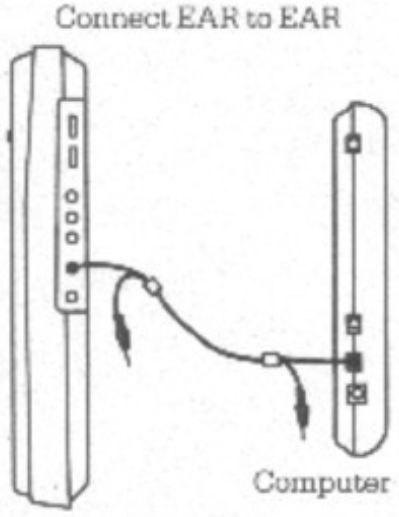
Turn the sound all the way down. You should now have a picture like this on your screen:

The K in the lower left-hand corner of the screen is called the *cursor*, and means the T/S 1500 is ready for action.

Note: The picture on your TV screen should be clear: if you are getting interference, try the following steps in order:

1. Adjust the fine tuning control on the set, then try the brightness, contrast, and horizontal hold (horizontal is usually on the back of the set).
2. Make sure your computer is set to the same channel as your TV. You can unplug the computer and then plug it in again, pressing the 3 key or not as appropriate.
3. Move the computer away from the TV set; or, if possible, place it lower than the set.
4. Plug the computer into a different circuit from the television set; usually outlets on opposite walls are on different branch circuits.
5. Consult an experienced radio/TV repairman; your set may need adjusting.

If you are going to use a cassette recorder to load or to save programs, connect your recorder to the computer with the dual audio cable.



Connect the Earphone socket of the recorder to the EAR socket on the computer in order to load a program from tape into the T/S 1500.

Connect the microphone socket of the recorder to the computer's MIC socket to save programs you've written or changed by adding information.

Notice that the plugs at the ends of the cables are color-coded, so that you can easily make the right connections: Just connect EAR to EAR with the same color plug at each end of the cable.

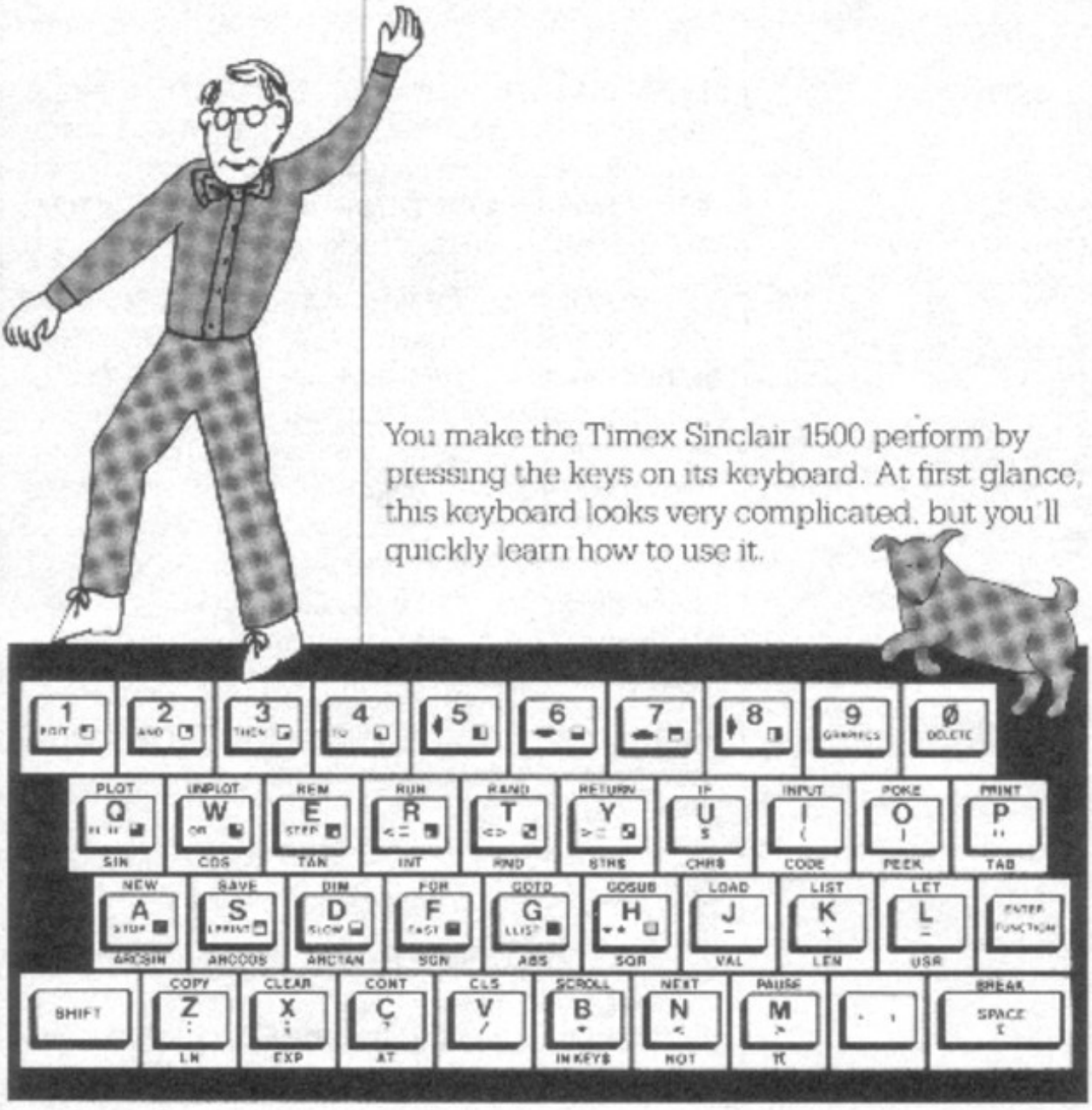
Now you're ready to use your Timex Sinclair 1500!

Moving Around the Keyboard

2

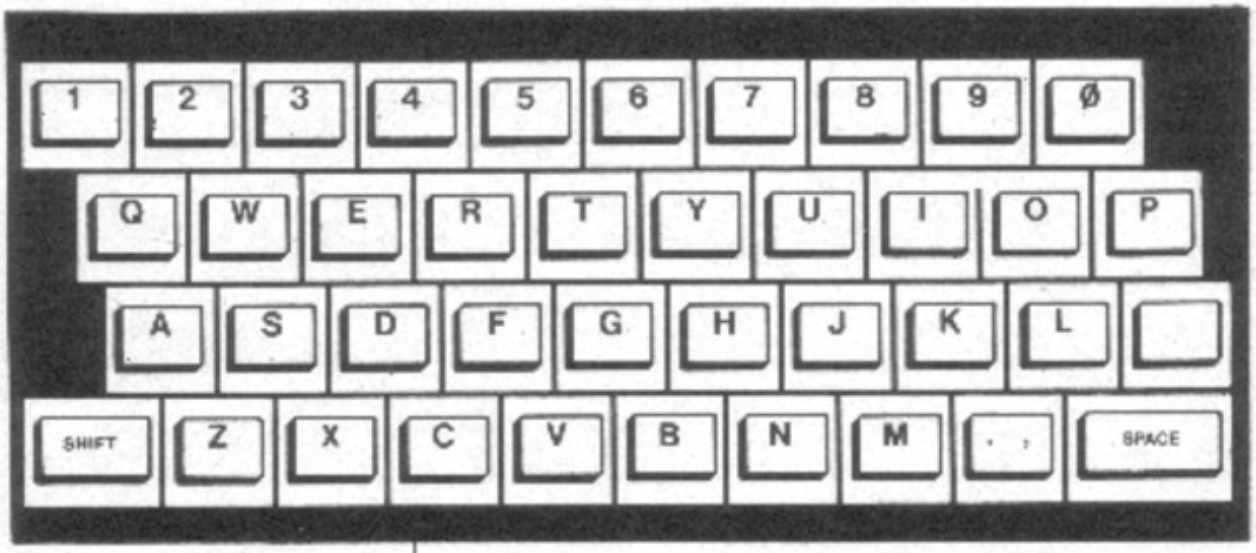
Chapter Preview

How the cursors — K, L, F, G — and SHIFT help you use all the functions on all the keys. We investigate DELETE and the "syntax error" marker, and learn how the left and right arrows work.



You make the Timex Sinclair 1500 perform by pressing the keys on its keyboard. At first glance, this keyboard looks very complicated, but you'll quickly learn how to use it.

Chapter 2: Moving Around the Keyboard



If you know how to type, you'll notice that the largest labels on the keys—the letters and numbers—are arranged just like a typewriter's keys. (If you don't type, you'll find that you won't have much trouble locating the keys anyway.)

The good news—for typists and non-typists alike—is that you don't have to type in the commands to the computer, letter by letter. Instead, you'll find they are indicated as complete words on and around the keys. In many cases, the "keywords" are written above or near the key for the letter that the word begins with: notice, for example, the word **PRINT** above the P key, and **POKE** above the key next to it. Look at the keywords above the G, C, I, and R keys, too.

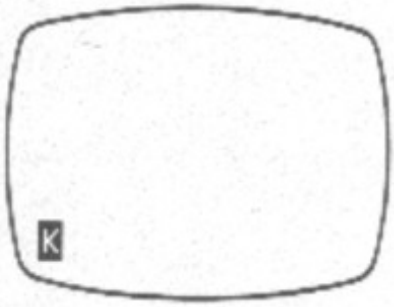
All of the words and symbols on and around each key mean, of course, that each key can do many different things as you give instructions to the computer.

What a key "means" to the computer when you press it depends on two things:

1. Which *cursor* is on the TV screen (remember the K we saw in Chapter 1?), and
2. Whether or not you use the **SHIFT** key while pressing another key.

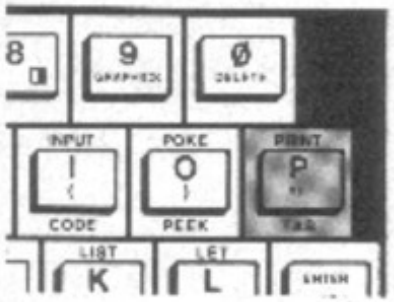
Chapter 2: Moving Around the Keyboard

Let's do a few things with the keyboard to see how to get all the different meanings from the keys. First, we'll ask you to do exactly as we tell you. Then, we'll have a chance to experiment.

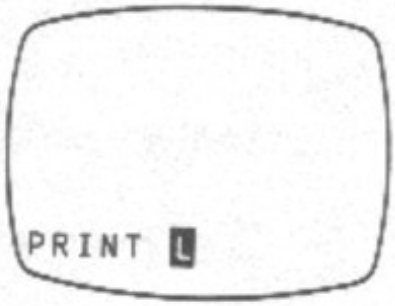


The K Cursor . . . Words Above the Keys

Set up and turn on the computer and television set as we did in the last chapter. You should now have the K cursor in the lower left hand corner of the screen.



Near the upper right hand corner of the keyboard is the P key. We'll call it—and other keys—by the name of the primary (largest) letter or character on it. This is the P key, then. Press it.

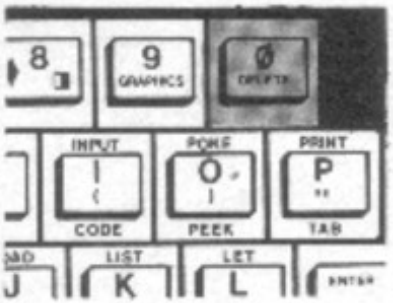


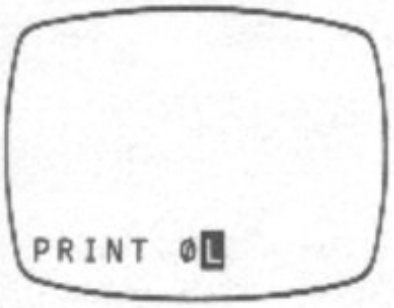
Two things have happened: the word **PRINT** has appeared at the bottom of the screen, and the cursor—which has moved to the right of **PRINT**—has changed from K to L.

K means keyword. Whenever the K is on the screen, pressing a key will always cause the "keyword" above the key to appear on the screen.

The *cursor* will also always change to L. This is because each keyword is a command to the computer, and needs to be followed by some information (or "data") to apply that command to.

Chapter 2: Moving Around the Keyboard



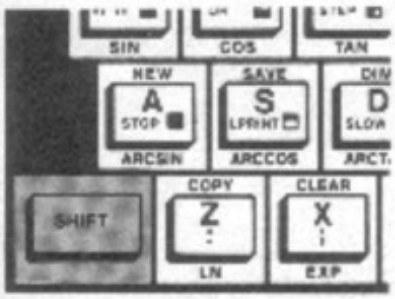


The L Cursor . . . the Main Characters on the Keys

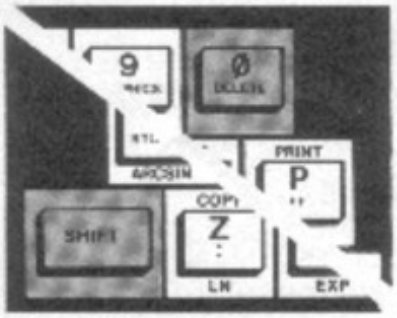
One of the keys you will need most is the DELETE key, just above the P. Let's erase **PRINT**. Press the DELETE key.

Now what? Not only is **PRINT** still on the screen, so is Ø.

Here's why: when the L cursor is on the screen, the T/S 1500 will print the large white letter, number or other item on the key.



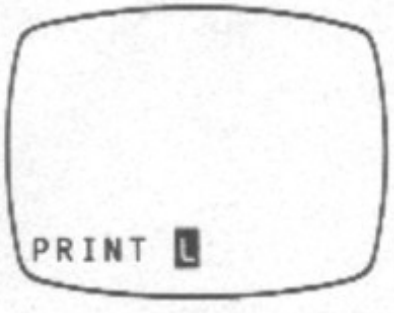
A clue: the word DELETE is in *black* on the key. Look at the **SHIFT** key (in the lower left corner of the keyboard). The word SHIFT is in black, too.



Press SHIFT while you press DELETE

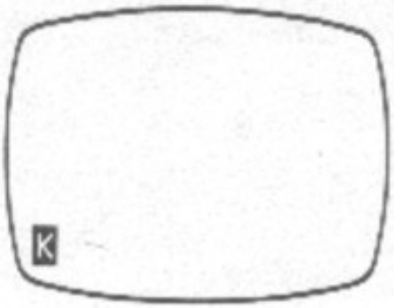
The Shift Key . . . Characters in Black on the Keys

Try pressing SHIFT and (while holding it down) pressing the Ø key.



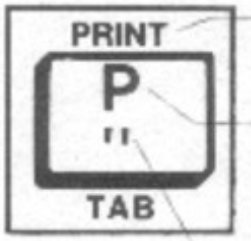
Do it again.

Chapter 2: Moving Around the Keyboard



DELETE—SHIFT 0—Erases the Character or Keyword to the Left of the Cursor.

Now, try typing in and then deleting things—letters, numbers, keywords—one at a time. Try typing in a few things and then deleting them all. Remember:

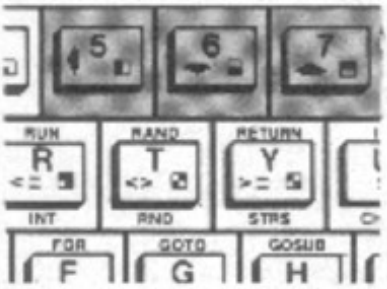


If the **K** *is on the screen, you'll get the keyword above the key.*

If the **L** *is on the screen, you'll get the primary character (the large white one) on the key. This is usually a* **L***etter or numeral.*

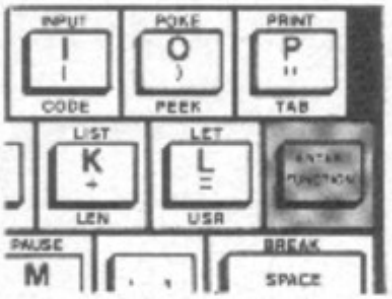
If you press any key while holding **SHIFT**, *you'll get the character on the key in black.*

Does this happen with the **K** cursor on the screen or only with the **L**? Try it and see.



What happens if you press a key with no keyword over it, when **K** is showing—like a number? Or the key marked **ENTER**?

Whatever you have on the line at the bottom of the screen, you can always get back to the beginning by pressing **DELETE** (**SHIFT 0**) as many times as you need to.

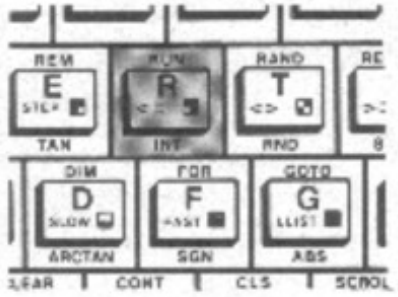


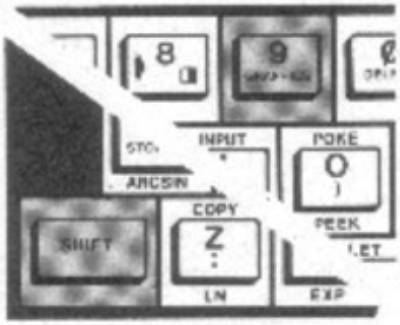
As a last resort, you can also unplug the computer and then plug it back in . . . it always starts "from scratch" when it's first plugged in. (Caution: this is a last resort because you will lose any information in the computer when you unplug it—don't do this if you want to keep anything you've previously typed in.)

Now, you know how to get

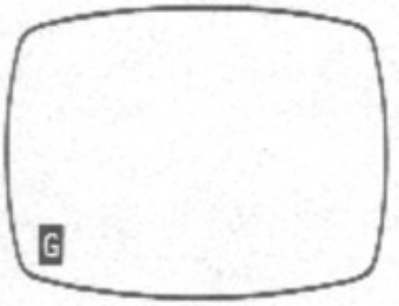
1. Keywords above the keys (**K**),
2. Letters in white on the keys (**L**),
3. Symbols in black on the keys (**SHIFT**).

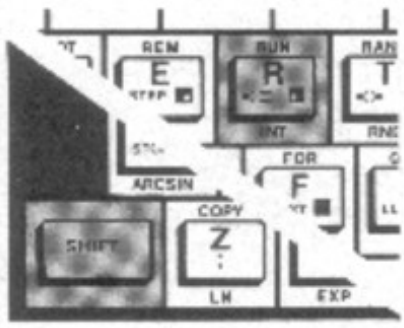
Chapter 2: Moving Around the Keyboard





Press **SHIFT** while you press 9





SHIFT while in Graphics gives you symbol

The G Cursor . . . Inverse Video and Graphic Symbols

On some keys, there's a "graphic symbol" — look at the R key, for instance.

How do we get that?

With yet one more cursor, that's how. Press **SHIFT** 9 ("**GRAPHICS**").

The G cursor says you are in *graphics mode*. Now press the R key.

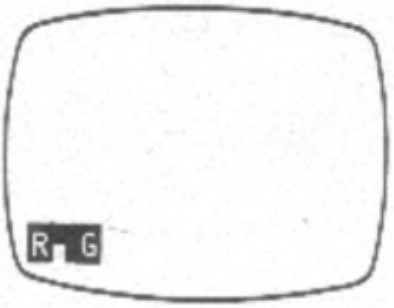
Hmmm . . . still not the symbol, but something else interesting. It is the *inverse* of the main character. Try it with some other keys.

So how do we get ■ and the other graphic symbols? Try holding **SHIFT** while pressing the key.

Okay, **SHIFT** while in graphics mode gives you the symbol. Try a few more.

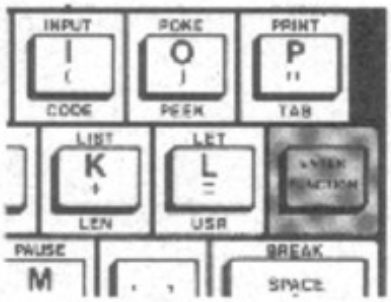
What about keys that don't have a graphic symbol on them? What do you get with **SHIFT** while the G cursor is showing? Try some.

Chapter 2: Moving Around the Keyboard



You can turn off the G mode by pressing SHIFT 9 again. Press it a few times—it turns the G cursor on and off. (You will also find that pressing the ENTER key will "turn off" the G cursor. But don't use this method; it's a bad habit that will cause trouble later.)

Okay, by now we have a long line of graphic symbols and inverse characters. Let's get rid of them with DELETE.

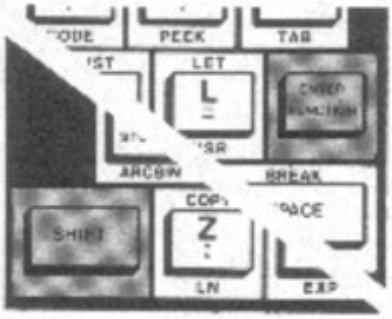


The F Cursor . . . Keywords Under the Keys

How do we get the words *under* the keys?

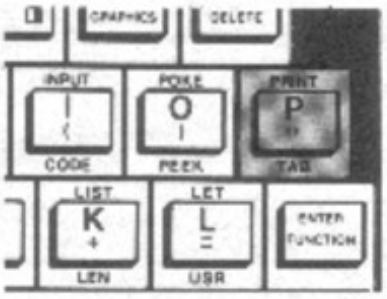
Many of those are mathematical *functions*, so we use the "function mode."

Look at the ENTER key.



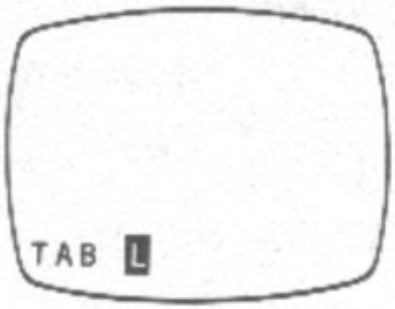
Try holding down SHIFT and pressing ENTER. What happens to the cursor?

It changes to F



Now press the P key.

You should see TAB.



F is just like K (except that you choose when to use the F cursor), but produces the words *under* the keys. The F changes to L after you press a key and obtain a function keyword. (If you just press ENTER, it changes back to K or L, whichever one it came from. Again, don't get in the habit of using ENTER to "turn off" the F mode.)

Chapter 2: Moving Around the Keyboard

Change it back to F (SHIFT ENTER). Press another key—try Q.

Just for fun, try entering a few more, then DELETE them all and get back to a single K cursor on the screen.

The Syntax Error Marker

If, at any time during this chapter—or any other time, for that matter—you see what looks like an S cursor, it means you have tried to enter into the computer an instruction that it cannot handle.

The S is not a cursor at all, but the *syntax error* marker. Syntax has to do with the structure of a language—the way words can be assembled into sentences—and BASIC is a language with very strict syntax rules. The T/S 1500 knows what it can do, and will sometimes say, "I can't do that" with an S marker.

For the purposes of this chapter, don't worry if the S appears; we'll be explaining it later. For now, just dispose of the offending line with DELETE.

The Cursor Arrows

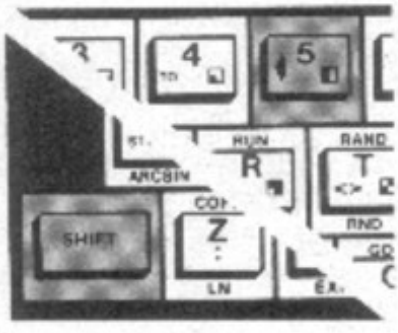
There are a few more keys you'll find very helpful. Try typing in a line like this:

PRINT ABCDEFGGIJKLM

(Yes, put in a second G instead of H. We need a mistake to correct.)

In order to correct the second G, we could DELETE our way back to that spot, erase the G, and then retype the letters, starting with a correct H.

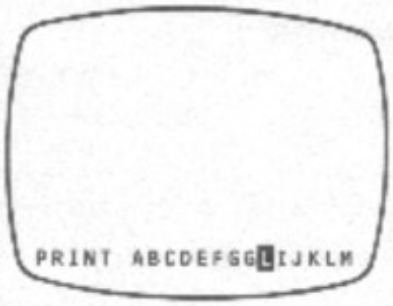
Instead of deleting and then retyping all the letters, try SHIFT 5, the left-pointing arrow.



F, S

Chapter 2: Moving Around the Keyboard

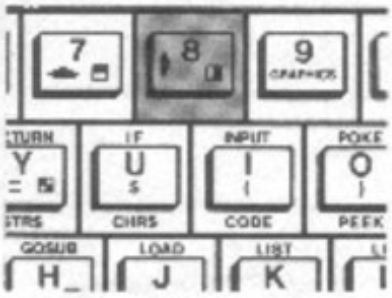
The cursor moves over, in among the letters, without deleting any.



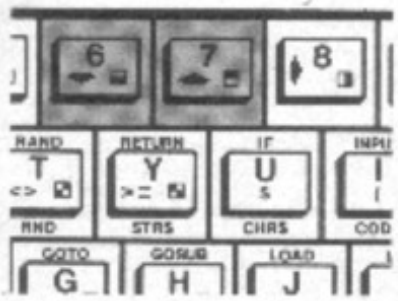
Press it a few more times, until the cursor is just to the right of the second G. Remember, **DELETE** eliminates the character to the left of the cursor.

Now press **SHIFT** Ø (**DELETE**).

And type in the correct letter, H.



You can move the cursor through a line to the right with the right arrow, **SHIFT** 8, as you might expect. Try moving the cursor back and forth.



Try moving it up and down with **SHIFT** 6 and **SHIFT** 7. Nothing happens, because there is only one line to work with. We'll use the up and down arrows in a later chapter.

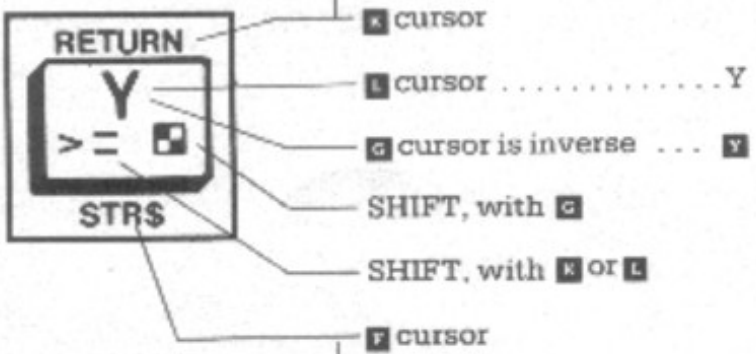
Our tour of the keyboard is now complete. To get ready for the next chapter, take a look at the keywords **LOAD** and **SAVE**. We'll use them next.

In the meantime, feel free to exercise the keyboard some more.

Chapter 2: Moving Around the Keyboard

Summary

1. K cursor: the keyword above the key.
2. L cursor: the main symbol in white on a key, usually a letter or number.
3. SHIFTed key: the black symbol on the key.
4. F cursor: the function under the key. Cursor obtained by SHIFT ENTER.
5. G cursor: inverse of main character on a key. Cursor turned on/off by SHIFT 9.
6. SHIFTed key in G mode: the graphic symbol on the key. If no graphic symbol, the inverse of the black symbol.
7. DELETE (SHIFT 0): erase one entry at a time to the left.
8. A sample key for reference:



Using Ready-To-Run Programs 3

Chapter Preview

How to use prerecorded programs on cassette tape, and store your own programs with LOAD and SAVE. We look at RUN and REM.

As of now, there are many programs available for the Timex Sinclair 1500 computer, programs which are fun and programs which are useful. Timex has a large library of them, and many other sources are also producing T/S 1500 programs. It is certain that there will be many more in the weeks and months to come. You will find that programs written for the Sinclair ZX81, the predecessor of the T/S 1500, will run on your computer, and of course programs for the Timex Sinclair 1000 will as well.

Sometimes you will need to load a program into the computer from a tape. When you are done and turn off the computer, it will disappear from the T/S 1500's internal memory but, of course, you will still have it on tape to load and use again. This is how you will usually use games, for example.

Chapter 3: Using Ready-To-Run Programs

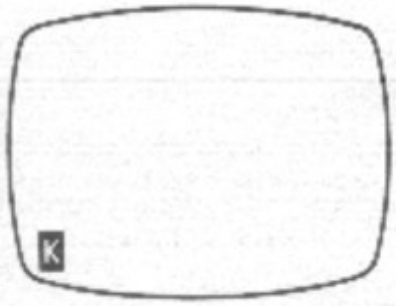
Sometimes you will type in a program from a book in order to use it, and then will want to save it onto a tape cassette. Then, the next time you want to use it, you won't have to type it in, but simply load it from the cassette.

And sometimes there will be programs you will load from a tape, add data to, and then save the program with the data on another part of the tape, separate from the original program.

Let's look at how each of these is done.

Loading a Program From Tape

Important: Before using a tape recorder with your computer, please read the separate sheet titled "Timex Sinclair 1500 Compatible Cassette Recorders" — This sheet contains the latest recorder recommendations.

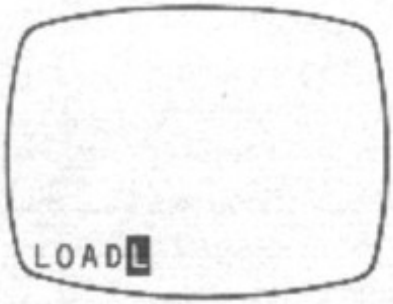


Every program should have a name, and any cassette that has more than one program on it should provide you with an index listing the names of all the programs on the tape.

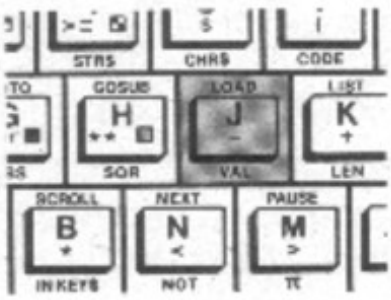
With all the components of your system connected and turned on, as discussed in Chapter 1, make sure that your tape is rewound to the beginning, and that the K cursor is on your TV screen.

Connect the EAR socket on the computer to the earphone or headphones socket on the tape recorder. Turn the volume control on the tape recorder to about three-quarters of the maximum volume; if it has tone controls, adjust them so the treble is high and bass is low.

Then press



LOAD

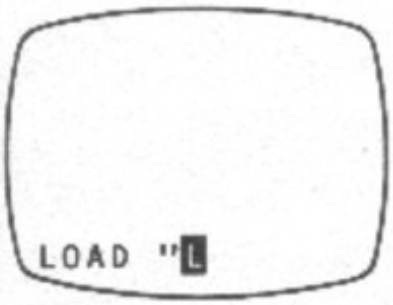


which is what you get when you press the J key while the K cursor is showing (remember, whenever the K cursor is on the screen, pressing any key will give you the keyword command above that key).

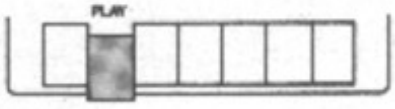
You'll notice that the cursor has changed to L. This means that the computer now will give you the main symbol on any key you press, or—if you hold the shift key down while you press another key—the shifted symbol, which is in black (the same color as **SHIFT**) on the key.

You will want to instruct the computer to load the program you want to use, so you must put the name in quotes. Suppose you want to run a program for a game called STAR ZAP.

Hold down the **SHIFT** key and press the P key, and you'll get quotation marks.

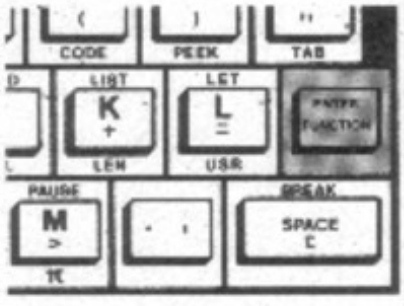


Then, without **SHIFT**, type in the name of the program, making sure you have it exactly right including spaces. Then type **SHIFT** P again for quotation marks. Your screen will look like this:

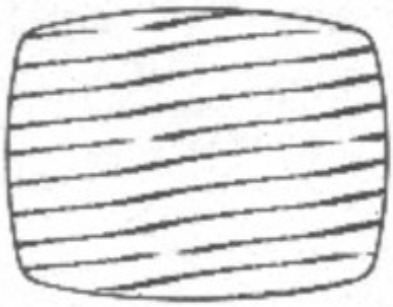


Now press the **PLAY** key on your cassette recorder, and then press

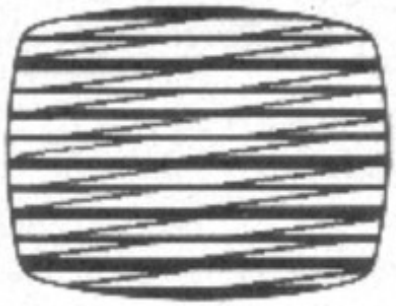
ENTER



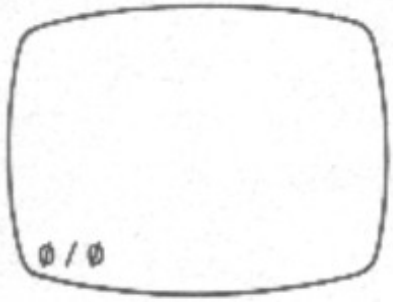
on the computer keyboard. (**LOAD** is a command that tells the computer what you want it to do, "STAR ZAP" tells it what to do it with, and **ENTER** is the signal that the instructions are finished, and the T/S 1500 should start the job.)



Searching



Loading



The TV screen will show you a pattern of thin black lines on the white screen, during the time the computer is searching for the program on the tape.

When the program has been found and is being loaded into the computer's memory, the screen pattern will change to heavier black lines alternating with white.

When the computer has finished loading the program, one of two things will happen:

1. Some commercial programs will begin running automatically, usually with a "title screen" or instructions to the user. (You should stop the tape immediately at this point so as to be in the proper position to load the next program if you wish.)

OR

2. The screen will be blank, except for a **Ø/Ø** in the lower left hand corner. The **Ø/Ø** is a *report code* and means that the computer has successfully loaded the program (more about report codes later). Again, stop the tape immediately. To execute the program, then, you press

   RUN (the R key)

   and

   ENTER

   to start the program.

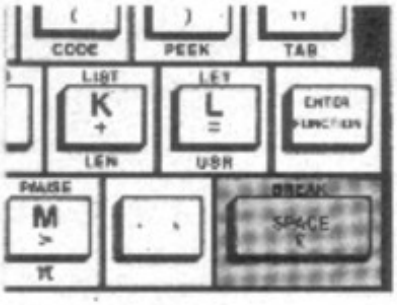
If you are using a cassette with more than one program on it, and wish to load a program that is not the first one, you will see the patterns of thin (searching) and thick (loading) lines more than once. Each program that goes by will cause a "loading" pattern on the screen, even though the computer is not loading it — the T/S 1500 will only start actually loading when it comes to the program you have named.

Occasionally a program will fail to load properly, and you'll have to investigate the reason and/or

try again. You'll know a **LOAD** has misfired if

1. The K cursor comes back on the screen.
2. The "searching" pattern comes back on the screen after the "loading" pattern (if you are certain that loading pattern was for the program you wish to load).

If the K is on the screen, you don't have to do anything except rewind the tape in order to try again.



If the searching and loading patterns are still going on the screen, you need to press the **BREAK** key to stop the process. Then you'll be ready to check into the problem.

The most likely problem is that the volume level is too high or too low. It needs to be (a) loud enough for the computer to pick up the program, (b) not so loud that the program is distorted (this is actually fairly rare), and (c) quiet enough for the silent part to be recognized as silent by the computer.

A way to determine the optimum volume setting for your particular recorder is to load a short (2K) program at various volume settings, noting the lowest and highest settings that produce acceptable loadings, then set your recorder volume in the *center* of that range. If no volume setting can be found that proves successful, you may have other problems.

Some tape recorders can form a feedback loop with the T/S 1500. For this reason, we recommend connecting only one lead at a time; if you've recorded something with both EAR and MIC connections made, you may not be able to load it.

Some tape recorders can record a 60 cycle AC hum. This can be avoided by operating them on batteries.

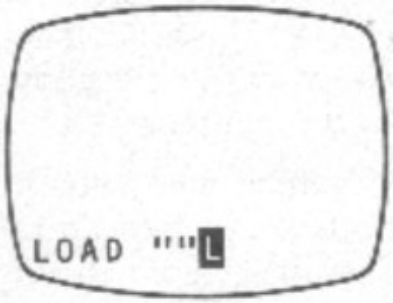
Some tape recorders—especially old, worn ones—are intrinsically noisy, produce a lot of extraneous noise on their tapes. You may have to invest in another recorder.

You may have to wiggle the plug in the earphone socket; on some recorders contact is lost if the plug is pushed in too far. If you pull it out just a bit, you may feel it settling into a more secure position.

It is possible that you have a tape that was recorded (**SAVE**d) on another recorder, and the recording heads on that machine are out of alignment. This is likely if the program can be loaded from that recorder, but not from yours. If you have trouble with a lot of tapes, including commercial ones, your recorder's heads may need adjusting. It is even possible that both recorders are slightly out of adjustment—not enough to keep them from saving and loading their own or commercial tapes, but enough that they cannot use each other's tapes.

It is possible to load a program without using its name. If you type

LOAD ""

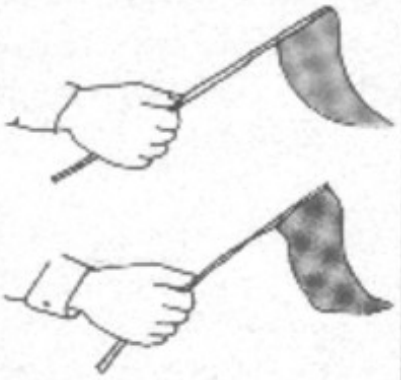


(that is, press the J key and then **SHIFT** P twice; do not put a space between the two quote marks), then start your recorder and press **ENTER**, the T/S 1500 will load the first program it comes to. This is useful if you have only one program on a tape, if you know you want the first one, or if you know you want the next one but have forgotten its name.

Typing in a Printed Program and Saving it on Tape

Many shorter programs are available in books and magazines. You can use them by simply typing them in. Type them exactly as they appear in the publication, making sure your spellings are correct, and all punctuation and spaces as well.

You can check your listing by comparing what you have on the screen with the printed version. See Chapter 2 for how to easily make corrections.



Beware—it is possible for the original listing in the book or magazine to have errors! You may type it correctly and still have trouble.

Note: It is a good idea to put the name of a program into the listing of that program, so you can doublecheck that you have the right one. The easiest way is to add a **REM** line at the beginning. You'll notice that nearly all programs are numbered in multiples of 10, so if a program doesn't already have a line giving its name in the listing, you can just type

5 **REM** PROGRAM: STAR ZAP

using a line number lower than the lowest in the listing (the computer will then automatically put it at the beginning) and, of course, the actual name of your program.

When you've finished typing the program in, you execute it by pressing

RUN and **ENTER**

as above. When you've finished using it—either you reach the end, or you interrupt the program by pressing the key marked **BREAK**—you can get the listing back on the screen by pressing

ENTER

again. Then, after verifying (by using it) that the program works and that you've typed it in correctly, you can save it for future use on tape. (You don't have to try it out first, of course, but then you might be going to the trouble of saving a program that won't run.)

Saving a Program on Tape

As we said earlier, every program should have a name. The T/S 1500, in fact, won't save a program on tape without a name. You can make up a name for a program you invent, use the name of a pro-

gram you have typed in as above, or even change that name to something you like better. Whatever you call the program when you save it will be the name you have to ask for to load it later.

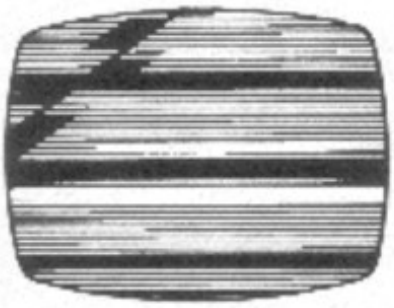
Connect the MIC socket of the computer to the microphone socket of the recorder. Position the tape in a part that is blank, or a part that you are prepared to overwrite. Enter the following:

SAVE "STAR ZAP"

using the keyword **SAVE** over the S key.

Start the tape recorder, on **RECORD**, then press

ENTER



Saving

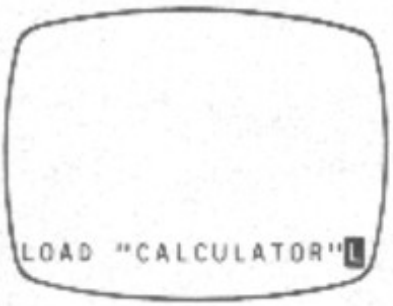
Watch the TV screen. You'll see a pattern of black and white lines—different from either the "searching" or "loading" patterns—and eventually the screen will show **0/0**, which in this case means "the **SAVE** is complete."

As a check on whether the recorder has received the program correctly, you can listen to it over the speaker. Rewind the tape to where you started and play it back. You should hear, in order:

1. A soft, humming buzz. This is the signal before you pressed **ENTER**.
2. Five seconds of silence.
3. A very harsh, high-pitched buzz, which is the program itself. This will go on longer for a longer program.
4. The soft, humming buzz again.

Note: A technique that may make it easier to find programs you record is to speak the name of the program onto the tape through the microphone before connecting the computer to the recorder. Then you can search for the sound of your announcement to find the program.

You can check on your success at saving a program by following the **LOAD** instructions above.



Saving Programs with Your Own Data Entered

Some programs are meant for you to enter your own data into—saving lists, figures, etc. These are easily used by following the same procedures we've just discussed.

1. **LOAD** the program as we've described.
2. **RUN** the program, entering your own data as it is called for.
3. **SAVE** the program with the data in it, using a new name to distinguish it from the original program. If, for example, you load a program called "Calculator" and then fill in your personal financial records, you may want to save the filled-in version under the name "Finances."

As you can see, there are many ways to use your Timex Sinclair 1500 without learning computer programming. But if you'd like to look into it, try the next chapters.

Summary

1. Many programs are available for use with the Timex Sinclair 1500 computer, and you don't have to know how to program to use them.
2. The **LOAD** command followed by the name of a program in quotes causes the computer to load that program you play into it from a tape cassette.
3. **LOAD** followed by two quotation marks with nothing between them causes the T/S 1500 to load the next program on the cassette tape.

4. The **SAVE** command followed by the name of a program in quotes tells the computer to send that program to a cassette recorder running in recording mode, saving the program on the tape.
5. You must have a name for the program when using **SAVE**, but you can make up any name you like, and change the names of alternative versions of the same program (like a program into which you put information that is updated periodically).

Telling the Computer What To Do 4

Chapter Preview

PRINT and ENTER help you start giving orders to the T/S 1500. Learn how to put words and numbers on the screen. We look at the quotation marks, "strings," and how to use functions like SQR.

We saw, in Chapter Two, how the keyboard works. In Chapter Three, we talked about loading and saving programs on tape. Starting with Chapter Five, we'll be learning how to write our own programs.

This chapter is a bridge between what has gone before and what comes after, if you want to go on into programming.

It is also a repetition of some of what is in Chapter Two, to increase your skill on the keyboard — even if you don't want to become a programmer.

As we have said, the computer is a very versatile machine. It can do many things, as long as it is told what to do. But it has to be told in words it can understand, and in small steps it can execute.

The Timex Sinclair 1500 is built to understand orders given to it in BASIC (which stands for

"Beginner's All-Purpose Symbolic Instruction Code"). Invented at Dartmouth College, BASIC looks more like English than other languages and is very easy to use.

The *keywords* above the keys, which are *commands* to the computer, are in English. They are also in BASIC, which means each word always means exactly the same thing.

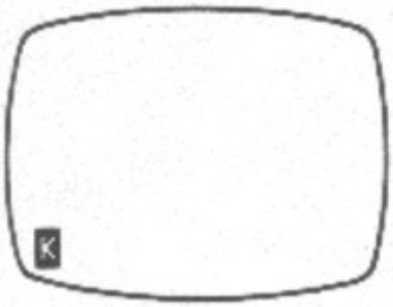
For example, the keyword we will probably use the most is **PRINT**. In English, this can mean

1. Make letters on paper with a writing instrument held in your hand,
2. Transfer letters from a printing press to paper, or
3. Publish.

There are probably several other meanings you could think of, slightly different from these.

In BASIC, **PRINT** means only one thing:

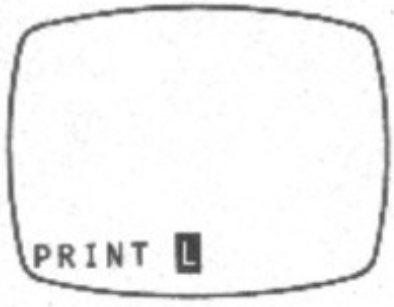
1. Print *on the screen* whatever follows the word **PRINT**.



Let's have some more practice at giving the computer commands and see how it performs.

Start by setting up the computer so that the screen looks like this.

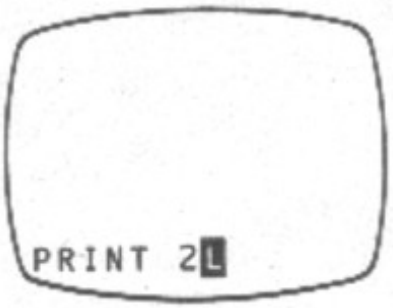
Now we'll have the computer figure out the sum of 2 and 2, and show the result on the screen.



First, type **PRINT**. Press the P key, and the word **PRINT** appears. Remember, although you can spell out the word P,R,I,N,T, you give the T/S 1500 its orders using the keywords above the keys. These appear if you press a key while the K cursor is on the screen.

In case you haven't noticed, we are indicating keywords in this manual by printing them in **BOLD FACE TYPE**.

Besides the word **PRINT** appearing on the screen, you see that the cursor has changed to L.

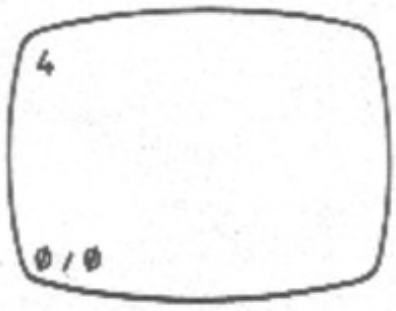


Next, type 2. The 2 appears on the screen, and the L moves to the right. Notice, by the way, that there is a space between the **PRINT** and the 2 on the screen. Timex Sinclair BASIC keywords come with their own spaces, and you don't have to put them in (it won't do any harm if you do put in extra ones).

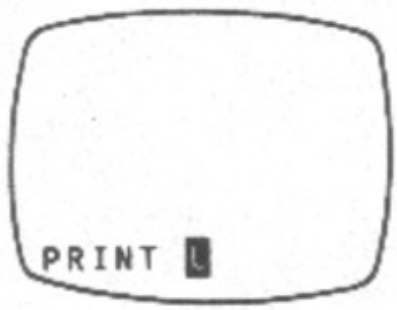
Third, type a plus symbol (+). Remember how we obtained symbols that are in black on the keys—using **SHIFT**. Hold down **SHIFT** and press the K key.

Next, type another 2. The screen will look like this:

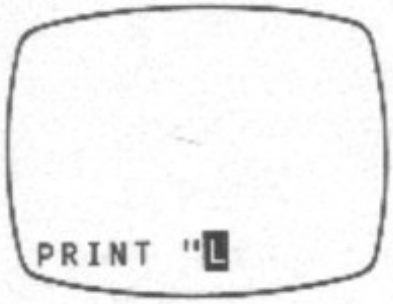
Finally, press **ENTER**. Whenever you are finished with a line or a command, and want the computer to do something, you signal this by pressing **ENTER**.



The computer will compute and display the answer. Of course, you don't need a computer to figure that one out, but you can use it for more difficult math (see Chapter 10).



Let's try something else. The *0/0 report code* is "hiding" a K cursor—if you press a key while a report code is on the screen, you'll get a keyword just as if a K was showing. So press P again, and get **PRINT**.



It's okay that the answer to the previous calculation is still on the screen.

Now, using **SHIFT**, get quotation marks by pressing P again.

And press 2. And + (using **SHIFT** again). And another 2.

Finally, close the quotation (**SHIFT** P). The screen should look like this:

Now press **ENTER**.

Aha. Let us examine why the difference. The first time, when we typed

PRINT 2 + 2

we were telling the computer to evaluate a mathematical expression and print the answer.

The second time, when we typed

PRINT "2 + 2"

we were telling the computer, "Don't do any calculation, just print whatever is in quotes."

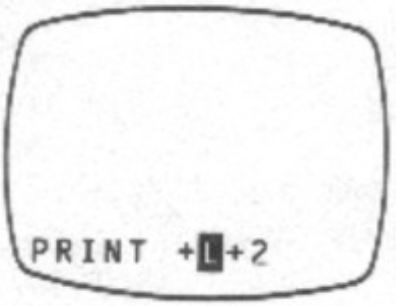
Try this (remember that, inside the quotes, you *will* have to put in spaces where you want them, using the **SPACE** key in the lower right corner of the keyboard).

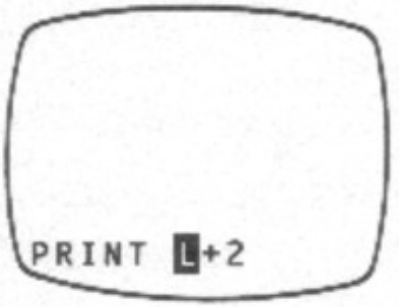
PRINT "ANYTHING THAT APPEARS IN QUOTES NO MATTER HOW LONG IT IS"

Don't forget to press **ENTER** when you are done.

Here's some computer jargon for you: anything we put in quotes is called a *string*. This sounds odd at first, but it essentially means that the entire "string" of characters within the quotes is treated by the computer as a single item.

Chapter 4: Telling the Computer What To Do





What To Do If You Make a Mistake

If you make a mistake, you can erase it with the **DELETE** key (**SHIFT** 0). Pressing **DELETE** erases the character just to the left of the cursor.

And, you can move the cursor to where you want to make a deletion by using the left and right arrow keys (**SHIFT** 5 and **SHIFT** 8).

For instance, type

PRINT + +2

and press **ENTER**.

The screen will look like this:

Remember that the S is the *syntax error* marker. It means that the T/S 1500 cannot execute that command (you can't add 2 to +). Of course, we meant **PRINT** 2 + 2, so let's make the correction, the way we did in Chapter Two.

First, we need to move the L cursor to the scene of the crime. Press the left arrow (**SHIFT** 5, holding down the **SHIFT** key while pressing the 5 key). The cursor moves one place to the left.

Press the left arrow again, which puts the cursor in the proper position.

Now press **DELETE** (**SHIFT** 0) and we have this on the screen.

We can then put in the correct character. Type 2.

Now, we can press **ENTER**. We don't need to move the L cursor back to the end of the line: it doesn't matter where it is, as long as the rest of the line is correct. The cursor is just a place marker, for your eyes only.

Time to practice:

Try typing in **PRINT** " and then any sentence you like. Don't forget to close the quotes at the end.

Practice moving the cursor backwards and forwards through the line with the left and right arrow keys.

Make any corrections you like, before pressing **ENTER**. Besides correcting errors you may notice, try adding new words by moving the cursor to the spot where you want to insert them and then typing the words.

Practice making up a line of graphics, by pressing **PRINT** " and then getting into *graphics mode* with **SHIFT** 9. Remember, you have to use **SHIFT** 9 (GRAPHICS) again to get out of the graphics mode in order to close your quotes and **ENTER** the line.

In case you are mathematically inclined, here's a quick early hint on the use of functions. Press **PRINT**, then enter the *function mode* by pressing **SHIFT ENTER**. With the F cursor on the screen, press the H key, giving you **SQR** (for *square root*). Then press 2 and **ENTER**.

The computer evaluates the mathematical expression and prints the square root of 2 — 1.4142136 — on the screen.

We won't spend much time on the other functions in this manual, except to mention them in Chapter 10 and list them in Appendix A. If you are a mathematician and recognize the abbreviations under the keys, fine. If not, don't worry about them.

Summary

1. **PRINT** (keyword command located above the P key) tells the computer to print something on the screen.
2. If a number follows **PRINT**, the number will be printed on the screen; if a mathematical expression (like 2 + 2) follows **PRINT**, the expression will be evaluated and printed (for example, 4).
3. Anything in quotation marks after **PRINT** will be printed on the screen exactly as it appears. The material in quotes is called a *string* because it is a string of characters.
4. When you press **ENTER**, you signal the computer that you are finished writing your command and would like it carried out.
5. Report Codes are printed by the computer in the lower left hand corner of the screen when it stops executing its instructions. The codes tell you whether it has successfully carried out your orders or whether it has encountered a problem.

Writing a Program 5

Chapter Preview

How to start with NEW, repeat with GOTO, stop with BREAK, and continue with CONT.

We are now ready to write our first computer program.

In the last chapter, we were operating in what is called the *immediate mode*, which means that the computer executed each command immediately.

When we write programs, we give a number of commands and the computer executes them, in order, when we tell it to. Until we tell it to execute —and afterwards, for that matter—the T/S 1500 remembers all the commands.

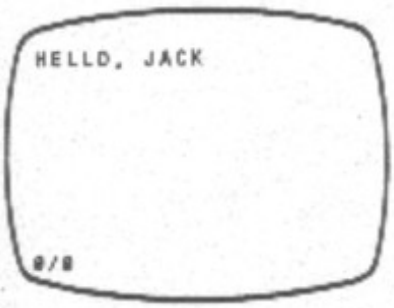
Although we will start small, there are programs of many thousands of lines, which direct computers to carry out lengthy and complicated procedures The power of computing is in the ability of the machine to receive, store and carry out many different complex programs.

Let's get started. If you are just plugging in the computer to start this chapter, you should have the K cursor in the corner of the screen. If you are continuing from the last chapter, press **NEW** (the A key, while the K cursor or a report code is showing). This clears both the screen and the computer's memory, so it is ready to start a **NEW** program. (Don't forget to press **ENTER**.)

Type in

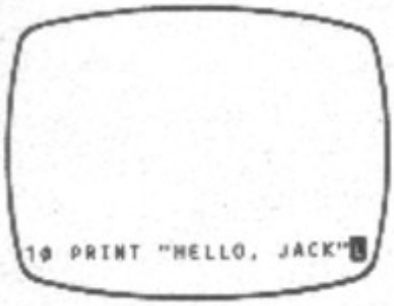
PRINT "HELLO, JACK"

and press **ENTER**. Just as in the last chapter, the computer executes the command immediately. (By the way, you can use your own name instead of Jack if you prefer.)

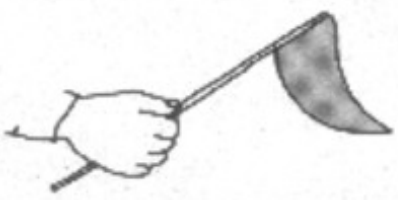


Now type in

10 **PRINT** "HELLO, JACK"



Notice that the K cursor doesn't change as you type the 1, and then the 0. This is why there are no keywords over the number keys: so you can put in program line numbers. It changes to L after the keyword **PRINT**.



If you are a typist, by the way, be careful to use the numeral 1 and not the lower case L for a 1. For one thing, the computer doesn't have lower case letters. And for another, each character, like each keyword, can only mean one thing to the T/S 1500.

Be careful also to use the numeral 0 (zero) and not the letter O. We will show zeros with the slash mark through them to distinguish them from letter O's (this is common in computing).

Now press **ENTER**. Notice that the entire program line has appeared at the top of the screen . . . instead of just the words HELLO, JACK.

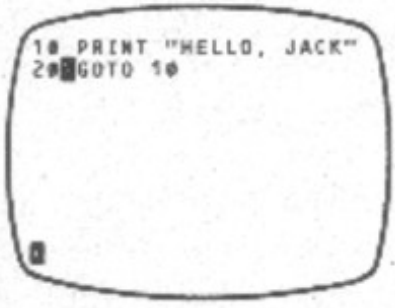
You will also notice a symbol between the 10 and the **PRINT**. This is the *program cursor*. It is placed at the line most recently entered into the program (and we'll use it for something later).

The only difference between the first line we typed and the second was the *line number*. When we put a number in front of a command, it becomes a *program line* and is not immediately executed.

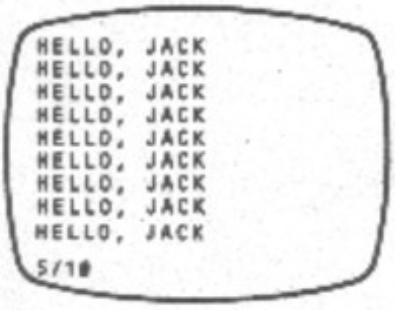
The cursor is ready again at the bottom of the screen. Type this:

20 GOTO 10 ENTER

Note that **GOTO** is a keyword (over the G key) and should not be spelled out (in fact, it appeared as soon as you pressed the G). Notice, also, that the program cursor now shows at Line 20.



Now you have a complete, if brief, program. The command in line 20 simply tells the computer to go back to line 10 and start over. Can you guess what will happen when you execute the program?



Let's try it and see. To execute a program, you simply press

RUN

(the keyword over R), and

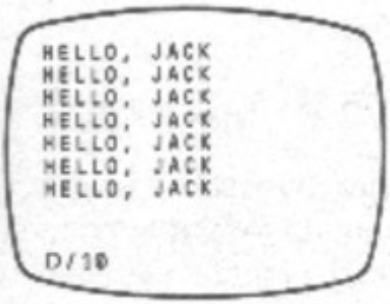
ENTER

How about that? That's a lot of stuff on the screen for such a short program. As we said, the computer is not very smart. But it is fast, accurate, and tireless doing repetitions.

One of the most powerful commands in BASIC is the **GOTO**, which is often used to direct the computer to go back to an earlier program line and repeat a process over and over again.

You'll notice the report code 5/10 at the bottom of the screen. That means it stopped because the screen was full (5—see the Report Codes at the back of the book), and the last line it executed was number 10.

You can clear the screen and allow the program to go on by pressing **CONT** (for **CONT**inue), the keyword over the C key, and **ENTER**.



Here's something else. While the computer is running HELLO, JACKs down the side of the screen, press the **BREAK** key (which doubles as the **SPACE** key when a program is not running) and notice that the list stops wherever it was when you pressed **BREAK**.

The report code for **BREAK** is D and any program line number may appear after the slash.

The computer checks at the end of every program line to see if anyone has pressed **BREAK**; if so, it stops the program. (You can also use **BREAK** to stop a runaway program—like an "endless loop," about which we'll say more later—or a misfired **LOAD** from a tape cassette. If **BREAK** doesn't work, you may have to resort to pulling the plug which, of course, means you lose any information in the computer.)

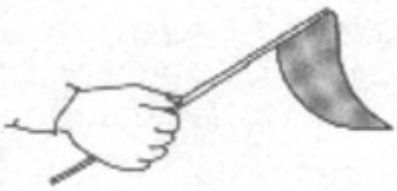
You can restart the program after **BREAK**, also by pressing **CONT** and **ENTER**. **CONT**inue lets you continue when the screen is full, when you interrupt the program with **BREAK** or **STOP**, or when the program itself interrupts with the **STOP** command. (**STOP** is used rather than **BREAK** within

a program, or when the program is waiting for input—yes, we'll be talking about **STOP** later, too—and **BREAK** is used while a program is in full gallop.)

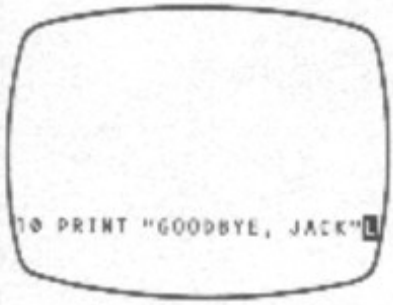
Okay. After you've had your fill of fooling around with **BREAK** and **CONT**, let the program stop with a full screen, and press **ENTER** again.

Your program is back on the screen again. You can **RUN** it again (go ahead, press **RUN** and **ENTER**).

You can also leave it in the memory and go on to something else. It will stay there until you erase it or unplug the machine.

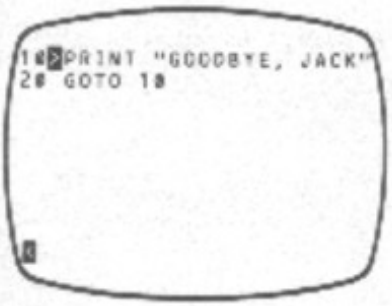


But there is one caution. Let's look into it. Get the program listing back on the screen, by pressing **ENTER** when it stops.



Now, type in

10 **PRINT** "GOODBYE, JACK"



and press **ENTER**.

The new line 10 replaces the old line 10 in the program. (And the program cursor shows that Line 10 was the last one you entered, even though it is not the last one in numerical order in the listing on the screen.)

There can only be one line 10 and any time you enter a new one, you lose the old one. This means that, if you leave a program in the T/S 1500 (instead of restarting with **NEW**), you run the risk of having the new program erase the old one. Or, worse, of having some lines from the old program interwoven with lines from the new one.

```
5 REM PROGRAM: GOODBYE
10 PRINT "GOODBYE, JACK"
20 GOTO 10

Ok
```

Try this:

5 **REM** PROGRAM: GOODBYE

and press **ENTER**

Program line 5 has been inserted into the program where it belongs in numerical order. This is why we usually number the lines in multiples of 10: it gives us room to insert new lines if we find we need them.

```
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
GOODBYE, JACK
5/10
```

Now press **RUN** and **ENTER**.

The program line (5) beginning with the keyword **REM** doesn't "do" anything in the program. Any line beginning with **REM** — for **REM**ark or **REM**inder — appears in the listing to help the user understand the program — but is disregarded by the computer when running the program.

```
 5 REM PROGRAM: GOODBYE
10 PRINT "GOODBYE, JACK"
15 PRINT "SEE YOU LATER"
20 GOTO 10

Ok
```

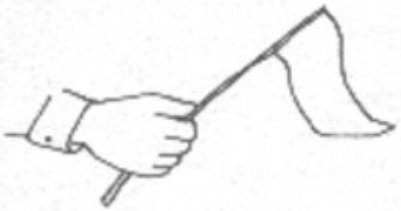
As another example, try typing

15 **PRINT** "SEE YOU LATER"

and **ENTER**

```
SEE YOU LATER
GOODBYE, JACK
SEE YOU LATER
GOODBYE, JACK
SEE YOU LATER
GOODBYE, JACK
SEE YOU LATER
GOODBYE, JACK
SEE YOU LATER
5/10
```

and **RUN** the program. Then press **ENTER** again to show the listing. Can you see why the program does what it does?



Now you might want to try to write a few simple programs of your own, using lines of strings to print or of mathematical calculations. Or even mixing the two (remember, the computer will do *exactly* what you tell it to do, even if the result doesn't make any sense).

Summary

1. **NEW** erases everything that might have been typed into the computer, to make room for a new program.
2. When you put line numbers in front of commands, they become program statements and are not executed immediately. Instead, they are carried out in numerical order in response to the **RUN** command.
3. **GOTO** is a very powerful BASIC command. It directs the computer to a line in the program other than the next one in numerical order, and allows the computer to repeat sequences of program lines over and over.
4. **BREAK** stops a program while it is executing.
5. **CONT**inue restarts a program that has stopped for certain reasons, most often because **BREAK** has been pressed or the screen is full.

Arranging Output on the Screen 6

Chapter Preview

This chapter shows you how to use EDIT, AT, TAB, commas and semi-colons to move things around. We use the up and down arrows.

You can do wonders with punctuation marks in a T/S 1500 BASIC program. For instance, let's start with the program we finished the last chapter with:

```
 5 REM PROGRAM: GOODBYE
10 PRINT "GOODBYE, JACK"
15 PRINT "SEE YOU LATER"
20 GOTO 10
```

The Program Cursor and the EDIT Command

We'll add some punctuation. If you retyped that program for this lesson, you'll see the *program cursor*, which looks like this—▶—at line 20, which was the last line you typed in. Press the up arrow (**SHIFT** 7) and it will move to line number 15. Press the up arrow again and the cursor moves to line 10.

(If you are starting this chapter right after finishing the last one, the program cursor is at line 15. Press the up arrow just once to move it to line 10.)

Then press **EDIT** (**SHIFT** 1) and line 10 will appear in the workspace at the bottom of the screen.

The Comma

Now move the cursor to the end of the line by repeatedly pressing the right arrow (**SHIFT** 8). Then type in a comma (**SHIFT** PERIOD), and **ENTER**.

The program doesn't look much different. But press **RUN** and **ENTER**.

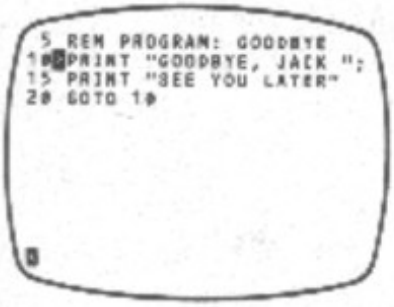
The T/S 1500 screen is 32 characters wide (they are numbered 0-31 instead of 1-32, by the way), and the comma moves the *print position* to the beginning of the next half screen.

What happens if you add a comma to the end of line 15 in the same way? Does it change the output (what appears on the screen)? Why or why not?

No punctuation at the end of a **PRINT** statement moves the print position to the beginning of the next line.

A comma at the end of the print statement moves the print position to either column #16 (the right half of the screen) or position #0 (the beginning of the next line), whichever is next.

Chapter 6: Arranging Output on the Screen



Reminder: what's the difference between the comma inside the quotes and the comma outside?

The Semicolon

Using the same technique as before (arrows, cursors, **EDIT** key) and the **DELETE** key, replace the comma at the end of line 10 with a semicolon.

Move the program cursor to line 10 with the up and/or down arrows (**SHIFT** 6, **SHIFT** 7).

Bring the line down into the work area with **EDIT** (**SHIFT** 1).

Move the cursor to the end of the line with the right arrow (**SHIFT** 8).

Erase the comma with **DELETE** (**SHIFT** 0).

Add a semicolon (**SHIFT** X).

Press **ENTER**. Now **RUN** the program.

Hmm. What have we here?

A semicolon moves the print position one space from where the previous statement ends.

We have to figure out a way to leave a space between the two phrases.

The way you do it is to add a space *inside* the quotation marks. Same way as before:

Press **ENTER** to get the program listing.

Move the program cursor to line 10.

Press **EDIT** to bring it down.

Move the cursor with the right arrow to a position just before the quotation marks.

Type a **SPACE**. It is inserted at the point of the cursor. If you want to get fancy, back up the cursor (with the left arrow) to before the space and also add a comma.

Chapter 6: Arranging Output on the Screen

```
 5 REM PROGRAM: GOODBYE
10 PRINT "GOODBYE, JACK ";
15 PRINT "SEE YOU LATER"
20 GOTO 10
```

Press ENTER.

```
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
GOODBYE, JACK SEE YOU LATER
3715
```

Press RUN and ENTER.

Note: if, for any reason, you want to separate two numbers, you have to add spaces *in quotes*, which means you have to add the quotes, too.

PRINT	1234	prints	1234
PRINT	12 34	prints	1234
PRINT	12;" ";34	prints	12 34

Don't forget the semicolons. . .

What would happen if you put a comma at the end of line 15? What about a semicolon?

There are other ways to position printing on the screen. Let's try them.

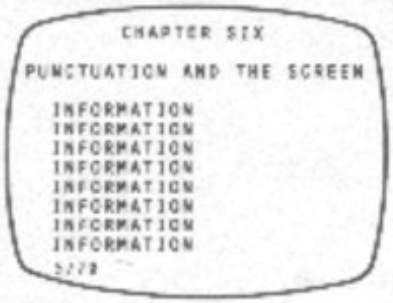
TAB and AT

First, let's clear the computer by pressing **NEW** and **ENTER**.

Then, type in this program:

```
 5 REM PROGRAM: CHAPSIX
10 PRINT TAB 10; "CHAPTER SIX"
20 PRINT AT 5,3; "PUNCTUATION
  AND THE SCREEN"
30 PRINT
40 PRINT
50 PRINT
60 PRINT
70 PRINT TAB 5; "INFORMATION"
80 GOTO 70
```

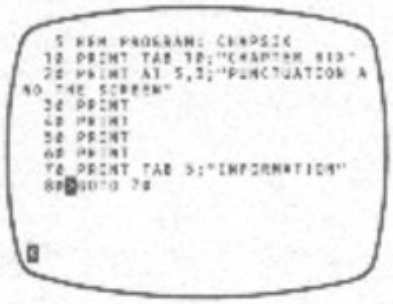
Chapter 6: Arranging Output on the Screen



Press **RUN** and **ENTER**. Nice page, eh?

What does the report code 5/70 mean?

Hint: Check the Report Code appendix.

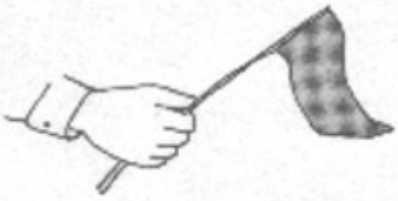


Let's examine the program, line by line. Press **ENTER** to get it back on the screen.

Line 5 is just our standard **REM** statement, containing the name of the program (the same name we use to **LOAD** or **SAVE** it). This is not required, but is probably a good habit to get into.

Line 10 prints beginning at column 10 across the screen—just like the TAB key of a typewriter, except that you specify the column number in the program line. Remember to obtain **TAB** as a *function* (under the P key), using **SHIFT ENTER** to get into *function mode*, rather than spelling it out.

Notice the semicolon after **TAB 10**. **TAB 10** puts the "print position" at the 10th column, but a comma would move it to column 16 and defeat the purpose of the **TAB** statement. And no punctuation at all would result in a syntax error marker.



Is **PRINT**, "JOE" (with the comma) the same as **PRINT TAB** 16; "JOE"?

Line 20 prints **AT** a location defined by the two numbers—the first is the line number, the second is the column number (five lines down, three columns across). **AT** is also a function keyword, under the C key. Notice, again, the semicolon.

Lines 30-60 each print a blank line, effectively moving the print position down by four lines, before line 70.

Line 70 prints, on the next line and at the specified **TAB** location, its information.

Line 80 causes line 70 to repeat until the screen is full.

Chapter 6: Arranging Output on the Screen

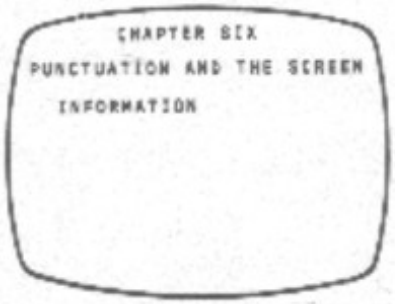
Could you change line 70 to read

PRINT AT 10,5; "INFORMATION"

Try it and **RUN** the program.

Hint: You'll have to use **BREAK** to stop the program.

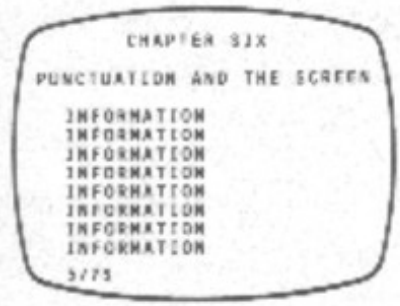
Does it make any difference if you add a comma at the end of line 70? Why or why not?



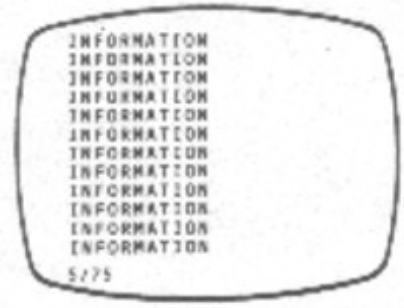
How about if you add a line 75 that reads like the original line 70, and change the **GOTO** in line 80, like this:

70 **PRINT AT** 10,5; "INFORMATION"
75 **PRINT TAB** 5; "INFORMATION"
80 **GOTO** 75

Then **RUN** and **ENTER**.



What happens if you press **CONT** and **ENTER**?



The Quote Image

You may be wondering, if quotation marks tell the computer where to start and stop a string, how can you print quotation marks themselves on the screen?

The double set of quotes on the Q key is called the *quote image*, and serves that function.

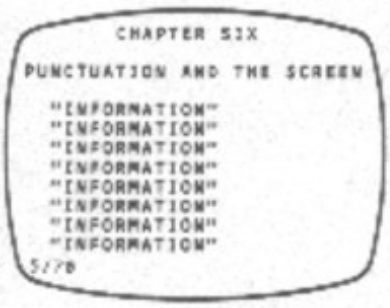
Chapter 6: Arranging Output on the Screen

Let's eliminate line 75 by typing 75 ENTER, change line 80 back to

80 GOTO 70

(Just type in the new line 80 and it replaces the old one), and then change line 70 to look like this

70 **PRINT TAB** 5; " " "INFORMATION" " "



(after **TAB** 5; you type **SHIFT** P and then **SHIFT** Q and, after typing INFORMATION, type **SHIFT** Q and then **SHIFT** P) you will get a curious looking arrangement in your program listing but the right kind of quotation marks when the program prints it.

Remember, keep the quote image (**SHIFT** Q) *inside* the real quotation marks (**SHIFT** P).

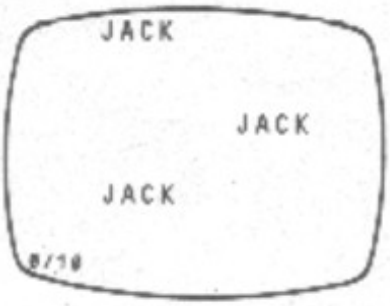
Here's something else you may want to try. In some forms of BASIC, you can have more than one command on a program line, but not in T/S 1500 BASIC. However, you can have multiple **PRINT** statements on one line.

First, we need to remove the CHAPSIX program from the computer. Type **NEW** and **ENTER** (This is the last time we'll remind you to do this . . .).

Then type in the following one-line program—very carefully!

10 **PRINT TAB** 5; "JACK"; **AT** 5,10; "JACK",,,,**TAB** 5; "JACK"

RUN the program.



It will print the screen shown at the left. **TAB** 5 prints in the first line at column 5, **AT** 5,10 prints at line 5, column 10, the four commas then move the print position to line 5, column 16, then line 6, column 0, then line 6, column 16, then line 7, column 0, and then the **TAB** 5 moves it to column 5, still on line 7.

Try a few arrangements yourself.

Summary

1. After a **PRINT** command with no punctuation following, the print position for the next print command moves to the beginning of the next line on the screen.

   PRINT "JACK"

2. A *comma* after a print command moves the print position to the middle of the screen, or to the beginning of the next line, depending on whether the end of the item that has just been printed is in the left- or right-hand half of the screen. You can use more than one comma to move the print position as far as you like by half-lines.

   PRINT "JACK",,,

3. The *semicolon* moves the print position just one character space to the right.

   PRINT "JACK";

4. **TAB** sets the print position at the column called for by the number following **TAB**. Remember, column 0 is really the first column, column 9 is really the tenth, and so on.

   PRINT TAB 10; "JACK"

5. **AT** sets the print position according to two numbers as co-ordinates, separated by a comma: the first is the line number (counting down from the top of the screen) and the second is the column number (counting across from the left edge)

   PRINT AT 5,15; "JACK"

6. **TAB** and **AT** move the print position; *before* printing what is on their program line: the comma and semicolon move the print position *after*, in preparation for the next line, so you could have both:

   PRINT AT 5,15; "JACK"'',

7. The *quote image* (**SHIFT** O) is used to print quotation marks on the screen.

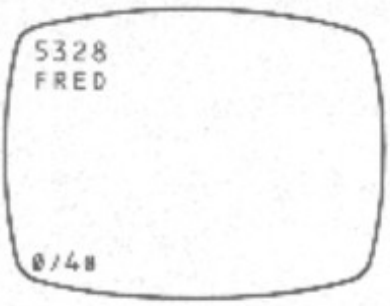
Saving Time and Space with Variables 7

Chapter Preview

You learn to use the LET command to name numbers, words or sentences. We clear the screen with CLS and the memory with CLEAR, and introduce "variables."

```
10 LET A = 5328
20 LET A$ = "FRED"
30 PRINT A
40 PRINT A$
```

Type in the above program. Remember the **SHIFT** key for $, = , and ". Then press **RUN** and **ENTER**. Can you see what has happened? Press **ENTER** again, and your *program listing* will be back on the screen.



Lines 10 and 30 have the same effect as if you had typed **PRINT** 5328. The letter A, when preceded by **LET** and followed by = , becomes a *variable*. In the program above, of course, it takes more time and more memory space in the computer to do it this way. But if you had a program where the

Chapter 7: Saving Time and Space with Variables

number was printed several times, using the variable A instead of the entire number 5328 each time would be quicker, and would save precious memory space in the computer.

In some programs, as we will see later, the number that A stands for changes during the program; that's why A is called a variable.

The program statement

10 **LET** A = 5328

is called an *assignment statement* because it *assigns* a *value* (5328) to a *variable*. The letter A is called a *variable name*. A variable name does not have to be a single letter. It can be any length and contain letters or numbers but the first character must be a letter. The statement could be written

10 **LET** NUMBER = 5328
10 **LET** THIS NUMBER = 5328
10 **LET** A5328 = 5328

and so forth.

Lines 20 and 40 do the same thing as lines 10 and 30, except that A$ is a *string variable*. (A$ is pronounced "A string" rather than "A dollar sign.")

Strings can be any length—whatever is between quotation marks—but the name of a string variable must be one letter followed by $. You could write

25 **LET** A$ = "HELLO"
25 **LET** A$ = "WAY DOWN UPON THE SWANNEE RIVER"
25 **LET** B$ = "5328"

Now we have the program on the screen. Press the V key, which in keyword mode will give you **CLS**, and press **ENTER**.

CLS stands for CLEAR SCREEN. The program is gone. But if you press **ENTER** again, it comes back. It was taken off the screen, but it stayed in memory.

Now press the A key, for **NEW**, and **ENTER**. Then press **ENTER** again. The program, this time, is gone. **NEW** erases everything in the computer and on the screen, and readies the T/S 1500 for a **NEW** program.

Let's go back to the immediate mode and try something else. Type

LET A = 5 **ENTER**

then press **ENTER** again. No program. Nothing in the computer? Try

PRINT A **ENTER**

Well, what do you know? The computer saves variables! Now this can get cluttered after a while, so there is a command to clear variables out of the memory. Press

CLEAR ENTER

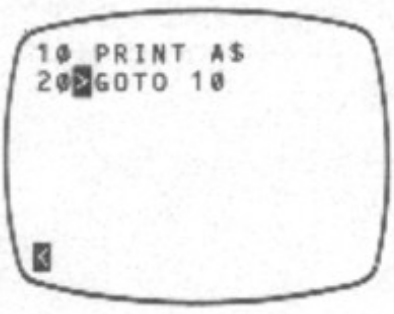
then

PRINT A **ENTER**

As the table of report codes at the back of the book tells us, 2/0 means we have an *undefined variable*. That's because we took the variable A and its assigned value out of the memory with **CLEAR**.

Let's try something else. Type this program

```
10 PRINT A$
20 GOTO 10
```



remembering, of course, to type **ENTER** at the end of each line.

From now on, we're not going to talk about **ENTER**! *But you must remember to press* **ENTER** *after every command or program line.*

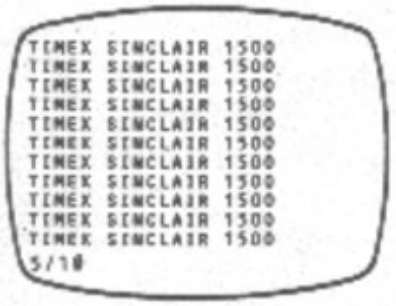
Can you guess what will happen when we **RUN** this program? Try it.

(Did you remember **ENTER**?).

A 2/10 report code this time — because we didn't define the variable A$. Press **ENTER** to bring back the program. Now type, without a line number,

LET A$ = "TIMEX SINCLAIR 1500"

(or any other words if you prefer). Press **ENTER** again to get the program back on the screen and notice that the assignment statement isn't in it (because we didn't give it a number).



Now type

GOTO 10

GOTO 10 started the program running, and it used the variable that was stored in the computer's memory. Why did we use **GOTO** 10 instead of **RUN**? Try to execute the program using **RUN**.

When you use **RUN** to execute a program, in effect you are saying **CLEAR** the variables and then **GOTO** line 1, the beginning of the program. This is so that leftover variables do not gum up the program. But if you want to use previously entered variables, just start a program with **GOTO**.

Enter a new string value for A$, using the **LET** statement. Run the program again, using **GOTO**. Then press **CLEAR**. Get the program back again with **ENTER**. Run it with **GOTO** again.

CLEAR eliminates variables, but leaves the program in the T/S 1500.

Summary:

1. **LET** assigns values to variables.
2. Numeric variable names begin with a letter and can be any length.
3. String variable names are a single letter and $.
4. String variables can be any length, enclosed by quotes.
5. **CLS** clears the screen.
6. **CLEAR** erases variables from the memory.
7. **RUN** starts a program after clearing variables.
8. **GOTO** starts a program (at any line number you choose) without clearing the variables.
9. **NEW** clears everything from the computer.

Programs That Repeat: Looping

8

Chapter Preview

Repetitious work is easy with FOR, TO, NEXT, and STEP. We also use LIST to print the program on the screen.

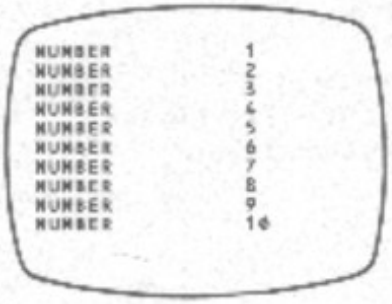

```
10 FOR I=1 TO 10
20 PRINT "NUMBER",I
30 NEXT I
```

10 FOR I – 1 TO 10
20 PRINT "NUMBER", I
30 NEXT I

Type this program into your Timex Sinclair 1500. In line 10, do not spell out **TO**, but use **SHIFT** 4. **RUN** the program.

We said earlier that **GOTO** was a very powerful BASIC statement. We used it to make a program repeat, by telling the computer to **GOTO** an earlier line number and execute the same operations again.

The process of repeating the same operations a number of times is called *looping*, and the part of the program that repeats is called a *loop*. When



we simply use **GOTO**, we have no control over how many times the program repeats; in fact, it does not stop (until the screen fills, or we press **BREAK**).

This is called an *endless loop*, and is not too useful.

The FOR/NEXT Loop

We need loops that are under our control. In Sinclair BASIC, the best way to do this is with what is called a **FOR/NEXT** *loop*. The program above shows the form of such a loop:

Line 10 contains the keywords **FOR** and **TO**, a number indicating how many repetitions are desired, and a *counter* or *control variable*. A control variable in a **FOR/NEXT** loop must be one single letter. Computer jocks usually use I, but any letter will do.

Line 20 contains the action that is to be repeated within the loop. There can be many lines of activity here, not just one.

Line 30 is necessary to close the loop. It tells the computer where to stop and go back to the **FOR** line.

Try this: get the program back on the screen with **ENTER**, then add

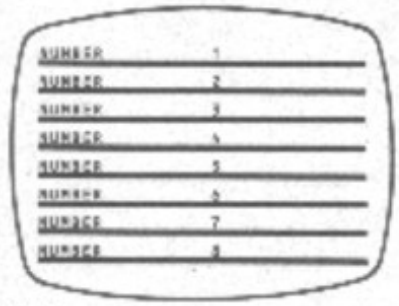
25 **PRINT** "

"

Between the quotes is the graphic symbol on the 7 key. Remember how to get it?

After you've pressed **SHIFT** P for the ", press **SHIFT** 9 to enter the *graphics mode*, with the G cursor on the screen. Then press **SHIFT** 7 (7 without **SHIFT** is just inverse 7) 32 times, because the screen is 32 characters wide.

Then press **SHIFT** 9 to exit the graphics mode and, with the L cursor showing, close the quotes with **SHIFT** P. (Then **ENTER** all that.) **RUN** the program and you'll see a dandy bunch of underlining.



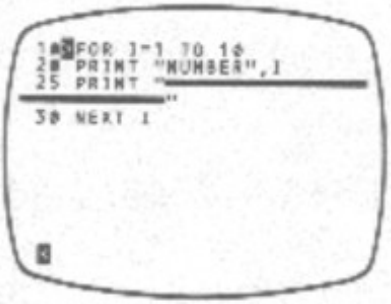
etc. . . .

Incidentally, another way to determine if you've got your full 32 characters—a full line on the screen—between the quotes is to see that the close quote is one space past the open quote (but on the next line). Put another way, see that the material between the quotes would meet if it were on the same line.

The LIST Command

Bring back the program with **ENTER**. You see the *program cursor* on line 25—the last line you entered. We want to **EDIT** line 10. But instead of using the cursor arrows, let's look at another technique. Type

LIST 10

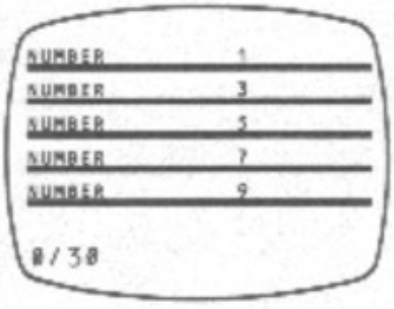


In most BASICs, pressing **ENTER** does not bring back a program listing as it does on the T/S 1500. The command **LIST** is required, and you can add a line number for where to begin the listing. When you do that on the Timex computer, you get the program cursor at the first line on the screen. (This is much more useful when you want to edit line number 90 in a program that goes to line 220 than it is for our example. . .)

Adding STEP to the FOR/NEXT Loop

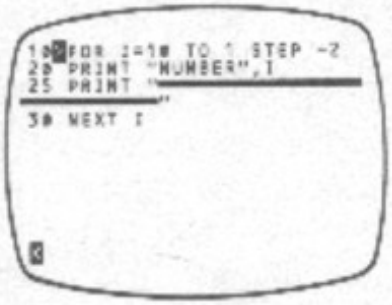
Nevertheless, we have the program cursor where we want it. Press **EDIT** and bring line 10 down, then use the right cursor arrow (**SHIFT** 8) to move to the end of the line and add **STEP** 2—using **SHIFT** E, not spelling out **STEP**):

10 **FOR** I = 1 **TO** 10 **STEP** 2



Enter the line back into the program, then **RUN**.

STEP does just what you'd expect: it "counts by" the number following the command **STEP**. How would you get the program to count 2, 4, 6, 8, 10 rather than 1, 3, 5, 7, 9? Try it.



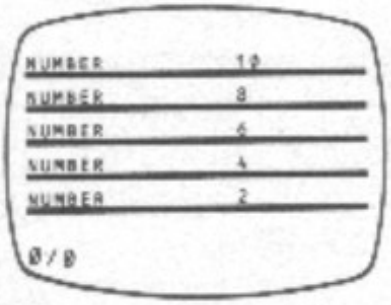
You can "count down" with **STEP**, too. Try changing line 10 to read

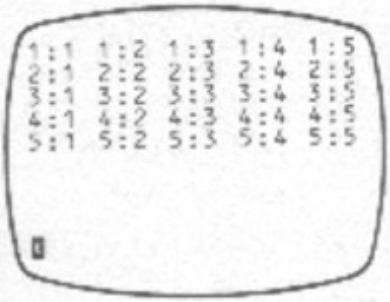
```
10 FOR I – 10 TO 1 STEP – 2
```

Will the computer count down by ones if you say

```
10 FOR I – 10 TO 1
```

or do you have to add **STEP** – 1? Try it and see.





Nested Loops

Can you have a loop inside a loop? Certainly. Many times, you'll want to repeat actions which contain other actions which you also want repeated. You can do this more or less indefinitely, as long as the loops are properly *nested*. Try this:

```
10 FOR I – 1 TO 5
20 FOR J – 1 TO 5
30 PRINT I;":";J;" ";
40 NEXT J
50 PRINT
60 NEXT I
```

Be careful with line 30: get it just right, and put a space between the second pair of quotes. And notice we had to pick a second letter, besides I, to count for the second loop.

RUN the program.

Effectively, the rows are created by the "I-loop" and the columns by the "J-loop." (The **PRINT** in line 50 serves to move the print position to the beginning of the next row to start a new I loop.)

The loops are correctly *nested*, because the entire J-loop is contained within the I-loop. You will have a problem if the loops overlap. Sometimes you'll get error messages, sometimes you'll just get incomprehensible results. Try exchanging lines 40 and 60, so the program looks like this:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 5
30 PRINT I;":";J;"  ";
40 NEXT I
50 PRINT
60 NEXT J
```

RUN it and try to figure out what you have!

```
1:1  2:1  3:1  4:1  5:1
6:2
7:3
8:4
9:5
```

One other problem you must be careful of when using **FOR/NEXT** loops: you can't "jump" into the middle of a loop from the outside by using a **GOTO**. When it hits the **NEXT** command without having passed a **FOR**, the trouble will start.

Summary

1. **FOR/NEXT** loops, using the keywords **FOR**, **TO** and **NEXT**, and a *control variable*, give you controlled repetitions.
2. Control variables are named by any single letter.
3. **STEP** is used in a **FOR/NEXT** loop to count by anything other than ones, and by negative numbers.
4. Multiple loops must be nested, not overlapped.
5. **LIST** brings a program listing to the screen; with a line number after **LIST**, it starts the listing at that number and places the program cursor at that line.

Programs That Decide: Branching

9

Chapter Preview

IF and THEN are used, with the mathematical relations =, <, >, <=, >=, and <>, to make decisions. We also use INPUT to give information to the T/S 1500.

We've considered three of the four reasons that the computer is such a powerful and valuable tool:

1. It works fast.
2. It can remember a lot of information, including its own instructions (programs).
3. It can repeat operations over and over, tirelessly (looping).

Now let us look into the fourth:

4. The computer can make decisions.

A program can contain a number of instructions, some of which are carried out by the computer in some cases, and others which are carried out in other cases. Such a program is said to *branch*, or to be a *branching program*. The command that makes all this possible is **IF**.

```
10 PRINT "NUMBER","LARGEST SO
FAR"
20 INPUT A
30 LET LARGEST=A
40 PRINT A,LARGEST
50 INPUT A
60 IF LARGEST<A THEN LET LARGE
ST=A
70 GOTO 40

L
```

```
10 PRINT "NUMBER","LARGEST SO FAR"
20 INPUT A
30 LET LARGEST = A
40 PRINT A,LARGEST
50 INPUT A
60 IF LARGEST< A THEN LET LARGEST = A
70 GOTO 40
```

Type in the program above but before you **RUN** it, let's talk about it—line by line—to review some earlier points and raise some new ones. **THEN**, incidentally, is **SHIFT** 3, not spelled out.

Line 10 is a simple print statement; the comma between the two *strings* means they will be printed at positions 0 and 16 on the screen—the left edge and the middle.

Line 20 is an **INPUT** statement. It asks for some information from the user, and assigns the input to a variable named A. When you **RUN** the program, the **L** cursor at the bottom of the screen signals you that the T/S 1500 is waiting for your input. More about **INPUT** in a moment.

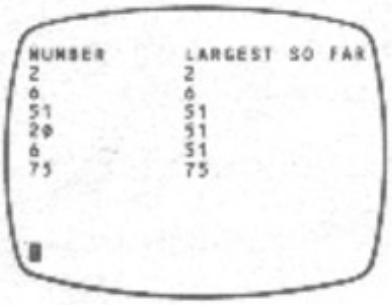
Line 30 sets a variable, LARGEST (notice that the variable name LARGEST helps you remember what it is) equal to the variable A.

Line 40 prints those two variables, directly under the labels printed in line 10 (notice the comma again).

Line 50 asks you to input another number and assigns it to A. This replaces the previous value for A.

Line 60 is the decision statement. **IF** the number that is called LARGEST is LESS THAN (<) the new value for A, the program sets it to be equal to A . . . the latest input is the "largest so far." If LARGEST is already larger than A (or equal to it), it is left alone.

Line 70 creates a loop by going back to line 40. Then only the portion of the program between



lines 40 and 70 is repeated, as often as you care to keep entering new numbers.

Okay, now **RUN** the program and enter some numbers to see how it works.

The STOP Statement

Because there is an endless loop in the program, it won't stop by itself. (This is a less serious problem in a program that pauses to wait for input than it would be in a program that could keep running.) To terminate the program, when the L cursor is on the screen waiting for input, type **STOP** (**SHIFT** A) instead of inputting a number.

Then you can recall the listing with **ENTER**, or restart the program with **RUN**.

As we will see in a moment, we can also tell a program when to **STOP** itself by putting that command in a program line.

It is also possible to interrupt this program by pressing the **BREAK** key, but you have to be fast—this only works when the program is not waiting for input. Try it: you have to press **BREAK** when the L cursor is *not* on the screen.

The INPUT Statement

The command **INPUT**, followed by a variable name, stops the program until you enter a value for that variable. The L cursor appears at the bottom of the screen; that is your clue that the computer is waiting for input.

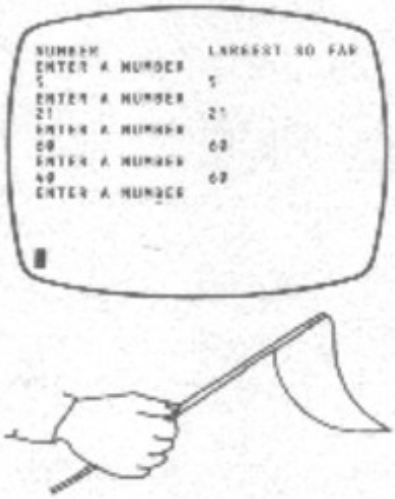
It is usually preferable to insert a **PRINT** statement just before an **INPUT** statement, describing the information the program needs. For example, we might add to our program

15 **PRINT** "ENTER A NUMBER"

and

45 **PRINT** "ENTER A NUMBER"

Chapter 9: Programs That Decide: Branching



Try it and see what happens. That's a little cluttered. We could erase the prompt with **CLS** each time, so that you would notice it when it came back—but then we'd erase our column of numbers and our heading!

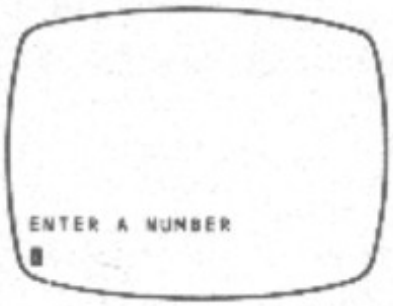
Here's a fancy way to add a prompt. Eliminate the new lines 15 and 45, if you added them above. Remember, you delete a line by replacing it with a blank line with the same number, so you'd type 15 **ENTER** and 45 **ENTER**.

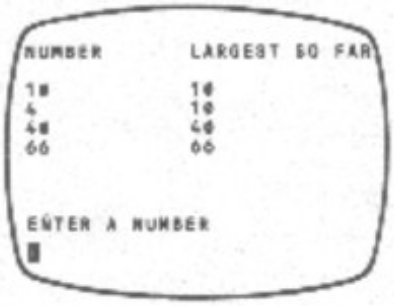
Then try adding a few lines so the program looks like this (remember, in lines 15, 40 and 45, **AT** is the function under the C key and is not spelled out):

```
10 PRINT "NUMBER","LARGEST SO FAR"
15 PRINT AT 20,1;"ENTER A NUMBER"
20 INPUT A
25 PRINT AT 20,1;"                "
30 LET LARGEST = A
35 LET X = 3
40 PRINT AT X,0; A, LARGEST
45 PRINT AT 20,1;"ENTER A NUMBER"
50 INPUT A
60 IF LARGEST< A THEN LET LARGEST = A
65 LET X = X + 1
70 GOTO 40
```

Lines 15 and 45 **PRINT** the prompt **AT** line 20, on the bottom of the screen (and starting at column position 1), while lines 25 and 55 effectively erase the prompt by "printing" a string of blank spaces instead.

Line 40 has to be changed to specify the location of the printing of the next pair of numbers because the **PRINT AT** command in lines 15 and 45 moves the print position to line 20. If we did not change line 40, the next pair of numbers would be printed on line 21.





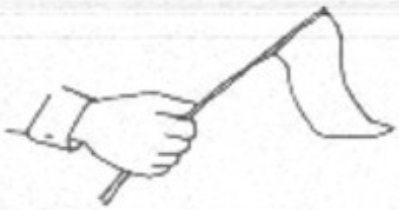
We use the variable X for the line in that **PRINT AT** statement in line 40. Line 35 defines X and prints the first pair of numbers on line 3, then line 65 increases X so that each repetition of the loop prints the numbers on the next line.

Incidentally, doesn't line 65 look odd? What kind of math is X = X + 1? Well, of course it isn't math at all, but another *assignment statement*. It means "let X equal the previous value of X, plus one."

This means you have another way of writing a loop. It is a little clumsier than a **FOR/NEXT** loop, so we seldom use it. The one advantage it has is that you could use a more descriptive variable name in place of X—you might say

```
35 LET LINE = 3
65 LET LINE = LINE + 1
```

where, you will remember, variable names that count for a **FOR/NEXT** loop must be only a single letter.



Try erasing line 65 and see what happens. It is because X is not increased, and returns to line 3 each time we repeat the loop.

Can you print your prompts in lines 15 and 45 in inverse letters?

The IF Statement

The **IF** statement checks to see whether a particular condition is true; if it is, the rest of the statement on that line is executed. If it is not, the rest of the line is ignored and the program moves to the next line.

Often the form of the statement is like

```
40 IF A = 5 THEN GOTO 100
```

```
10 LET I=1
20 PRINT I*100
30 LET I=I+1
40 IF I=6 THEN STOP
50 GOTO 20

K
```

```
100
200
300
400
500

0/40
```

```
10 FOR I=1 TO 5
20 PRINT I*100
30 NEXT I


K
```

meaning that if a variable called A is equal to 5, the program goes to line 100 and starts running at that line. If A is not equal to 5, the next line executed would be the one after 40 (probably 50, right?).

In T/S 1500 BASIC, by the way, we must include the **THEN** in line 40 (some BASICs let you omit it). Among the things **THEN** does is to return the K cursor to the screen so you can use a keyword like **GOTO**; otherwise you wouldn't be able to give the computer any commands to execute IF A = 5.

Sometimes—as in the example at the beginning of the chapter—we have to **INPUT** the value on which the decision is made. More often, the value results from some calculations within the program. We can even use **IF** to terminate a loop:

```
10 LET I = 1
20 PRINT I*100
30 LET I = I + 1
40 IF I = 6 THEN STOP
50 GOTO 20
```

How would you write that program using a **FOR/NEXT** loop?

```
10 FOR I = 1 TO 5
20 PRINT I*100
30 NEXT I
```

The **IF** statement compares values using these mathematical symbols:

=	is equal to	SHIFT L
<	is less than	SHIFT N
>	is greater than	SHIFT M
<=	is less than or equal to	SHIFT R
>=	is greater than or equal to	SHIFT Y
<>	is not equal to	SHIFT T

Do not assemble a "less than or equal to" sign by typing **SHIFT** N and **SHIFT** L. You must use **SHIFT** R for the combination. The same goes for > = and < >

Comparing Strings

You can also use the symbols to compare strings. Usually you will do this with = or < > to see if an input matches a previously-chosen word. Type in this program, in which Fred wants you to try to guess his name.

```
10 INPUT A$
20 IF A$="FRED" THEN GOTO 40
30 GOTO 10
40 PRINT A$
```

10 INPUT AS
20 IF A$ = "FRED" **THEN GOTO** 40
30 GOTO 10
40 PRINT AS

RUN the program. Notice the L cursor at the bottom of the screen is in quotes, prompting you to enter a string.

```
FRED


0/40
```

Make a few wrong guesses, then input the right answer.

Now try this: add a new line

5 **LET** B$ = "FRED"

and change line 20 to read

20 **IF** A$< = BS **THEN GOTO** 40

```
 5 LET B$="FRED"
10 INPUT A$
20 IF A$<=B$ THEN GOTO 40
30 GOTO 10
40 PRINT A$
```

You can use the cursor arrows, **EDIT** and **DELETE**, to substitute or you can just type a new line in. You are replacing "FRED" with B$, and the = sign with < = (**SHIFT** R, not **SHIFT** N and **SHIFT** L).

Then guess these names, JIM, HARRY, JOE, and AL. What happened?

The mathematical symbols operate on a string by comparing the first letter of the string. If the first letter of A$ comes earlier in the alphabet than the first letter of B$, then A$ is said to be

less than B$. If the first letters of two strings are the same, then the second letters are compared, and so on. So, with strings: ***earlier** in the alphabet is **less than**, **later** in the alphabet is **greater than**.*

In fact, what the computer actually does is compare the *code numbers* of any characters in the T/S 1500's character set. If you refer to the Appendix titled THE CHARACTER SET, you'll find that A> 9. The alphabet follows the numerals in the character set, so any letter is greater than any numeral.

Test this with Fred's program. Type in a number—say, 25—in answer to the input request.

How would you add prompts to Fred's program—like "Guess my name" and "Wrong. Guess again"?

Hint: you can also change line 40 to read

```
40 PRINT "RIGHT. MY NAME IS  ";A$
```

but only if you have the = sign in line 20, not the < =. Notice the space just before the close quote.

AND, OR, NOT

We call =, <, >, < =, > = and < > *relations*. We can combine them by using the *logical relations* **AND**, **OR** and **NOT**.

```
IF A = B AND C = D THEN GOTO 100
```

means that if both pairs of variables are equal, the program goes to line 100.

```
IF A< B OR C> D THEN GOTO 100
```

sends the program to line 100 if *either* A< B *or* C> D is true.

```
IF NOT A = B THEN GOTO 100
```

is the same as

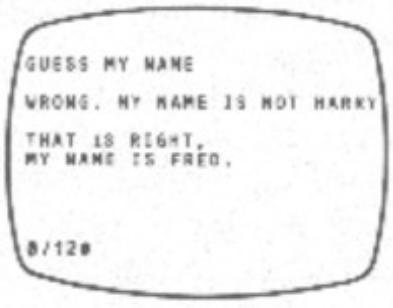
IF A< > B **THEN GOTO** 100

In Sinclair BASIC, unlike some other "dialects," you must include both **THEN** and **GOTO** in lines like the above, to enable the unique keyword system to operate.

But in addition, using both makes the program clearer and easier to understand.

Let's add a few lines to the name-guessing program:

```
 10 PRINT "GUESS MY NAME"
 20 LET I = 0
 30 INPUT A$
 40 IF A$ = "FRED" THEN GOTO 110
 50 LET I = I + 1
 60 IF A$< > "FRED" AND I = 3 THEN GOTO 90
 70 PRINT "WRONG. MY NAME IS NOT ";A$
 80 GOTO 30
 90 PRINT "SORRY, YOU LOSE."
100 GOTO 120
110 PRINT "THAT IS RIGHT,"
120 PRINT "MY NAME IS FRED."
```



RUN the program and try a few guesses. The combination of relations in line 60 stops the game if you've had three guesses. Trace the logic in the program and figure out how and why it moves from one line to another.

Try to tidy up the screen a bit by adding

15 **PRINT**

and

75 **PRINT**

```
 10 PRINT "GUESS MY NAME"
 20 LET I=0
 30 INPUT A$
 40 IF A$="FRED" THEN GOTO 110
 50 LET I=I+1
 60 IF A$<>"FRED" AND I<3 THEN
GOTO 90
 70 PRINT "WRONG. MY NAME IS NO
T "|A$
 80 GOTO 30
 90 PRINT "SORRY, YOU LOSE."
100 GOTO 120
110 PRINT "THAT IS RIGHT."
120 PRINT "MY NAME IS FRED."
```

Now we have a confession to make. Did you discover, when you analyzed the program, that line 60 doesn't need both relations?

Since line 40 transfers the program to line 110 if A$ = "FRED" then, logically, A$ must be not equal to "FRED" if we have reached line 60. So line 60 really only needs to say

60 **IF** I – 3 **THEN GOTO** 90

Try it and prove it to yourself. The lesson in this is to always plan a program carefully in advance and work out the logic of it so that you do things in the most direct way. The search for the simplest solution to a programming problem is what makes programming fun!

Summary

1. **INPUT** causes a program to stop and wait for a number or a string to be entered. An L cursor at the bottom of the screen signals that the computer is waiting for a number, and "L" means a string is expected.
2. It is a good idea, even with the cursor serving as a "prompt" for **INPUT**, to **PRINT** a line explaining the information the program wants; the **PRINT** statement should be in the program line just before the **INPUT** statement.
3. **STOP** (**SHIFT** A) can be input in response to the L cursor prompt, and stops the program.
4. **IF/THEN** evaluates a condition; if the condition is true, the program does what is called for after **THEN** (usually **GOTO** another location in the program); if not, the next line in the program is executed.

5. IF evaluates mathematical values using the relations
   - = equal to
   - < less than
   - > greater than
   - <= less than or equal to
   - >= greater than or equal to
   - <> not equal to
6. More than one mathematical relation can be combined, using
   AND (both are true)
   OR (either is true)
   NOT (a relation is not true)

Mathematics with the T/S 1500

10

Chapter Preview

You can add, subtract, multiply, divide and use built-in functions like RND and INT.

If you have used a calculator, you are used to typing in something like this

2 + 2 =

and getting the answer. Try this on your T/S 1500. Nothing happens. Press **ENTER**. A syntax error marker appears.

This doesn't look promising. Press **SHIFT 0** (**DELETE**) until you get rid of it all.

You can use your computer as a calculator, but you have to ask the right questions. Try this:

PRINT 2 + 2

Okay, that's more like it. You can use any mathematical operation in the same way. The signs are

+	SHIFT K	addition
–	SHIFT J	subtraction
*	SHIFT B	multiplication
/	SHIFT V	division
**	SHIFT H	raising to a power

The addition and subtraction signs are the ones you are used to. Division uses a sign you have probably seen, because the computer does not have the ÷ sign in its character set. And an asterisk is used to stand for multiplication because the X is being used as a letter.

Raising a number to a power is a special case. The T/S 1500 cannot insert or understand *superscripts*, which is what we call our usual notation, so we use the ** symbol. (Other computers and BASICs use other symbols.)

3^2 = 3 squared	=	3**2
3^3 = 3 cubed	=	3**3
3^4 = 3 to the fourth power	=	3**4
3^{10} = 3 to the tenth	=	3**10

All of these mathematical operations can take place in programs, of course.

Priorities and Parentheses

If you have a program line containing a number of mathematical operations, the Timex Sinclair 1500 will perform them in this order:

First, it will work out any powers, starting with the left end of the line and working to the right.

Second, it will do all multiplication and division, again working from left to right.

Finally, it will do all addition and subtraction, once again from left to right.

These are called the *priorities*, and are part of the way the computer is designed. Just as the

character set includes more than just the letters of the alphabet, in fact, the priority rankings go well beyond the basic mathematical operations—a complete table can be found in Appendix A.

You may want to have operations performed in an order different from the computer's way, and you can arrange this by using parentheses: anything in parentheses is done first (left to right) and the result is treated as a single number.

For example

```
PRINT 3*4 + 3 = 15
```

because 3*4 = 12 (multiplication first) and then 12 + 3 = 15. But

```
PRINT 3*(4 + 3) = 21
```

because 4 + 3 = 7 (parentheses first) and then 3*7 = 21.

You can go further, putting parentheses inside parentheses (correctly nested, just like loops): the innermost parentheses will be done first, and then the computer will work its way out from there.

Scientific Notation

Sometimes, when a number is going to be more than 14 spaces long, the T/S 1500 will print it in *scientific notation* instead. This is a number with one digit to the left of a decimal point, some digits to the right of it, and then an E (for *exponent*), a + (or sometimes a minus) and a number which multiplies the rest of the expression by powers of 10.

For example,

```
2.34E + 14
```

is 2.34 times 10 to the 14th power.

Try typing in any number more than 14 digits long (after a **PRINT** statement), like

PRINT 23456789234567 89

and see what happens.

You can also use scientific notation in entering numbers, and the computer will convert the number to an ordinary expression—until it gets to be more than 14 digits long, and then it will go into scientific notation itself. Try

PRINT 2.34E0
PRINT 2.34E1
PRINT 2.34E2

and so on. At what point does the computer start returning scientific notation?

The T/S 1500, with **PRINT**, also has a habit of displaying only eight *significant digits*. Type

PRINT 123456789

and see what the computer displays.

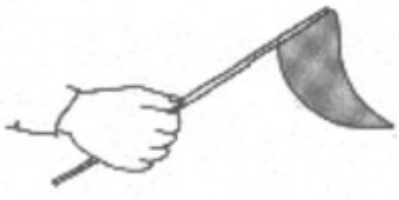
Have no fear—the T/S 1500 is holding the accurate number even though it rounds off and shows only the eight significant digits. Try

PRINT 123456782

to check the rounding-off process, then

PRINT 123456782 + 4

to see that it's all there (if it weren't it would round down again).



Rounding Errors

All computers have a problem with *rounding errors*—answers which are sometimes slightly

incorrect due to the "rounding off" process. This is inherent in the binary-to-decimal conversion operation. Most large computers have editing routines to correct for this built into their math handling procedures, but small personal computers lack such a feature.

If you were to write

```
100 LET A = 300.3 – 236.11 – 55
110 PRINT A
```

you would get an answer from the computer of 9.1899999, when the correct answer is 9.19.

You can correct for this by adding

```
105 LET B = INT(A*100 + .5)
```

and then changing line 110 to read

```
110 PRINT B
```

Functions

Many mathematical functions are built into Sinclair BASIC. To take the square root of 9, for example, you would type

```
PRINT SQR 9
```

using the function **SQR** under the H key. Remember, you get a function with the F cursor, which you obtain by pressing **SHIFT ENTER**.

We will not spend any more time on functions at this time; you'll find them all defined in Appendix A. As we said back in Chapter Four, mathematicians probably know them anyway and the rest of us probably don't need them . . .

The Random Number Generator

There is one function we want to discuss a bit further, because it is very useful in programming educational exercises and games. That is the random number generator.

The function **RND**, located under the T key, gives you a number between 0 and 1. Try a few of them:

```
PRINT RND
```

Now, this may not seem terribly useful, but you can massage it a bit to make it give you whole numbers in any range you want. For example, if you want the computer to "pick a number between 1 and 6," you type

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

INT (for *integer*) is the function under R, conveniently close to **RND**. You have to dip into function mode twice to do this. Here's what is happening:

1. **RND***6 is generating a decimal fraction between 0 and 1, and multiplying it by 6.
2. **INT** is rounding that number down to a whole number. If it's 3.09345622, it becomes 3. If it's 0.97888545, it becomes 0. If it's 5.8760, it becomes 5.
3. That gives you a whole number between 0 and 5, and you wanted one between 1 and 6, so we add one (+ 1). This step is necessary because the **INT** function rounds down, not up.

That's a lot of trouble but, sorry, that's the way it is. You probably ought to write that formula down somewhere, because you'll use it a lot.

"Picking a number between 1 and 6" simulates the roll of a die. Do it twice and you have a dice roll.

The value returned by a **RND** function can be assigned to a variable—

```
LET A = INT (RND*32) + 1
```

—and used for all kinds of things, including deciding where on the screen some symbol is to be placed!

One last thing: **RND** is actually not a true random number generator, but only a "pseudo-random" function. It actually gives you, one at a time, numbers from a long table that has been randomly generated. The table is so long that you won't be able to memorize it, but you can memorize the first few numbers, which will always be the same the first time you plug in the computer.

Prove it. A couple of times, unplug it, then plug it in and try

```
PRINT RND
```

a few times. It's the same sequence!

To get away from the early part of this list, you use the **RAND** keyword, above the T key. When you first plug in the computer, or within a program, before any **RND** functions, type

```
RAND 0
```

When you use the **0**, **RAND** (which stands for *randomize*) finds a place to start in the table based on how long your computer has been on (how many frames have been sent to the TV), which is about as random as you'll need.

On the other hand, if you use any number other than 0, **RAND** starts using the table at a certain point based on that number, so that

```
RAND 50
```

will always start **RND** with the same number. Try it out. You can amaze your friends with it.

Summary

1. The T/S 1500 will carry out the mathematical operations of + (addition), − (subtraction), * (multiplication), / (division) and ** (raising to a power).
   This can be done anytime within a program; in the immediate mode, you need to use a **PRINT** command to see the result.

   ```
   PRINT 2 + 2
   ```

2. Mathematical operations are carried out in a particular order of priority; you can circumvent the priority ranking by using parentheses. Operations in parentheses are executed first.
3. The Timex Sinclair 1500 can understand scientific notation, and will use it with large numbers.
4. The computer has a number of built-in mathematical functions, which can be accessed with single keys in function mode.
5. The function **RND** generates pseudo-random numbers.

Subroutines 11

Chapter Preview

Recycle your program lines with self-contained subprograms, using GOSUB and RETURN.

In a long program, you will sometimes need to repeat the same operation at two or more different locations within the program.

It is a lot of extra work—and unnecessary—to write the same statement lines over and over each time such a repeating operation is needed. Instead, you can use the same portion of the program repeatedly.

If you have a program segment you want to use over and over, you can direct the computer to it by using **GOTO** and the line number. At the end of the segment, you direct the computer somewhere else, with another **GOTO**. For example:

```
10 REM THE MAIN PROGRAM
20
30
40 GOTO 1000
50
60
70

1000 REM THE SUBROUTINE
1010
1020
1030
1040 GOTO 50
```

But suppose you want to call that subroutine from several different places in the program . . . and you don't want to **GOTO** 50 each time when the subroutine is done.

That's what **GOSUB** and **RETURN** are for. **GOSUB** 1000 sends the computer to line 1000 just as **GOTO** 1000 does. **RETURN** directs the computer to the line after the **GOSUB** command. For instance:

```
10 REM THE MAIN PROGRAM
20
30
40 GOSUB 1000
50
60
70 GOSUB 1000
80
90

1000 REM THE SUBROUTINE
1010
1020
1030
1040 RETURN
```

In the program model above, the command **RETURN**, ending the subroutine, directs the computer to line 50 after the first execution (called by line 40) and to line 80 after the second go-around (called by line 70).

GOSUB and **RETURN** can save work for you, and space in the computer's memory. But perhaps more importantly, they help you organize your programs so that other people trying to use them—and you, coming back to them after a lapse of time—can understand how they work.

Structured Programming

If you hang around professional programmers, you'll hear a lot of talk about *structured programming* (and some opinions that "you can't do structured programming in BASIC").

There are many competing definitions of "structured programming," but probably the most common element of the definitions involves breaking large programs into smaller, more manageable *modules*.

Subroutines can be those modules, in BASIC. It can be very helpful to try to use subroutines whenever possible as you create larger programs: plan the overall strategy, and then use subroutines (and sub-subroutines within the subroutines) to execute specific tasks that contribute to solving the overall problem.

```
10 REM PROGRAM: MATH
20 LET A = INT (RND*9) + 1
30 LET B = INT (RND*9) + 1
40 PRINT "DO YOU WANT TO"
50 PRINT TAB 10; "ADD—PRESS 1"
60 PRINT TAB 10; "SUBTRACT—PRESS 2"
70 PRINT TAB 10; "MULTIPLY—PRESS 3"
80 PRINT TAB 10; "DIVIDE—PRESS 4"
90 INPUT D
100 IF D< 1 OR D> 4 THEN GOTO 40
110 CLS
120 GOSUB D*1000
130 INPUT E
140 PRINT AT 10,15;E
150 IF E = C THEN PRINT AT 15,10; "CORRECT"
160 IF E<>C THEN PRINT AT 15,10;
    "SORRY, WRONG NUMBER"
170 PRINT AT 17,10; "WANT ANOTHER—
    Y OR N?"
180 INPUT A$
190 CLS
200 IF A$<>"Y" THEN STOP
210 GOTO 20
1000 LET C = A + B
1010 PRINT AT 10,10;A;" + ";B;" = ?"
1020 RETURN
2000 LET C = A – B
2010 PRINT AT 10,10;A;" – ";B;" = ?"
2020 RETURN
3000 LET C = A*B
3010 PRINT AT 10,10;A;"*";B;" = ?"
3020 RETURN
4000 LET C = A/B
4010 PRINT AT 10,10;A;"/";B;" = ?"
4020 RETURN
```

In line 100 use the keyword **OR** (**SHIFT** W), but in line 170 spell out the word OR.

This example program includes a number of concepts we have discussed in previous chapters. It also illustrates how the use of subroutines

can serve to make a program's structure easy to follow. In fact, this program does not require subroutines to get the job done. Why not? (We'll give you the answer after we call attention to a few of the other features of the program.)

1. Lines 20 and 30 make use of the random number generator.
2. Lines 50-80 use **TAB** to format an indented column on the screen.
3. Lines 90, 130 and 180 use **INPUT**.
4. Line 100 is an *error trap* which repeats the question if you input any character other than one of the choices the program can deal with.
5. Line 120, instead of using 4 different lines with separate **GOSUB** addresses, uses multiplication to select the subroutine. This technique can also be done with **GOTO**.
6. Line 140 uses **PRINT AT**.
7. Lines 150 and 160 are both necessary, unlike the example in Chapter Nine. Why?
8. Line 200 **STOP**s the program if any other key than Y (for yes) is pressed, even though the screen asks for "Y.OR N?" This is because, otherwise, inputting anything other than Y or N would be an error, causing the program to "crash." Could you use an error trap like the one in line 100 instead?
9. Each subroutine does a different mathematical operation, but works on whichever random numbers have been generated and provides an answer against which the user's answer is checked.
10. Rounding errors may give you trouble in the division subroutine. Can you use the correction routine in Chapter Ten to fix this?

The answer to our question is that you could actually use **GOTO** D*1000 for each "subroutine" and **GOTO** 130 instead of **RETURN** at the end of each one.

A Useful Subroutine

The version of BASIC used by the Timex Sinclair 1500 does not have READ, DATA and RESTORE statements. This is not a major problem, as the function of DATA statements can be mimicked in many flexible ways. Occasionally, however, you might wish to translate programs from a "dialect" of BASIC using the DATA statement.

The DATA statement allows a programmer to define a list of numbers in a program. The READ statement retrieves these numbers, one at a time, and assigns them to a variable. The RESTORE statement forces the READ statement to start over at the beginning of the DATA list.

Here's a way to use a subroutine to translate a program which uses DATA statements:

First, include the following subroutine in your program (you may have to change the line numbers if you are already using these):

```
500 LET DI = DI + 1
510 IF D$(DI) = "," THEN GOTO 500
520 LET DJ = DI
530 LET DJ = DJ + 1
540 IF DJ> LEN D$ THEN GOTO 560
550 IF D$(DJ)< > "," THEN GOTO 530
560 LET D = VAL D$(DI TO DJ - 1)
570 LET DI = DJ
580 RETURN
```

Then, add this line to the program, somewhere near the beginning (before any **READ** statements):

```
LET DI = 0
```

Follow that with a substitute data statement using **LET** D$ = "your data". For example, if the original data statement reads

```
DATA 5,10.4, - 3,6.9E - 10
```

you would change it to

LET D$ = "5,10.4 – 3,6.9E – 10"

Replace all **RESTORE** statements in the program with

LET DI = 0

Next, replace all READ statements with

GOSUB 500
LET VARIABLE = D

for example

190 READ A

would become

190 **GOSUB** 500
191 **LET** A = D

(We are indebted to Dave Wood of the Sinclair/ Timex User Group, Boston Computer Society, for this routine.)

Summary

1. Subroutines help you use techniques of "structured programming" in BASIC to make programs easier to use and understand.
2. **GOSUB** directs the program to a specific line, as does **GOTO**, but stores the location of the program line containing the **GOSUB**.
3. **RETURN**, the last line of the subroutine, directs the program to the next line after the **GOSUB**.
4. **GOSUB** and **RETURN** must be used together, like **FOR** and **NEXT**.

Graphics and Other Wonders 12

Chapter Preview

We make pictures and diagrams with PLOT, UNPLOT and the keyboard graphics symbols, and learn how to make things SCROLL. CODE and CHR$ are also explained.


```
10 REM PROGRAM: INCHWORM
20 FOR X = 1 TO 5
30 PLOT X,21
40 NEXT X
50 FOR X = 5 TO 63
60 PLOT X,21
70 UNPLOT X - 5,21
80 NEXT X
```

The program above uses **PLOT** and **UNPLOT** to simulate the progress of an inchworm across the screen. Type it in and **RUN** it.

The format of the **PLOT** statement is

PLOT x, y

with x being a number from 0 to 63, from left to right on the screen, and y being a number from 0 to 43, from bottom to top on the screen.

```
10 PLOT 0,0
20 INPUT A$
30 PLOT 0,43
40 INPUT A$
50 PLOT 63,0
60 INPUT A$
70 PLOT 63,43
```

The **PLOT** command fills in, with a black box, the *pixel* (picture element) located by the two coordinates. The **UNPLOT** command, as you would guess, erases anything **PLOT**ted at the indicated location.

Here's a program that defines the screen for the **PLOT** command, placing a square at each corner:

```
10 PLOT 0,0
20 INPUT A$
30 PLOT 0,43
40 INPUT A$
50 PLOT 63,0
60 INPUT A$
70 PLOT 63,43
```

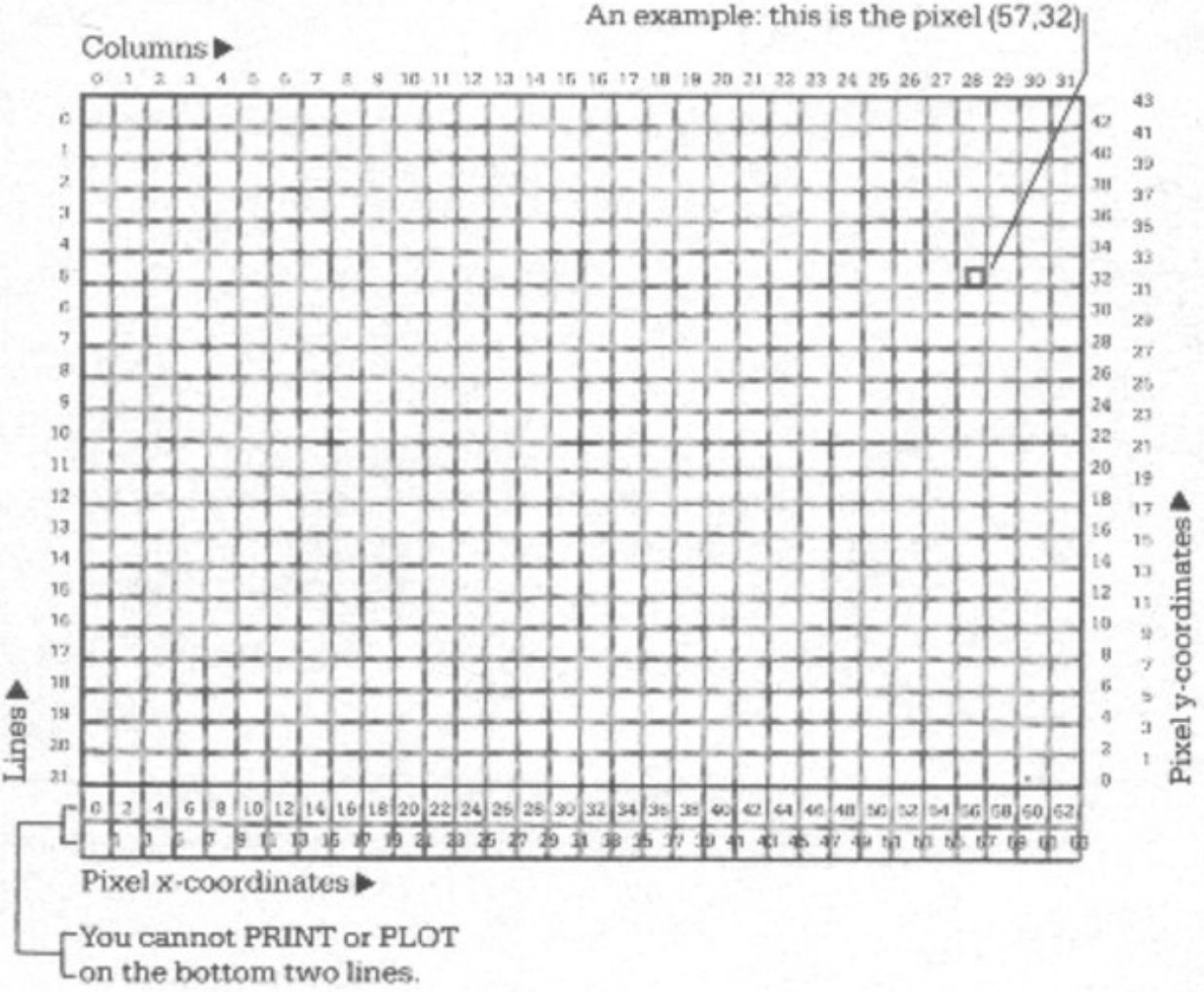
Rounding errors may give you trouble in the division subroutine. Can you use the correction routine in Chapter Ten to fix this?

After typing in the program, press **RUN** and **ENTER** and the first square will be located. The program will wait for an **INPUT** before plotting each of the other corners—but notice, it won't *do* anything with the input. We can use **INPUT** this way as a device to make the computer wait until we are ready to proceed. (You can just press **ENTER** in response to the **INPUT** prompt.)

You may recall we divided the screen into 32 characters while discussing the **TAB** and **PRINT AT** commands. It has 24 lines, but only 22 are available to **PRINT** on; the bottom two are reserved for writing program lines.

Multiplying those two numbers (32 and 22) by two gives you 64 and 44. . . and these are the dimensions of the **PLOT** screen (0-63 and 0-43). So you can **PLOT** twice as many points as you can **PRINT**, right?

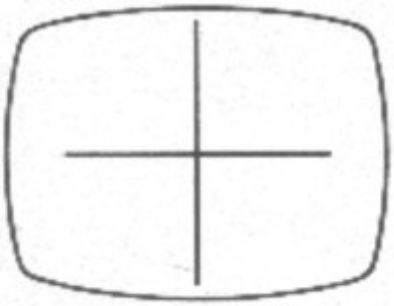
Wrong! As those of you who are paying attention can tell, you can **PLOT** *four times* as many locations. Here's a diagram to explain further:




```
10 REM PROGRAM: X/Y AXES
20 FOR X=0 TO 63
30 PLOT X,21
40 NEXT X
50 FOR Y=0 TO 43
60 PLOT 32,Y
70 NEXT Y
```

You can locate the X and Y axes on the screen with this little program:

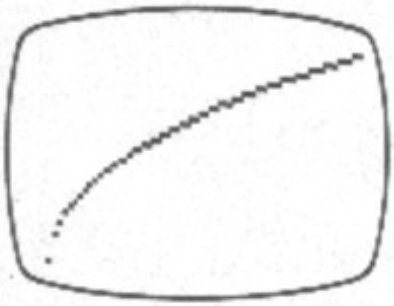
```
10 REM PROGRAM: X/Y AXES
20 FOR X = 0 TO 63
30 PLOT X,21
40 NEXT X
```



```
50 FOR Y = 0 TO 43
60 PLOT 32,Y
70 NEXT Y
```

And you can plot graphs of functions, such as this brief program that shows a part of a parabola as a graph of **SQR** between 0 and 4.

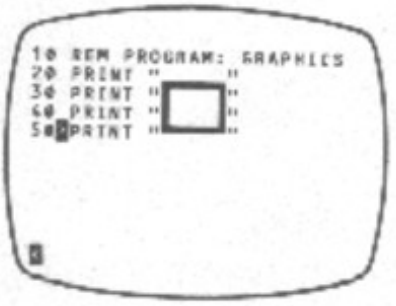
```
10 FOR N = 0 TO 63
20 PLOT N,20*SQR (N/16)
30 NEXT N
```



And remember, anything you can **PLOT**, you can **UNPLOT**. And by the way, **PLOT** and **UNPLOT**, like many other commands we've seen, move the print position to the **PLOT** or **UNPLOT** location.

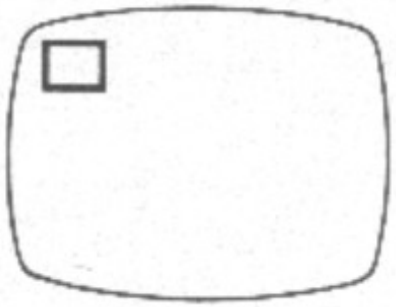
Using Keyboard Graphic Symbols

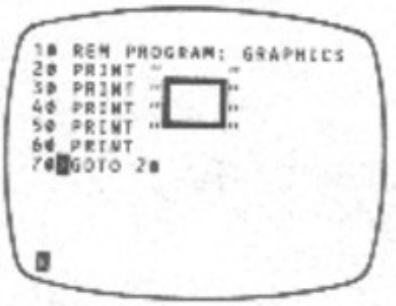
You can use the various keyboard graphic symbols, in combinations of **PRINT** statements, to draw figures on the screen. Try

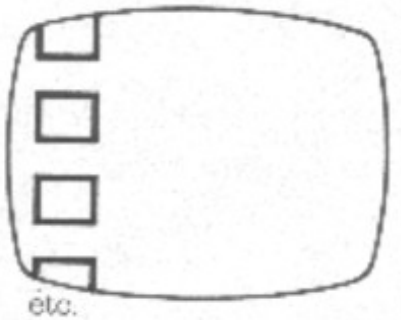



```
10 REM PROGRAM: GRAPHICS
20 PRINT "          "
30 PRINT "          "
40 PRINT "          "
50 PRINT "          "
```

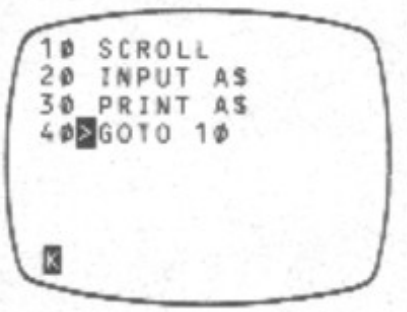
Hint: In line 20, after the first quotation marks, press **SHIFT** 9 to enter graphics mode. Press **SHIFT** 6 five times, then press **SHIFT** 9 again to exit graphics mode and close the quotes.







etc.



In line 30, use a **SHIFT** 8 for the left-hand graphic symbol, then get out of graphics mode and type in three spaces. Finally, go back into graphics mode and type a **SHIFT** 5 for the right symbol.

Line 40 repeats 30, and in line 50, you'd use the graphic on the 7 key.

You can also use graphic drawings more than once in a program. Add, to the above program, the lines

60 **PRINT**
70 **GOTO** 20

and **RUN** it.

What would it look like without line 60? How about if you **GOTO** 30 instead?

Can you use **TAB**, **PRINT AT** and the comma and semicolon to move the rectangle around the screen?

Try drawing other, more elaborate figures with the graphic symbols. Notice how they can be made to fit together (even to the small difference between the symbols on the G and H keys).

SCROLL

One more interesting thing you can do with the screen involves the **SCROLL** command. **SCROLL** moves the display up one line, causing the top line (when there is one) to disappear off the top of the screen, and it moves the print position to the bottom of the screen. Try this:

10 **SCROLL**
20 **INPUT** A$
30 **PRINT** A$
40 **GOTO** 10

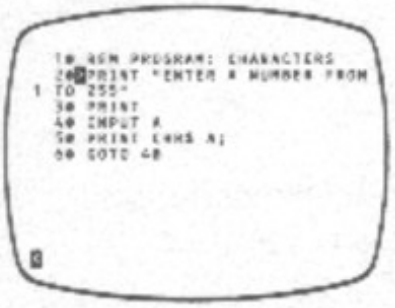
When the prompt appears, type in a letter, group of letters, or word and then press **ENTER**. Keep it

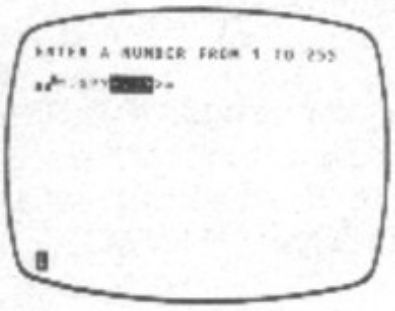
up until the display reaches the top of the screen and starts **SCROLL**ing off.

The **SCROLL** command can be used to keep a program from stopping because the screen is full.

This program sets up an endless loop. Can you stop it by pressing **STOP** when it asks for input?

You can if you **DELETE** the left quotation mark before the **L** cursor. Otherwise it just treats **STOP** as another string to be dealt with.





Other Wonders: CHR$ and CODE

We talked earlier about the T/S 1500's "extended alphabet," in which all its letters and numbers (and even keyword commands) are ranked in a 256-character listing.

Each character or keyword has a **CODE** number (a number between 0 and 255) and each of those code numbers has, of course, a character which we call a **CHR$** or *character string* (it is a very short character string . . .).

CHR$** (using the function **CHR$ located below the U key) applied to a number gives the single character string whose code is that number. Here's a program to find the character when you know the code:

```
10 REM PROGRAM: CHARACTERS
20 PRINT "ENTER A NUMBER
FROM 1 TO 255"
30 PRINT
40 INPUT A
50 PRINT CHR$ A;
60 GOTO 40
```

We used 1 to 255 in line 20. Code 0 is for the blank space, and we could just as well have included it.

Each time you input a number between 1 and 255, you are shown the corresponding character.

In many cases, you'll be shown a question mark; this is what the computer will put on the screen if it does not have a symbol it can print for that number.

Notice the semicolon at the end of line 50; feel free to change it if you like.

Here's one way to go the other way: input a character from the keyboard, and get its code. Use the function **CODE** located below the I key.

```
10 REM PROGRAM: CODES
20 PRINT "PRESS ANY KEY"
30 PRINT
40 INPUT A$
50 PRINT CODE A$
60 GOTO 40
```

In the sample run, we've obtained the code for, respectively,

Y
SHIFT Y
GRAPHICS Y
GRAPHICS SHIFT Y
FUNCTION Y

Summary

1. **PLOT** places a black square in a pixel (picture element) defined by two numbers separated by a comma: 0-63 from left to right across the screen and 0-43 from bottom to top.

   PLOT 63,43

2. **UNPLOT** uses the same coordinates to erase any previous **PLOT** marking.
3. Keyboard graphic symbols can be combined into charts and pictures.
4. **SCROLL** moves the display up one line (eliminating the top line, if any) and places the **PRINT** position on the bottom line.

5. **CHR$**, applied to a number between 0 and 255, returns the character whose code is that number.

```
PRINT CHR$ 140
```

6. **CODE**, applied to a single character string, returns the code number for that character.

```
PRINT CODE A$
```

Here are the 22 graphic symbols.

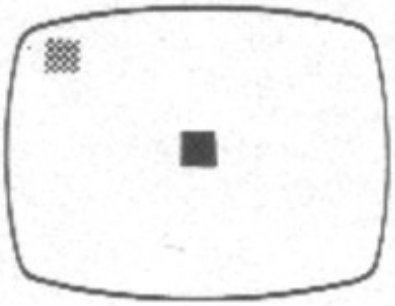
Symbol	How obtained	Code	Symbol	How obtained	Code
	K or L SPACE	0		G SPACE	128
	G shifted 1	1		G shifted Q	129
	G shifted 2	2		G shifted W	130
	G shifted 7	3		G shifted 6	131
	G shifted 4	4		G shifted R	132
	G shifted 5	5		G shifted 8	133
	G shifted T	6		G shifted Y	134
	G shifted E	7		G shifted 3	135
	G shifted A	8		G shifted H	136
	G shifted D	9		G shifted G	137
	G shifted S	10		G shifted F	138

Matters of Timing 13

Chapter Preview

This chapter shows you how to make things move with INKEY$, stop for a while with PAUSE, and how the computer works in FAST or SLOW mode.

```
10 REM PROGRAM TAG
20 LET X = 11
30 LET Y = 15
40 LET A = 0
50 LET B = 0
60 PRINT AT X,Y;"■"
70 PRINT AT A,B;"▒"
80 LET C = INT (RND*4) + 1
90 IF C = 1 THEN LET X = X + 3
100 IF C = 2 THEN LET X = X - 3
110 IF C = 3 THEN LET Y = Y + 3
120 IF C = 4 THEN LET Y = Y - 3
130 IF INKEY$ = "5" THEN LET B = B - 1
140 IF INKEY$ = "6" THEN LET A = A + 1
150 IF INKEY$ = "7" THEN LET A = A - 1
160 IF INKEY$ = "8" THEN LET B = B + 1
170 IF A < 0 OR B < 0 OR X < 0 OR Y < 0 OR A > 21
    OR B > 31 OR X > 21 OR Y > 31 THEN STOP
180 IF A = X AND B = Y THEN STOP
190 CLS
200 GOTO 60
```



The command **INKEY$** is like **INPUT** except that it does not wait for you. **INKEY$** scans the keyboard to see which key is being pressed—if any—and takes action according to the program.

The above program is a game of tag. Lines 20-70 place a black square in the center of the screen (to get a black square, press the **SPACE** key while in graphics mode—the black square is the inverse of the white space) and put you in control of a grey square (graphic H) in the upper left hand corner.

Lines 80-120 use a random number generator to move the black square, three spaces at a time in any direction.

Lines 130-160 move the grey square, according to your direction. You can only move one space at a time, but presumably you are moving purposefully.

Line 170 stops the program (and you lose) if the black square escapes to the edge of the screen—or if your grey square falls off the edge!

Line 180 stops the program (and you win) if you "tag" the black square with your grey one; that is, if their coordinates are the same.

When you play this game, remember that **INKEY$** checks the keyboard to see what key is being pressed when that program line comes by. Rather than tapping the keys, you ought to hold down the key that corresponds to the direction you wish to move to close in on the black square.

Notice, also, that you use the four cursor arrow keys to move in the direction of the arrow, but you don't have to press **SHIFT**.

If you are having trouble winning, try changing lines 40 and 50 so you start out closer to the black square. .

Here's a short program to show that **INKEY$** waits for no one. If you don't type a character (press any key) very quickly, the computer prints a blank space and scrolls merrily onward and upward. Use the **BREAK** key when you want to stop it.

10 **SCROLL**
20 **PRINT INKEY$**
30 **GOTO** 10

PAUSE

The **PAUSE** command does just what you'd expect it to, and you can set it using a numerical value.

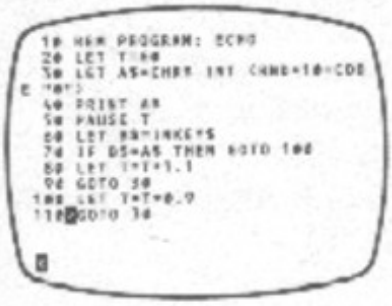
PAUSE 60 is about one second. More is a higher number and less is lower. Add to the above program:

25 **PAUSE** 60

and it will be a lot easier to keep up with the scrolling.

Here is a diabolical program that changes the length of **PAUSE** in response to your success at the game! Your task is to "echo" the computer's output on the screen. When the computer prints a random digit between 0 and 9, you have to type that digit before the computer goes on to the next one.

The good news is that, if you miss some, the computer will slow down for you. The bad news is that, if you get some right, the pace will speed up.



```
10 REM PROGRAM: ECHO
20 LET T=60
30 LET A$=CHR$ INT (RND*10+CODE
"0")
40 PRINT A$
50 PAUSE T
60 LET B$=INKEY$
70 IF B$=A$ THEN GOTO 100
80 LET T=T*1.1
90 GOTO 30
100 LET T=T*0.9
110 GOTO 30
```

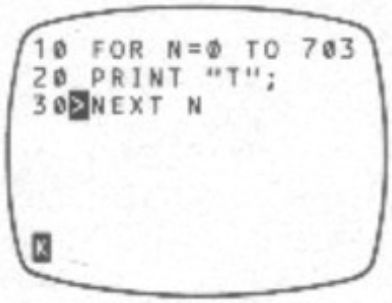
SLOW and FAST

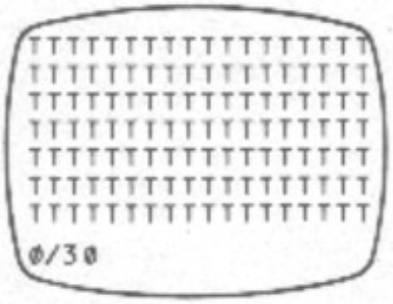
The Timex Sinclair 1500 has two speeds, **SLOW** and **FAST**. Normally it runs in the **SLOW** mode, which is fast enough for most purposes and has the advantage of maintaining the screen for the use of graphics.

What happens in the **SLOW** mode is that the computer maintains the screen display and does its computing between "screen refreshes"—during the period when the television is producing the black line you see between frames when the vertical hold slips.

When you have lengthy computation to be done, you may want to slip into **FAST** mode. Roughly four times as fast, this mode computes first and puts a picture on the screen only when it is not otherwise occupied. This results in periods of grey screen during long calculations, and even brief flickers when entering characters from the keyboard.

You can put the commands **SLOW** and **FAST** into programs, directing the T/S 1500 to use the most appropriate mode for what it has to do. You can also press **FAST** outside of a program and the computer will stay in **FAST** mode until you press **SLOW**.





Try typing in this program:

```
10 FOR N = 0 TO 703
20 PRINT "T";
30 NEXT N
```

Press **SLOW** (**SHIFT** D) just to make sure you are in that mode, then **RUN** the program and time it. Then press **FAST** (**SHIFT** F) and **RUN** and time the program again. Pretty impressive, eh?

Summary

1. **INKEY$** reads the keyboard and inputs as a character string any key being pressed. If no key is being pressed at the time, it "reads" the empty or "null string." **PRINT INKEY$** will then act like **PRINT** alone; if no key is being pressed, the computer will print a blank line.
2. **PAUSE** causes a program to wait a specified length of time (60 = one second) before continuing to the next line. If a key is pressed, the pause ends.
3. **SLOW**, the mode in which the computer normally runs, gives priority to maintaining the screen display.
4. **FAST**, which can be obtained by a program statement or a separate command, gives priority to computation and runs about four times as fast as **SLOW**.

Arrays

14

Chapter Preview

Organize your data and save space with "arrays," using DIM, subscripts, and string slicing.

An *array* is a way of structuring a number of values and keeping track of them. Each of the values is called an *element* of the array. You can think of an array as looking like a calendar:

SUN	MON	TUES	WED	THU	FRI	SAT
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

The array

An element

You can assign numeric values to *elements* of an *array*, instead of giving each value a separate variable name. This can be used simply to save space, or when there is a relationship between or among the values.

Suppose you have a row of numbers:

12 5 7 22 14

You could assign the values to separate variables

```
LET A = 12
LET B = 5
LET C = 7
LET D = 22
LET E = 14
```

Or, you can consider the whole row to be an array:

```
LET A(1) = 12
LET A(2) = 5
LET A(3) = 7
LET A(4) = 22
LET A(5) = 14
```

The number in parentheses is called the *subscript*.

Array A(5)

Picture the array this way:

(1)	(2)	(3)	(4)	(5)
12	5	7	22	14

DIM . . . The Dimension Statement

Before you can assign values to each element in the array, you have to reserve space for the array with a *dimension* statement (**DIM**). To prepare for the array above, you would have to enter

```
DIM A(5)
```

There are a few rules regarding array variables:

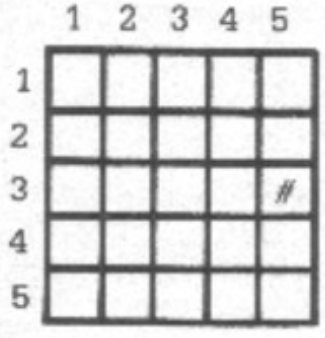
1. An array variable name must be a single letter (like a control variable in a **FOR/NEXT** loop).
2. An array variable name can be the same as the name of a simple variable (there can be a simple variable named A at the same time as an array named A; you can tell them apart because elements of the array variable are always referred to with the subscript).
3. A **LET** statement erases a previous simple variable by the same name. Similarly, when you create an array with the **DIM** statement, you delete any previous array of the same name. But while you can build a new simple variable from an old one (as in **LET** A = A + 1), you would simply lose an old array by **DIM**ensioning a new one by the same name.

Arrays in More Than One Dimension

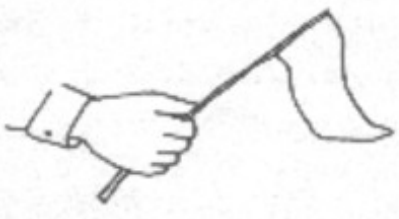
You can have arrays of as many dimensions as you care to try to keep track of, as long as you dimension them properly at the outset:

DIM A(5,5)

would set up an array you can think of as looking like this.



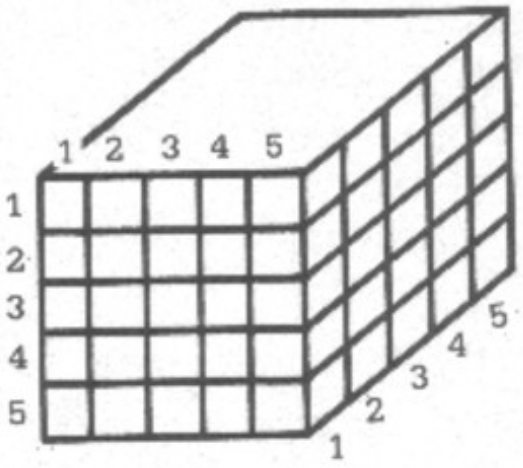
You can think of the first number as identifying the row and the second, the column. Therefore, location # in the diagram is A(3,5).



What value occupies the location (3,5) in our calendar?

You can have an array in three dimensions; think of it as looking like this:

DIM A(5,5,5)



And you can have arrays in four or more dimensions, but don't ask us to help you picture them.

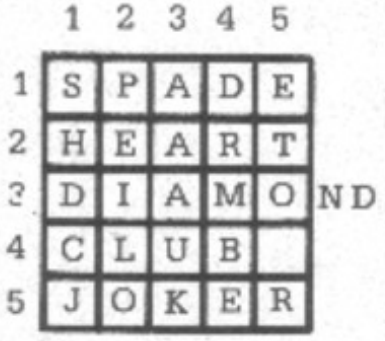
String Arrays

You can assign strings to arrays, as

DIM A$(5,5)

but there are a few rules here, too.

1. When you DIMension the string array, you delete any previous string arrays *and* any previous simple string variables *that have the same variable name.*
2. In two dimensions, you can think of the first number as being the identifier of each string, or "word," and the second as the number of letters in each word.
3. Assignment to the string variable elements is Procrustean, which means that the strings are filled in from character #1 up through a number of characters equal to the second subscript, and
   a. If the string is too long, it will be truncated from the end.
   b. If the string is too short, the space will be filled in with blanks.



Array A$(5,5)

You access the strings by using the first subscript only, and individual letters by using the second subscript as well.

PRINT A$(3) returns DIAMO

(the ND are truncated), and

PRINT A$(4) returns CLUB

(including a blank space after the B). Also,

PRINT A$(3,2) returns I

PRINT A$(4,5) returns the blank space

You can also use *string slicing* (more about that in a moment) to obtain a portion of a string, as

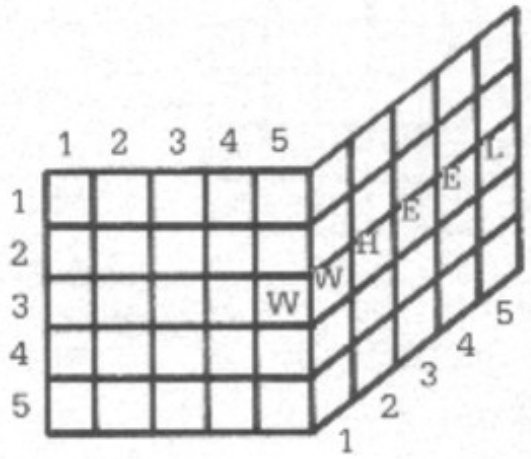
PRINT A$(4,2 **TO** 4) returns LUB

PRINT A$(3, **TO** 3) returns DIA

String Arrays in Three of More Dimensions

You can have string variable arrays in as many dimensions as you like. However many numbers (dimensions) are separated by commas, the *last number* identifies the number of characters in each string, and the other numbers serve to specify the string by its location. In three dimensions:

PRINT A$(3,5,5) would return WHEEL in the array diagrammed below.



Again, it is not easy to picture string arrays in more than three dimensions, but just remember that the last subscript specifies the number of characters in each string in the array, while the other subscripts locate the string.

Slicing Strings

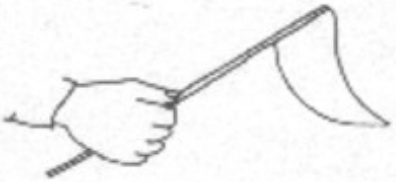
Running the program

```
10 LET A$ = "ONES AND TWOS"
20 PRINT A$
30 PRINT A$( )
40 PRINT A$(6)
50 PRINT A$(1 TO 4)
60 PRINT A$(TO 4)
70 PRINT A$(10 TO 13)
80 PRINT A$(10 TO)
```

```
ONES AND TWOS
ONES AND TWOS
A
ONES
ONES
TWOS
TWOS
```

demonstrates string slicing. Line 20 prints the string. Line 30 shows the empty brackets after the string variable name means that the entire string is its own *substring*.

Line 40 selects, as a substring, only one character —the sixth (don't forget to count the spaces in the string).



Line 50 prints characters 1 to 4 of the string; line 60 shows that you can omit the first digit and the first character is implied. Line 70 shows how to print the last four characters, and line 80 shows that you can omit the last number and "last character" is implied.

How would you print "AND" out of that string?

Now, we can use string slicing to save space in a program by assigning a long string to a variable name and then cutting pieces out of it, rather than assigning a lot of variables. (If you use this technique with numbers and want to perform mathematical operations with them, you need to use the **VAL** function. See Appendix A.)

Summary

1. An array is made up of a number of elements, all with the same array variable name, and distinguished from each other by means of subscripts.
2. The name of a numeric array must be a single letter; there can also be a simple variable using the same letter as a name.
3. The name of a string variable must be a single letter followed by $; there cannot be a simple string variable with the same name in the computer's memory.
4. Before assigning values to the elements of an array, you must reserve space for it in the computer with the **DIM** statement.
5. You can "slice" strings and obtain substrings by using the **TO** statement.

Using the Printer 15

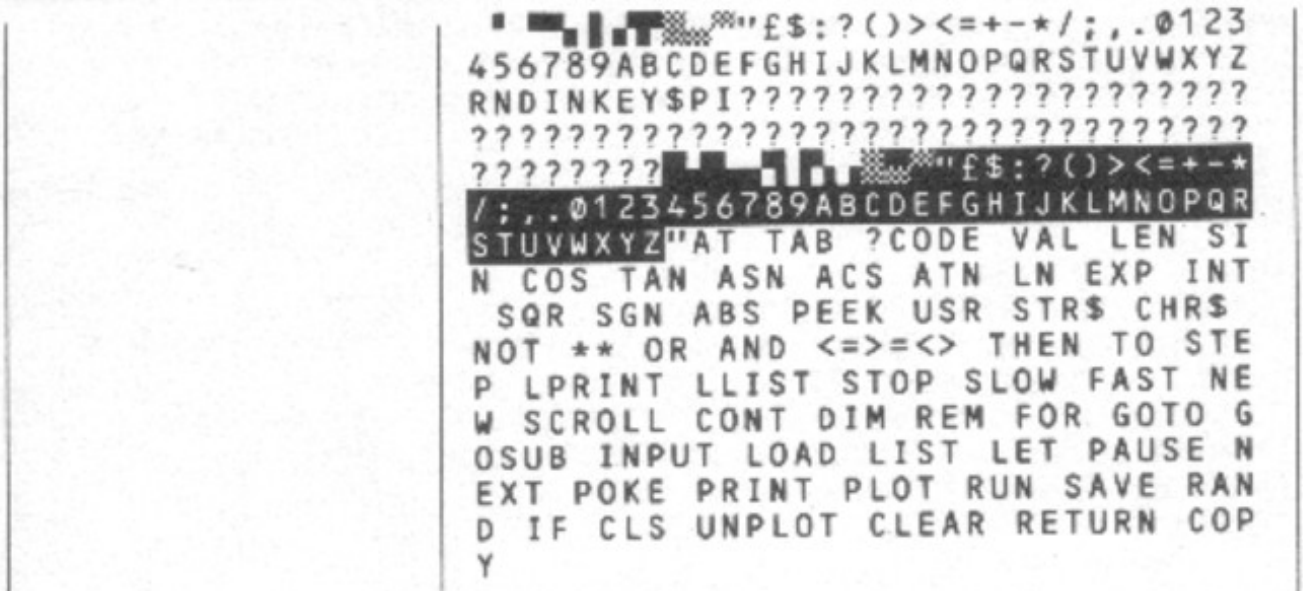
Chapter Preview

You can get programs or output on paper with LPRINT, LLIST, and COPY if you have a printer.

```
10 REM PROGRAM: PRINTER
20 LPRINT "THIS PROGRAM"...
30 LLIST
40 LPRINT
50 LPRINT "PRINTS OUT THE
   CHARACTER SET."...
60 FOR N = 0 TO 255
70 LPRINT CHR$ N;
80 NEXT N
```

You can obtain copies of your programs, and their results, on paper, by attaching a printer to your Timex Sinclair 1500. This is called a "hard copy" because it will be around for a while, as opposed to what you see on your screen.

Run the program above, with the Timex printer attached. Notice that it prints a listing of itself, and the T/S 1500's character set.



The Timex printer is an inexpensive device that attaches to the back of your 1500 and is then operated by just three simple commands.

LPRINT

LPRINT (**SHIFT** S), is the same as **PRINT**, except that the material to be printed is sent to the printer instead of the screen.

The "L" originally stood for "line printer," although the command is now used for any kind of "hard copy" printer. When BASIC was invented, the usual display was a kind of electric typewriter rather than a TV screen, so **PRINT** really did mean print. If you wanted a lot of output fast, a line printer would turn it out more quickly than a character-at-a-time typewriter.

LLIST

In the same way, **LLIST** (**SHIFT** G), lists the program currently in the computer's memory on the printer instead of the screen.

LLIST can be used to "pull a listing" without putting a program line in front of the command. If you add a program line—as in **LLIST** 9Ø—the listing will be printed starting with that line. In other

words, if you know you want a copy of the program the computer has in it, just press **LLIST** and **ENTER**, and you'll get it.

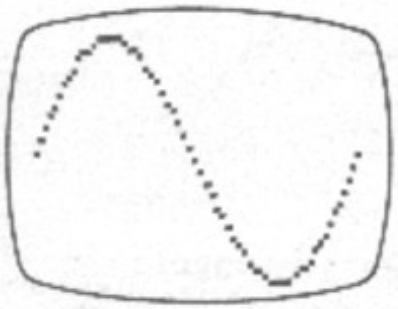
Or, **LLIST** can be used within a program like the one above—now is a good time to try it out, if you haven't already.

(**LPRINT**, of course, can also be used either way. Usually, you'll use **LPRINT** within a program—it needs to **LPRINT** "SOMETHING"—while you'll more often use **LLIST** outside the program, just to get a copy of it to refer to.)

COPY

The third printer command is **COPY** (keyword over Z). This simply makes a copy, on the printer, of whatever is on the screen when **COPY** is pressed. You can use **COPY** as a program line or as a separate command anytime you want to copy the screen.

Try this program, which graphs the function **SIN** (resulting in a *sine wave*.)



```
10 REM PROGRAM: SINEWAVE
20 FOR N = 0 TO 63
30 PLOT N,22 + 20*SIN (N/32*PI)
40 NEXT N
50 COPY
```

(In line 30, **PI** is not spelled out—use the function π under the M key.)

Questions:

Will you get the same effect by eliminating line 50, and then pressing **COPY** and **ENTER** after the program has stopped running?

Why do you use the range 0 to 63 for N in line 20?

Will you get the same effect by pressing **LLIST** and **ENTER** as you get by using **ENTER** to get the program listing on the screen and then pressing **COPY** and **ENTER**?

Stopping the Printer with BREAK

When the printer is running, you can stop it with **BREAK**. You might want to save paper if, for example, you are using **COPY** to print out the five line program listing above. The command will run out all 24 lines' worth of paper, unless you press **BREAK** when you see line 50 appear.

If you try to execute any of the three printer commands without a printer attached, usually the program will just proceed to the next line without printing anything. Sometimes, however, the computer will get hung up and you'll need to press **BREAK** to rescue it.

Print Format Statements with the Printer

All but one of the screen printing format commands will work with **LPRINT**. The comma, semicolon, and **TAB** can be used, but **AT** does not work. To illustrate, try this:

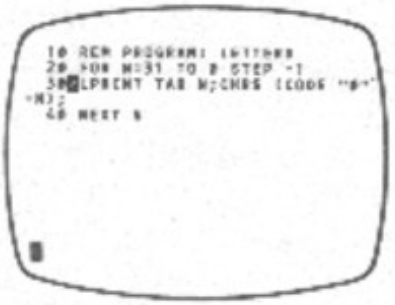
```
10 REM PROGRAM: LETTERS
20 FOR N=31 TO 0 STEP -1
30 PRINT AT 31-N,N;CHR$ (CODE
"0"+N);
40 NEXT N
```

```
10 REM PROGRAM: LETTERS
20 FOR N = 31 TO 0 STEP - 1
30 PRINT AT 31 - N,N;CHR$(CODE "0" + N);
40 NEXT N
```

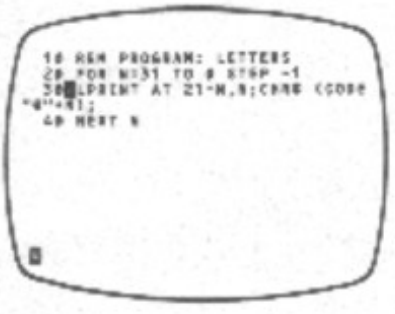
RUN the program. You'll see a diagonal row of letters working its way down the screen, until it stops with report code 5: the screen is full.

Next, change **AT** 31 – N,N in line 30 to **TAB** N. **RUN** it again and you'll get the same effect.

Be patient, we'll get to the point.



Now, change **PRINT** in line 30 to **LPRINT**. The program will **RUN**, and the pattern will continue for ten more lines since the printer can't be "full" as the screen can. You'll get no report code 5.



Finally, change **TAB** N to **AT** 21 – N,N. **RUN** the program one more time.

This time the printer shows a single row of characters! This is simply because **AT** does not send a "line feed" to the printer. It will move the print position over a column, but not down a line.

The printer will print a line

1. After an **LPRINT** statement that does not end in a comma or semicolon.
2. When a comma or **TAB** statement requires a new line to be started.
3. At the end of a program, if there is anything "left over" to be printed.
4. Any time the "buffer" is full. The buffer is the area where characters to be printed are stored until they are printed. The buffer is exactly one line (32 characters) long, so unless one of the events above (1, 2 or 3) occurs first, the printer will print a line when a full line is in the buffer (among other things, it needs to empty the buffer before new characters can be stored there).

Remember how the printer, in the first program of the chapter, seemed to hesitate before printing each line of the character set? It was filling the buffer to a full line during those pauses.

Summary

1. **LPRINT** prints on the printer just like **PRINT** prints on the screen.
2. **LLIST** sends a program listing to the printer just as **LIST** sends it to the screen.
3. **COPY** duplicates on the printer whatever is showing on the screen.
4. **BREAK** stops the printer when it is running, or interrupts printer commands when they cause a problem.
5. **TAB**, comma and semicolon can be used to format **LPRINT** statements, but **AT** does not.

Appendix A: Summary of T/S 1500 BASIC

General

If you already know BASIC then you should not have much trouble using the T/S 1500.

(i) Words are not spelled out, but have keys of their own—this is described in Chapter 2. In the text, these words are printed in **BOLD TYPE**.

(ii) Sinclair BASIC lacks READ, DATA and RESTORE (but see Chapter 11 concerning this), user-defined functions (FN and DEF; but **VAL** can sometimes be used), and multi-statement lines.

(iii) The string handling facilities are comprehensive.

(iv) The T/S 1500 character set is completely its own.

(v) The television display is not, in general, memory-mapped.

(vi) If you are accustomed to using **PEEK** and **POKE** on a different machine, remember that all the addresses will be different on the T/S 1500.

Appendix A: Summary of T/S 1500 BASIC

Speed

The machine works at two speeds, called *compute and display* mode and *fast* mode.

In compute and display, the TV screen is generated continuously, and computing is done during the blank parts at the top and bottom of the picture.

In fast mode the television picture is turned off during computing and is displayed only at the end of a program, while waiting for **INPUT** data, or during a pause (see **PAUSE**).

Fast mode runs about four times as fast and should be used for programs with a lot of computing as opposed to output, or when typing in long programs.

Switching between speeds is done with the **FAST** and **SLOW** statements (q.v.).

The keyboard

T/S 1500 characters comprise not only the single *symbols* (letters, digits, etc.), but also the compound *tokens* (keywords, function names, etc.; these are printed here in **BOLD TYPE**), and all are entered from the keyboard rather than being spelled out. To fit this in, some keys have up to five distinct meanings, given partly by shifting the keys (i.e., pressing the **SHIFT** key at the same time as the required one) and partly by having the machine in different modes.

The mode is indicated by the *cursor*, an inverse video letter that shows where the next character from the keyboard will be inserted.

K mode (for keywords) occurs automatically when the machine is expecting a command or program line (rather than **INPUT** data), and from this position on the line it knows it should expect a line number or a keyword. This is at the beginning of the line, or just after some digits at the beginning of the line, or just after **THEN**. If unshifted, the next key will be interpreted as either a keyword (these are written *above* the keys) or a digit.

L mode (for letters) normally occurs at all other times. If unshifted, the next key will be interpreted as the main symbol on that key.

Appendix A: Summary of T/S 1500 BASIC

In both K and L modes, a shifted key will be interpreted as the subsidiary black character in the top right-hand corner of the key.

F mode (for functions) occurs after **FUNCTION** (shifted **ENTER**) is pressed and lasts for one key depression only. That key will be interpreted as a function name, which appears *under* the keys.

G mode for graphics occurs after **GRAPHICS** (shifted 9) is pressed and lasts until it is pressed again. An unshifted key will give the inverse video of its L mode interpretation. A shifted key will as well, provided that it is a symbol; but if the shifted key would normally give a token, in graphics mode it will give the graphics symbol that appears in the bottom right-hand corner of the key.

The screen

This has 24 lines, each 32 characters long, and is divided into two parts. The top part is at most 22 lines, and displays either a listing or program output. The bottom part, at least two lines, is used for inputting commands, program lines and **INPUT** data, and also for displaying reports.

Keyboard input: this appears in the bottom lines of the screen as it is typed, each character (single symbol or compound token) being inserted just before the cursor. The cursor can be moved left with ◀ (shifted 5) or right with ▶ (shifted 8). The character before the cursor can be removed with **DELETE** (shifted 0) (Note: the whole line can be deleted by typing **EDIT** (shifted 1) followed by **ENTER**.)

When **ENTER** is pressed, the line is executed, entered into the program, or used as **INPUT** data as appropriate, unless it contains a syntax error, in which case the symbol S appears just before the error.

As the program lines are entered, a listing is displayed in the top half of the screen. The last line to be entered is called the *current line* and is indicated by the symbol >, but this can be changed by using the keys ▼ (shifted 6) and ▲ (shifted 7). If **EDIT** (shifted 1) is pressed, the current line is brought down to the bottom part of the screen and can be edited.

When a command is executed or a program run, output is displayed in the top half of the screen and

Appendix A: Summary of T/S 1500 BASIC

remains until a program line is entered, or **ENTER** is pressed with an empty line, or ▼ or ▲ is pressed. In the bottom part appears a report of the form m/n, where m is a code showing what happened (see Report Codes), and n is the number of the last line executed—or 0 for a command. The report remains until a key is pressed (and indicates **K** mode).

In certain circumstances, the **SPACE** key acts as a **BREAK**, stopping the computer with report D. This is recognized

(i) at the end of a statement while a program is running,
(ii) while the computer is looking for a program on tape, or
(iii) while the computer is using the printer (or by accident trying to use it when it is not there).

The BASIC

Numbers are stored to an accuracy of 9 or 10 digits. The largest number you can get is about 10^{38}, and the smallest (positive) number is about $4*10^{-39}$.

A number is stored in the T/S 1500 in floating-point binary with one exponent e($1 <= e <= 255$), and four mantissa bytes m($1/2 <= m < 1$). This represents the number $m*2^{e-128}$.

Since $1/2 <= m < 1$, the most significant bit of the mantissa m is always 1. Therefore in actual fact we can replace it with a bit to show the sign: 0 for positive numbers, 1 for negative.

Zero has a special representation in which all 5 bytes are 0.

Numeric variables have names of arbitrary length, starting with a letter and continuing with letters and digits. Spaces are ignored.

Control variables of **FOR-NEXT** loops have names a single letter long.

Numeric arrays have names a single letter long, which may be the same as the name of a simple variable. They may have arbitrarily many dimensions of arbitrary size. Subscripts start at 1.

Strings are completely flexible in length. The name of a string consists of a single letter followed by $.

String arrays can have arbitrarily many dimensions of arbitrary size. The name is a single letter followed by $ and may not be the same as the name of a string. All the strings in a given array have the same fixed length, which is specified as an extra, final dimension in the **DIM** statement. Subscripts start at 1.

Slicing: Substrings of strings may be specified by using *slicers*. A slicer can be

(i) empty or

(ii) a numerical expression or

(iii) an optional numerical expression **TO** optional numerical expression and is used in expressing a substring either by

(a) string expression (slicer)
or by

(b) string array variable (subscript, ..., subscript, slicer)
which means the same as
string array variable (subscript, ..., subscript)(slicer)

In (a), suppose the string expression has the value s$. If the slicer is empty, the result is s$ considered as a substring of itself. If the slicer is a numerical expression with value m, the result is the mth character of s$ (a substring of length 1).

If the slicer has the form (iii), suppose the first numerical expression has the value m (the default value is 1), and the second, n (the default value is the length of s$).

If 1<=m<=n<= the length of s$, the result is the substring of s$, starting with the mth character and ending with the nth.

If 0<=n<m, the result is the empty string. Otherwise, error 3 results.

Slicing is performed before functions or operations are evaluated, unless brackets dictate otherwise.

Substrings can be assigned to (see **LET**).

The argument of a function does not need brackets if it is a constant or a (possibly subscripted or sliced) variable.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
	(x)	
ABS	number	Absolute magnitude.
ACS	number	Arccosine in radians. Error A if x not in the range −1 to +1.
AND	binary operation, right operand always a number.	
	Numeric left operand:	A **AND** B = {A if B≠0 {0 if B = 0
	String left operand:	A$ **AND** B = {A$ if B≠0 {"" if B = 0
ASN	number	Arcsine in radians. Error A if x not in the range −1 to +1
ATN	number	Arctangent in radians.
CHR$	number	The character whose code is x, rounded down to the nearest integer. Error B if not in the range 0 to 255.
CODE	string	The code of the first character in x (or 0 if x is empty string.)
COS	number (in radians)	Cosine.
EXP	number.	e^x.
INKEY$	none	Reads the keyboard. The result is the character representing (in L mode) the key pressed if there is exactly one, else the empty string.
INT	number	Integer part (always rounds down).
LEN	string	Length.
LN	number	natural logarithm (to base e). Error A if x<−0.
NOT	number	0 if x ≠ 0, 1 if x = 0. NOT has priority 4.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
OR	binary operation, both operands numbers	A OR B = {1 if B≠0; A if B = 0. OR has priority 2.
PEEK	number	The value of the byte in memory whose address is x (rounded to the nearest integer). Error B if x not in the range 0 to 65535.
PI	none	π (3.14159265 . .)
RND	none	The next pseudo-random number y in a sequence generated by taking the powers of 75 modulo 65537, subtracting 1 and dividing by 65536. 0<= y<1.
SGN	number	Signum: the sign (– 1, 0 or + 1) of x.
SIN	number (in radians)	Sine.
SQR	number	Square root. Error B if x <0.
STR$	number	The string of characters that would be displayed if x were printed.
TAN	number (in radians)	Tangent.
USR	number	Calls the machine code subroutine whose starting address is x. On return, the result is the contents of the bc register pair.
VAL	string	Evaluates x (without its bounding quotes) as a numerical expression. Error C if x contains a syntax error, or gives a string value. Other errors possible, depending on the expression.
—	number	Negation.
		The following are binary operations:
+		Addition (on numbers), or concatenation (on strings)
–		Subtraction
*****		Multiplication

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
/		Division
**		Raising to a power. Error B if left operand is negative.
=		Equals
>		Greater than
<		Less than
<=		Less than or equal to
>=		Greater than or equal to
<>		Not equal to

Both operands must be of the same type. The result is a number, 1 if the comparison holds, and 0 if it does not.

Functions and operations have the following priorities:

Operation	*Priority*
Subscribing and slicing	12
All functions except **NOT** and unary minus	11
**	10
Unary minus	9
*,/	8
+, – (binary –)	6
=,>,<,<=,>=,<>	5
NOT	4
AND	3
OR	2

Statements

In this list,

@	represents a single letter
v	represents a variable
x,y,z	represents numerical expressions
m,n	represents numerical expressions that are rounded to the nearest integer
e	represents an expression
f	represents a string-valued expression
s	represents a statement

Note that arbitrary expressions are allowed everywhere (except for the line number at the beginning of a statement).

All statements except **INPUT** can be used either as commands or in programs (although they may be more sensible in one than the other).

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
CLEAR		Deletes all variables, freeing the space they occupied.
CLS		(CLear Screen) Clears the display file. See Chapter 26 concerning the display file.
CONT		Suppose a/b were the last report with a non-zero. Then **CONT** has the effect GOTO b if a ≠ 9 GOTO B + 1 if a = 9 (**STOP** statement)
COPY		Sends a copy of the display to the printer, if attached; otherwise does nothing. Report D if **BREAK** pressed.
DIM@ (n_1,...,n_k)		Deletes any array with the name @ and sets up an array of numbers with k dimensions n_1,...,n_k. Initializes all the values to 0. Error 4 occurs if there is no room to fit the array in. An array is undefined until it is dimensioned in a **DIM** statement.
DIM@ $ (n_1,...,n_k)		Deletes any array or string with the name @ $ and sets up an array of characters with k dimensions n_1, ..., n_k. Initializes all the values to "". This can be considered as an array of strings of fixed length n_k, with k – 1 dimensions n_1, ..., n_{k-1}. Error 4 occurs if there is no room to fit the array in. An array is undefined until it is dimensioned in a **DIM** statement.
FAST		Starts fast mode, in which the display file is displayed only at the end of the program, while **INPUT** data is being typed in, or during a pause.
FOR@ = x TO y		FOR@ = x TO y STEP 1
FOR@ = x TO y STEP z		Deletes any simple variable and sets up a control variable with value x, limit y, step z, and looping address 1 more than the line number of the **FOR** statement (– 1 if it is a command). Checks if the initial value is greater (if step > = 0) or less (if step <0) than the limit, and if so, then skips to statement **NEXT@** at the beginning of a line. See **NEXT@**. Error 4 occurs if there is no room for the control variable.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
GOSUB n		Pushes the number of the GOSUB statement onto a stack; then as GOTO n. Error 4 can occur if there are not enough **RETURN**s.
GOTO n		Jumps to line n (or, if there is none, to the first line after that).
IF x THEN s		If x is true (non-zero), then s is executed. The form '**IF** x **THEN** line number' is not allowed.
INPUT v		Stops (with no special prompt) and waits for the user to type in an expression; the value of this is assigned to v. In fast mode, the display file is displayed. **INPUT** cannot be used as a command; error 8 occurs if you try. If the first character in the **INPUT** line is **STOP**, the program stops with report D.
LET v = e		Assigns the value of e to the variable v. **LET** cannot be omitted. A simple variable is undefined until it is assigned to in a **LET** or **INPUT** statement. If v is a subscripted string variable, or a sliced string variable (substring), then the assignment is *Procrustrean*: the string value of e is either truncated or filled out with spaces to the right, to make it the same length as the variable v.
LIST		LIST 0
LIST n		Lists the program on the TV screen, starting at line n, and makes n the current line. Error 4 or 5 if the listing is too long to fit on the screen; **CONT** will do exactly the same again.
LLIST		LLIST 0
LLIST n		Like **LIST**, but using the printer instead of the television. Should do nothing if the printer is not attached. Stops with Report D if **BREAK** is pressed.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
LOAD f		Looks for a program called f on tape and loads it and its variables. If f = "", then loads the first program available. If **BREAK** is pressed, then (i) if no program has yet been read in from tape, stops with report D and old program; (ii) if part of a program has been read in, then executes **NEW**.
LPRINT . . .		Like **PRINT**, but using the printer instead of the television. A line of text is sent to the printer (i) when printing spills over from one line to the next, (ii) after an **LPRINT** statement that does not end in a comma or semicolon, (iii) when a comma or **TAB** item requires a new line, or (iv) at the end of the program, if there is anything left unprinted. In an **AT** item, only the column number has any effect; the line number is ignored. An **AT** item never sends a line of text to the printer. There should be no effect if the printer is absent. Stops with report D if **BREAK** is pressed.
NEW		Restarts the BASIC system, deleting program and variables and using the memory up to but not including the byte whose address is in the system variable RAMTOP (bytes 16388 and 16389).
NEXT @		(i) Finds the control variable @. (ii) Adds its step to its value. (iii) If the step $>= 0$ and the value $>$ the limit; or if the step < 0 and the value $<$ the limit, then jumps to the looping line. Error 1 if there is no control variable @.
PAUSE n		Stops computing and displays the display file for n frames (at 50 frames per second) or until a key is pressed. $0 <= n <= 65535$, else error B. If $n >= 32767$, then the pause is not timed, but lasts until a key is pressed.
PLOT m,n		Blacks in the pixel ($\|m\|$, $\|n\|$); moves the **PRINT** position to just after the pixel. $0 <= \|m\| <= 63$, $0 <= \|n\| <= 43$, else error B.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
POKE m,n		Writes the value n to the byte in store with address m. 0 <= m <= 65535, −255 <= n <= 255, else error B.
PRINT . . .		The '. . .' is a sequence of **PRINT** items, separated by commas or semicolons. They are written to the display file for display on the television. The position (line and column) where the next character is to be printed is called the **PRINT** position. A **PRINT** item can be (i) empty, i.e., nothing (ii) a numerical expression First, a minus sign is printed if the value is negative. Now let x be the modulus of the value. If $x <= 10^{-5}$ or $x >= 10^{13}$, then it is printed using scientific notation. The mantissa part has up to eight digits (with no trailing zeros), and the decimal point (absent if only one digit) is after the first. The exponent part is E, followed by + or −, followed by one or two digits. Otherwise x is printed in ordinary decimal notation with up to eight significant digits, and no trailing zeros after the decimal point. A decimal point right at the beginning is always followed by a zero, so, for instance, .03 and 0.3 are printed as such. 0 is printed as a single digit 0. (iii) a string expression. The tokens in the string are expanded, possibly with a space before or after. The quote image character prints as ". Unused characters and control characters print as ?. (iv) **AT** m,n The **PRINT** position is changed to line m (counting from the top), column n (counting from the left). 0 <= |m| <= 21, 0 <= |n| <= 31, else error B. If |m| = 22 or 23, error 5. (v) **TAB** n n is reduced modulo 32. Then, the **PRINT** position is moved to column n, staying on the same line unless this would involve backspacing, in which case it moves on to the next line. 0 <= n <= 255, else error B.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
		A semicolon between two items leaves the **PRINT** position unchanged, so that the second item follows immediately after the first. A comma, on the other hand, moves the **PRINT** position on at least one place; and after that, as many as are necessary to leave it in column 0 or 16, moving to a new line if necessary. At the end of the **PRINT** statement, if it does not end in a semicolon or comma, a new line is started. Error 4 (out of memory) can occur with 3K or less of memory. Error 5 means that the screen is filled. In both cases, the cure is **CONT**, which will clear the screen and allow the program to continue.
RAND		RAND 0
RAND n		Sets the system variable (called SEED) used to generate the next value of **RND**. If n ≠ 0, the SEED is given the value n; if n = 0, it is given the value of another system variable (called FRAMES) that counts the frames so far displayed on the television, and so should be fairly random. Error B occurs if n is not in the range 0 to 65535.
REM . . .		No effect. ' . . . ' can be any sequence of characters except **ENTER**.
RETURN		Pops a line number from the **GOSUB** stack and jumps to the line after it. Error 7 occurs when there is no line number on the stack. There is some mistake in your program; **GOSUB**s are not properly balanced by **RETURN**s.
RUN		RUN 0
RUN n		CLEAR, and then **GOTO** n.
SAVE f		Records the program and variables on tape and calls it f. **SAVE** should not be used inside a **GOSUB** routine. Error F occurs if f is the empty string, which is not allowed.

Appendix A: Summary of T/S 1500 BASIC

Function	Type of operand	Result
SCROLL		Scrolls the display file up one line, losing the top line and making an empty line at the bottom. Note that the new line is genuinely empty with just an ENTER character and no spaces.
SLOW		Puts the computer into compute and display mode, in which the display file is displayed continuously and computing is done during the spaces at the top and bottom of the picture.
STOP		Stops the program with Report 9. CONT will resume with the following line.
UNPLOT m,n		Like PLOT, but blanks out a pixel instead of blacking it in.

Appendix B: The Character Set

This is the complete T/S 1500 character set, with codes in decimal and hex. If one imagines the codes being Z80 machine code instructions, then the right-hand columns give the corresponding assembly language mnemonics. As you are probably aware, certain Z80 instructions are compounds starting with CBh or EDh, as shown in the right-hand columns.

Code	Character
0	space
1	▘
2	▝
3	▀
4	▖
5	▌
6	▞
7	▛
8	▒
9	▚

Hex	Z80 assembler	-after CB	-after ED
00	nop	ric b	
01	ld bc,NN	ric c	
02	ld (bc), a	ric d	
03	inc bc	ric e	
04	inc b	ric h	
05	dec b	ric l	
06	ld b,N	ric (hl)	
07	rlca	ric a	
08	ex af,af'	rrc b	
09	add hl,bc	rrc c	

Appendix B: The Character Set

Code	Character
10	▒
11	"
12	£
13	$
14	:
15	?
16	(
17	)
18	>
19	<
20	=
21	+
22	-
23	*
24	/
25	;
26	'
27	.
28	0
29	1
30	2
31	3
32	4
33	5
34	6
35	7
36	8
37	9
38	A
39	B
40	C
41	D
42	E
43	F
44	G
45	H
46	I
47	J
48	K
49	L
50	M
51	N
52	O
53	P

Hex	Z80 assembler	-after CB	-after ED
0A	ld a,(bc)	rrc d	
0B	dec bc	rrc e	
0C	inc c	rrc h	
0D	dec c	rrc l	
0E	ld c,N	rrc (hl)	
0F	rrca	rrc a	
10	djnz DIS	rl b	
11	ld de,NN	rl c	
12	ld (de),a	rl d	
13	inc de	rl e	
14	inc d	rl h	
15	dec d	rl l	
16	id d,N	rl (hl)	
17	rla	rl a	
18	jr DIS	rr b	
19	add hl,de	rr c	
1A	ld a,(de)	rr d	
1B	dec de	rr e	
1C	inc e	rr h	
1D	dec e	rr l	
1E	ld e,N	rr (hl)	
1F	rra	rr a	
20	jr nz,DIS	sla b	
21	ld hl,NN	sla c	
22	id (NN),hl	sla d	
23	inc hl	sla e	
24	inc h	sla h	
25	dec h	sla l	
26	ld h,N	sla (hl)	
27	daa	sla a	
28	jr z,DIS	sra b	
29	add hl,hl	sra c	
2A	ld hl,(NN)	sra d	
2B	dec hl	sra e	
2C	inc l	sra h	
2D	dec l	sra l	
2E	ld l,N	sra (hl)	
2F	cpl	sra a	
30	jr nc,DIS		
31	ld sp,NN		
32	id (NN),a		
33	inc sp		
34	inc (hl)		
35	dec (hl)		

Appendix B: The Character Set

Code	Character
54	Q
55	R
56	S
57	T
58	U
59	V
60	W
61	X
62	Y
63	Z
64	RND
65	INKEY$
66	PI
67	not used
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	

Hex	Z80 assembler	-after CB	-after ED
36	id (hl),N		
37	scf		
38	jr c,DIS	srl b	
39	add hl,sp	srl c	
3A	ld a,(NN)	srl d	
3B	dec sp	srl e	
3C	inc a	srl h	
3D	dec a	srl l	
3E	ld a,N	srl (hl)	
3F	ccf	srl a	
40	ld b,b	bit 0,b	in b,(c)
41	ld b,c	bit 0,c	out (c),b
42	ld b,d	bit 0,d	sbc hl,bc
43	ld b,e	bit 0,e	ld (NN),bc
44	ld b,h	bit 0,h	neg
45	ld, b,l	bit 0,l	retn
46	ld b,(hl)	bit 0,(hl)	im 0
47	ld b,a	bit 0,a	ld i,a
48	ld c,b	bit 1,b	in c,(c)
49	ld c,c	bit 1,c	out (c),c
4A	ld c,d	bit 1,d	adc hl,bc
4B	ld c,e	bit 1,e	ld bc,(NN)
4C	ld c,h	bit 1,h	
4D	ld c,l	bit 1,l	reti
4E	ld c,(hl)	bit 1,(hl)	
4F	ld c,a	bit 1,a	ld r,a
50	ld d,b	bit 2,b	in d,(c)
51	ld d,c	bit 2,c	out (c),d
52	ld d,d	bit 2,d	sbc hl,de
53	ld d,e	bit 2,e	ld (NN),de
54	ld d,h	bit 2,h	
55	ld d,l	bit 2,l	
56	ld d,(hl)	bit 2,(hl)	im 1
57	ld d,a	bit 2,a	ld a,i
58	ld e,b	bit 3,b	in e,(c)
59	ld e,c	bit 3,c	out (c),e
5A	ld e,d	bit 3,d	adc hl,de
5B	ld e,e	bit 3,e	ld de,(NN)
5C	ld e,h	bit 3,h	
5D	ld e,l	bit 3,l	
5E	ld e,(hl)	bit 3,(hl)	im 2
5F	ld e,a	bit 3,a	ld a,r
60	ld h,b	bit 4,b	in h,(c)
61	ld, h,c	bit 4,c	out (c),h

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
98	not used	62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld, la	bit 5,a	rid
112	cursor up ▲	70	ld (hl),b	bit 6,b	
113	cursor down ▼	71	ld (hl),c	bit 6,c	
114	cursor left ◀	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor right ▶	73	ld (hl),e	bit 6,e	ld (NN),sp
116	GRAPHICS	74	ld (hl),h	bit 6,h	
117	EDIT	75	ld (hl),l	bit 6,l	
118	ENTER	76	halt	bit 6,(hl)	
119	DELETE	77	ld (hl),a	bit 6,a	
120	K / L mode	78	ld a,b	bit 7,b	in a,(c)
121	FUNCTION	79	ld a,c	bit 7,c	out (c),a
122	not used	7A	ld a,d	bit 7,d	adc hl,sp
123	not used	7B	ld a,e	bit 7,e	ld sp,(NN)
124	not used	7C	ld a,h	bit 7,h	
125	not used	7D	ld a,l	bit 7,l	
126	number	7E	ld a,(hl)	bit 7,(hl)	
127	cursor	7F	ld a,a	bit 7,a	
128	■	80	add a,b	res 0,b	
129	▟	81	add a,c	res 0,c	
130	▙	82	add a,d	res 0,d	
131	▄	83	add a,e	res 0,e	
132	▛	84	add a,h	res 0,h	
133	▜	85	add a,l	res 0,l	
134	▚	86	add a,(hl)	res 0,(hl)	
135	▗	87	add a,a	res 0,a	
136	▒	88	adc a,b	res 1,b	
137	▌	89	adc a,c	res 1,c	
138	▀	8A	adc a,d	res 1,d	
139	inverse"	8B	adc a,e	res 1,e	
140	inverse	8C	adc a,h	res 1,h	
141	inverse$	8D	adc a,l	res 1,l	

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
142	inverse:	8E	adc a,(hl)	res 1,(hl)	
143	inverse ?	8F	adc a,a	res 1,a	
144	inverse (	90	sub b	res 2,b	
145	inverse)	91	sub c	res 2,c	
146	inverse >	92	sub d	res 2,d	
147	inverse <	93	sub e	res 2,e	
148	inverse =	94	sub h	res 2,h	
149	inverse +	95	sub l	res 2,l	
150	inverse -	96	sub (hl)	res 2,(hl)	
151	inverse *	97	sub a	res 2,a	
152	inverse /	98	sbc a,b	res 3,b	
153	inverse ;	99	sbc a,c	res 3,c	
154	inverse ,	9A	sbc a,d	res 3,d	
155	inverse .	9B	sbc a,e	res 3,e	
156	inverse 0	9C	sbc a,h	res 3,h	
157	inverse 1	9D	sbc a,l	res 3,l	
158	inverse 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverse 3	9F	sbc a,a	res 3,a	
160	inverse 4	A0	and b	res 4,b	ldi
161	inverse 5	A1	and c	res 4,c	cpi
162	inverse 6	A2	and d	res 4,d	ini
163	inverse 7	A3	and e	res 4,e	outi
164	inverse 8	A4	and h	res 4,h	
165	inverse 9	A5	and l	res 4,l	
166	inverse A	A6	and (hl)	res 4,(hl)	
167	inverse B	A7	and a	res 4,a	
168	inverse C	A8	xor b	res 5,b	ldd
169	inverse D	A9	xor c	res 5,c	cpd
170	inverse E	AA	xor d	res 5,d	ind
171	inverse F	AB	xor e	res 5,e	outd
172	inverse G	AC	xor h	res 5,h	
173	inverse H	AD	xor l	res 5,l	
174	inverse I	AE	xor (hl)	res 5,(hl)	
175	inverse J	AF	xor a	res 5,a	
176	inverse K	B0	or b	res 6,b	ldir
177	inverse L	B1	or c	res 6,c	cpir
178	inverse M	B2	or d	res 6,d	inir
179	inverse N	B3	or e	res 6,e	otir
180	inverse O	B4	or h	res 6,h	
181	inverse P	B5	or l	res 6,l	
182	inverse Q	B6	or (hl)	res 6,(hl)	
183	inverse R	B7	or a	res 6,a	
184	inverse S	B8	cp b	res 7,b	lddr
185	inverse T	B9	cp c	res 7,c	cpdr

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
186	inverse U	BA	cp d	res 7,d	indr
187	inverse V	BB	cp e	res 7,e	otdr
188	inverse W	BC	cp h	res 7,h	
189	inverse X	BD	cp l	res 7,l	
190	inverse Y	BE	cp (hl)	res 7,(hl)	
191	inverse Z	BF	cp a	res 7,a	
192	" "	C0	ret nz	set 0,b	
193	AT	C1	pop bc	set 0,c	
194	TAB	C2	jp nz,NN	set 0,d	
195	not used	C3	jp NN	set 0,e	
196	CODE	C4	call nz,NN	set 0,h	
197	VAL	C5	push bc	set 0,l	
198	LEN	C6	add a,N	set 0,(hl)	
199	SIN	C7	rst 0	set 0,a	
200	COS	C8	ret z	set 1,b	
201	TAN	C9	ret	set 1,c	
202	ASN	CA	jp z,NN	set 1,d	
203	ACS	CB		set 1,e	
204	ATN	CC	call z,NN	set 1,h	
205	LN	CD	call NN	set 1,l	
206	EXP	CE	adc a,N	set 1,(hl)	
207	INT	CF	rst 8	set 1,a	
208	SQR	D0	ret nc	set 2,b	
209	SGN	D1	pop de	set 2,c	
210	ABS	D2	jp nc,NN	set 2,d	
211	PEEK	D3	out N,a	set 2,e	
212	USR	D4	call nc,NN	set 2,h	
213	STR$	D5	push de	set 2,l	
214	CHR$	D6	sub N	set 2,(hl)	
215	NOT	D7	rst 16	set 2,a	
216	**	D8	ret c	set 3,b	
217	OR	D9	exx	set 3,c	
218	AND	DA	jp c,NN	set 3,d	
219	<=	DB	in a,N	set 3,e	
220	>=	DC	call c,NN	set 3,h	
221	<>	DD	prefixes instructions using ix	set 3,l	
222	THEN	DE	sbc a,N	set 3,(hl)	
223	TO	DF	rst 24	set 3,a	
224	STEP	E0	ret po	set 4,b	
225	LPRINT	E1	pop hl	set 4,c	
226	LLIST	E2	jp po,NN	set 4,d	
227	STOP	E3	ex (sp),hl	set 4,e	
228	SLOW	E4	call po,NN	set 4,h	

Appendix B: The Character Set

Code	Character	Hex	Z80 assembler	-after CB	-after ED
229	FAST	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4,(hl)	
231	SCROLL	E7	rst 32	set 4,a	
232	CONT	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GOTO	EC	call pe,NN	set 5,h	
237	GOSUB	ED		set 5,l	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RAND	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	UNPLOT	FC	call m,NN	set 7,h	
253	CLEAR	FD	prefixes instructions using iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	

Appendix C: Using Machine Code

This appendix is written for those who understand Z80 machine code, the set of instructions that the Z80 processor chip uses. If you wish to learn more about Z80 machine codes and programming, many books are available.

The ultimate authority is the *Z80 Assembly Language Programming Manual*, together with the *Z80-CPU, Z80A-CPU Technical Manual*, published by Zilog; but these can hardly be recommended for beginners.

Machine code routines can be executed from within a BASIC program by using the function **USR**. The argument of **USR** is the starting address of the routine, and its result is a two-byte unsigned integer, the contents of the bc register pair on return. The return address to the BASIC is stacked in the usual way, so return is by a Z80 **RET** instruction.

Appendix C: Using Machine Code

There are certain restrictions on USR routines:

(i) On return, the i register must have the value 1Eh.

(ii) The display routine uses a', f', ix, iy and r registers, a USR routine should not use these if compute and display is operating. (It is not even safe to read the af' pair.)

All these lines from the processor are exposed at the back of the T/S 1500, so in principle you can do anything with a T/S 1500 that you can with a Z80. The T/S 1500 hardware might sometimes get in the way, though, especially in compute and display. Here is a diagram of the exposed connections at the back:

A piece of machine code in the middle of memory runs the risk of being overwritten by the BASIC system. Some safer places are

(i) In a REM statement: type in a REM statement with enough characters to hold your machine code, which you then poke in. Avoid halt instructions, since these will be recognized as the end of the REM statement.

(ii) In a string: set up a long enough string, and then assign a machine code byte to each character.

In both of these the code is safe, but likely to move about; this is especially the case with strings. In the Appendix, the character set, you will find the characters and Z80 instructions written side by side in order, and you may well find this useful when entering code.

(iii) At the top of the memory. When the T/S 1500 is switched on, it tests to see how much memory there is and puts the machine stack right at the top so that there is no space for USR routines there. It stores the address of the first nonexistent byte (e.g. 33K, or 32768, if you have 16K memory) in a system variable known as RAMTOP, in the two bytes with addresses 16388 and 16389. NEW, on the other hand, does not do a full memory test,

but only checks up as far as just before the address in RAMTOP. Thus, if you poke the address of an existing byte into RAMTOP, for **NEW** all the memory from that byte on is outside the BASIC system and is left alone. For instance, suppose you have 1K memory and you have just switched on the computer.

PRINT PEEK 16388 + 256 * **PEEK** 16389

tells you the address (32768) of the first nonexistent byte.

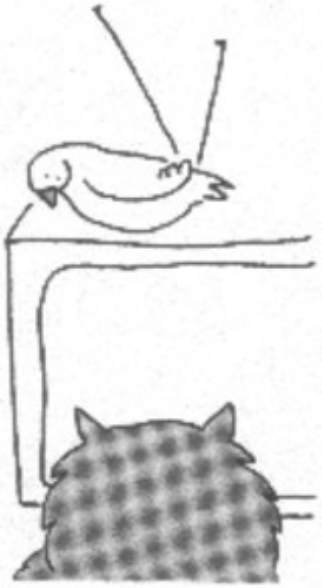
Now suppose you have a **USR** routine 20 bytes long. You want to change RAMTOP to 32748 = 236 + 256*127 (how would you work this out in the computer?), so type

POKE 16388,236
POKE 16389,127

and then **NEW**. The twenty bytes of memory from address 32748 to 32768 are now yours to do with what you like. If you then type **NEW** again it will not affect these twenty bytes.

The top of memory is a good place for **USR** routines, safe (even from **NEW**) and immobile. Its main disadvantage is that it is not saved by **SAVE**.

Appendix D: Organization of Storage

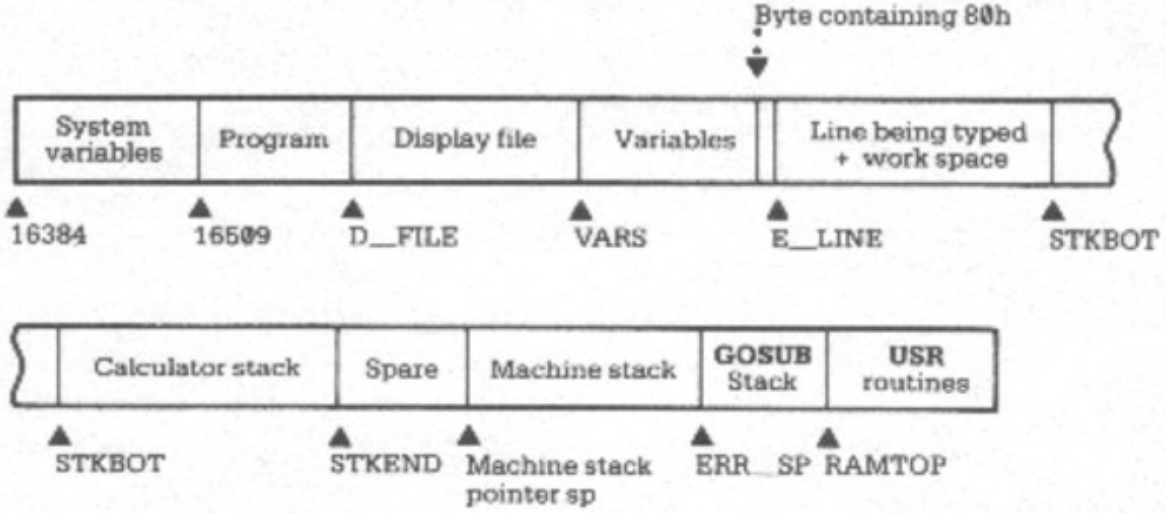


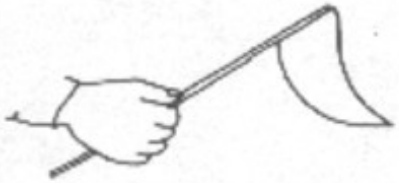
The memory is divided into different areas for storing different kinds of information. The areas are only large enough for the information they actually contain, and if you insert more at a given point (for instance, by adding a program line or variable), space is made by shifting up everything above that point. Conversely, if you delete information, everything above the deletion is shifted down.

The operating system is contained in a ROM with a starting address of 0 and an ending address of 8191.

RAM memory starts at address 16384 and extends to 32767 for your T/S 1500.

Appendix D: Organization of Storage





An additional 16K Ram Pack installed on the system is not detected by the system software. Hence, the system variable RAMTOP must be manually changed and a **NEW** command executed to incorporate the RAM into the system for use by BASIC. The following will incorporate the additional RAM into the system:

```
POKE 16388, 0
POKE 16389, 192
NEW
```

The user must be aware of one fact when using the 16K RAM pack on the T/S 1500 system. When BASIC program lines are typed (or loaded from external device) into the system, the display file, i.e. what is on the screen at the time, moves higher into memory. If more text is displayed on the screen, the display file gets larger and moves higher into memory. The problem is if any part of the display file moves past 32767 into higher memory, system operation is impaired and may not function properly.

The danger of the display file crossing the 32K boundary only exists while BASIC programs are being entered after poking RAMTOP to a value greater than 32768. This is because the system only compares the amount of memory used with RAMTOP to determine if the system is out of memory.

To find out how many bytes of RAM are still available for BASIC program lines, type this test:

```
PRINT 32768 — PEEK 16400 — 256 * PEEK 16401
```

Appendix D: Organization of Storage

You will only find it necessary if you are writing a very long program.

The memory from 32768 to 49151 is available for variables and machine code that resides in string variables or between RAMTOP and the actual top of memory.

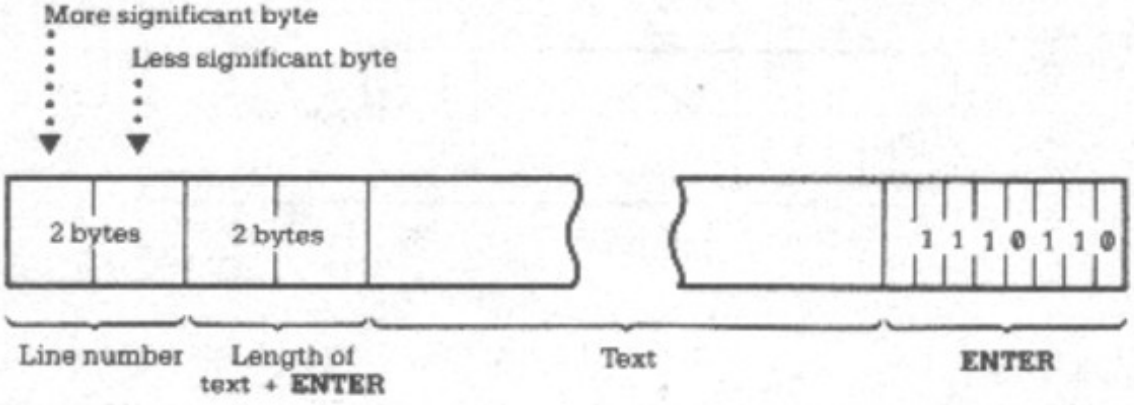
If you want to check your memory you can try typing in this test—it will show whatever memory is available, e.g., 16K, 32K.

MEMORY TEST PROGRAM

```
1 POKE 18000, 33
2 POKE 18001, 11
3 POKE 18002, 0
4 POKE 18003, 57
5 POKE 18004, 68
6 POKE 18005, 77
7 POKE 18006, 201
8 PRINT (USR (19000) – 16373)/1024; "K"
```

The system variables contain various pieces of information that tell the computer what sort of state the computer is in. They are listed fully in the next chapter, but for the moment note that there are some (called D__FILE, VARS, E__LINE and so on) that contain the addresses of the boundaries between the various areas in the memory. These are not BASIC variables, and their names will not be recognized by the computer.

In the program, each line is stored as



Appendix D: Organization of Storage

Note that, in contrast with all other cases of two-byte numbers in the Z80, the line number here (and also in a **FOR-NEXT** control variable) is stored with its more significant byte first: that is to say, in the order that you would write them down.

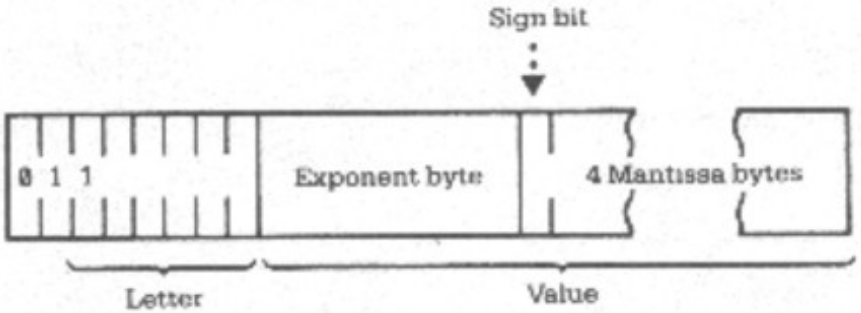
A numerical constant in the program is followed by its binary form, using the character **CHR$** 126 followed by five bytes for the number itself.

The display file is the memory copy of the television picture. It begins with an **ENTER** character, followed by twenty-four lines of text, each finishing with an **ENTER**. The system is so designed that a line of text does not need a full thirty-two characters: final spaces can be omitted. This is done to save space when the memory is small.

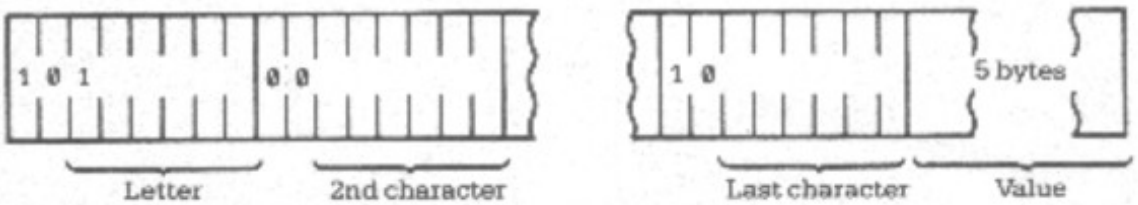
When the total amount of memory (according to the system variable RAMTOP) is less then 3¼K, a clear screen—as set up at the start or by **CLS**—consists of just 25 **ENTER**s. When the memory is bigger, a clear screen is padded out with 24*32 spaces, and on the whole it stays at its full size; **SCROLL**, however, and certain conditions where the lower part of the screen expands to more than two lines, can upset this by introducing short lines at the bottom.

The variables have different formats, depending on their different natures.

Number whose name is one letter only:

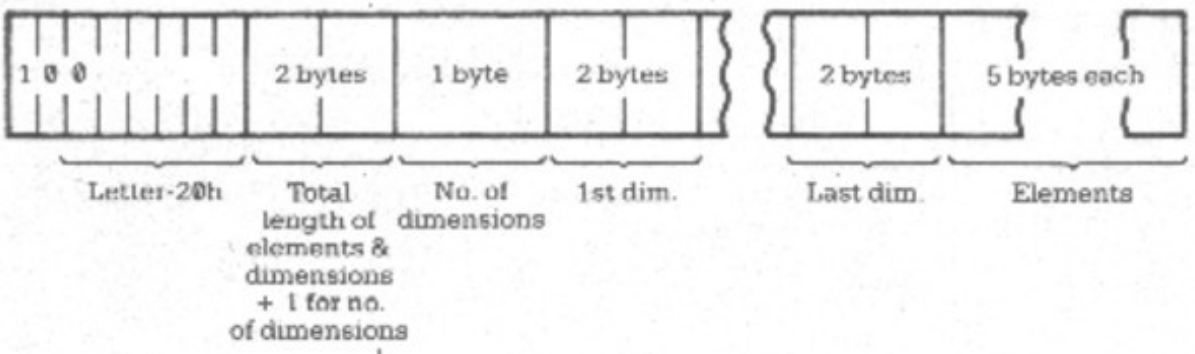


Number whose name is longer than one letter:



Appendix D: Organization of Storage

Array of numbers:



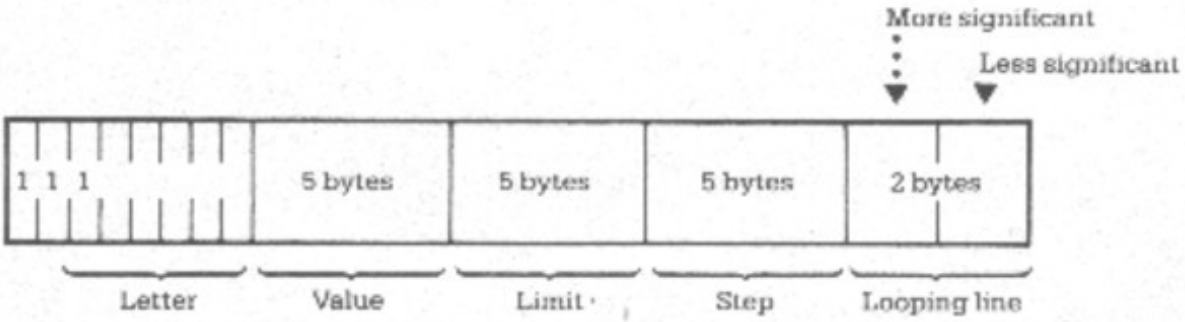
The order of the elements is:

first, the elements for which the first subscript is 1
next, the elements for which the first subscript is 2
next, the elements for which the first subscript is 3
and so on for all possible values of the first subscript.

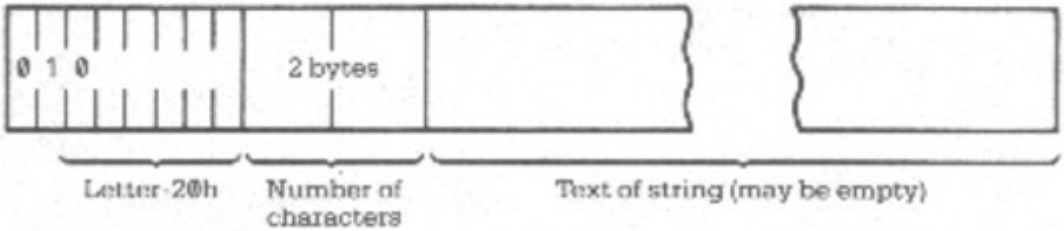
The elements with a given first subscript are ordered in the same way using the second subscript, and so on down to the last.

As an example, the elements of the 3 × 6 array B in Chapter 22 are stored in the order B(1,1), B(1,2), B(1,3), B(1,4), B(1,5), B(1,6), B(2,1), B(2,2),. . ., B(2,6), B(3,1), B(3,2),. . ., B(3,6).

Control variable of a **FOR-NEXT** loop:

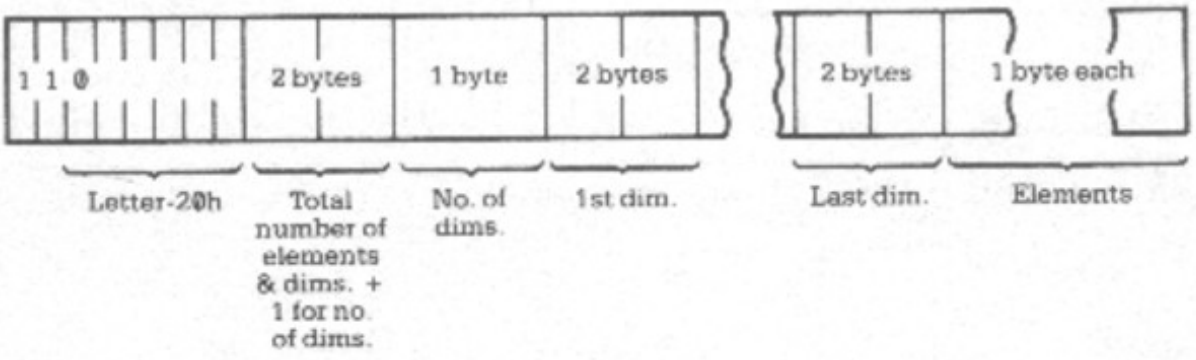


String:



Appendix D: Organization of Storage

Array of characters:



The part starting E_LINE contains the line being typed (as a command, a program line, or **INPUT** data) and also some work space.

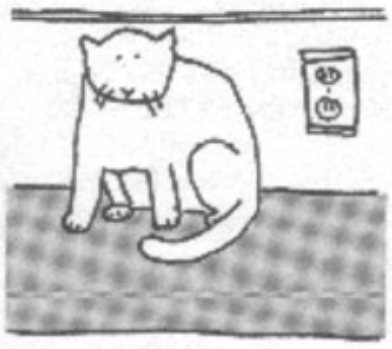
The calculator is the part of the BASIC system that deals with arithmetic, and the numbers it is operating are held mostly in the calculator stack.

The spare part contains the space so far unused.

The machine stack is the stack used by the Z80 chip to hold return addresses and so on.

The space for **USR** routines has to be set aside by you, using **NEW** as described in the Appendix on Using Machine Code.

Appendix E: The System Variables



The bytes in memory from 16384 to 16508 are set aside for specific uses by the system. You can peek them to find out various things about the system, and some of them can be usefully poked. They are listed here with their uses.

These are called system variables and carry names, but do not confuse them with the variables used by the BASIC. You cannot use the names in a BASIC program; they are simply mnemonics that are used to make it easier to refer to the variables.

The abbreviations in column 1 have the following meanings.

X The variable should not be poked, because the system might crash.
N Poking the variable will have no lasting effect.
S The variable is saved by **SAVE**.

The number in column 1 is the number of bytes in the variable. For two bytes, the first one is the *less* significant byte—the reverse of what you might expect. So to poke a value v to a two-byte variable at address n, use

Appendix E: The System Variables

POKE n,v—256***INT**(v/256)
POKE n + 1,**INT** (v/256)

and to peek its value,use the expression:

PEEK n + 256***PEEK** (n + 1)

Notes	Address	Name	Contents
1	16384	ERR_NR	1 less than the report code. Starts off at 255 (for – 1), so **PEEK** 16384, if it works at all, gives 255. **POKE** 16384,n can be used to force an error halt: 0<=n<=14 gives one of the usual reports, 15<=n<=34 or 99<=n<=127 gives a non-standard report, and 35<=n<=98 is likely to mess up the display file.
X1	16385	FLAGS	Various flags to control the BASIC system.
X2	16386	ERR_SP	Address of first item on machine stack (after **GOSUB** returns).
2	16388	RAMTOP	Address of first byte above BASIC system area. You can poke this to make **NEW** reserve space above that area or to fool **CLS** into setting up a minimal display file.
N1	16390	MODE	Specifies K, L, F or G cursor.
N2	16391	PPC	Line number of statement currently being executed. Poking this has no lasting effect except in the last line of the program.
S1	16393	VERSN	0 Identifies 8K ROM in saved programs.
S2	16394	E_PPC	Number of current line (with program cursor).
SX2	16396	D_FILE	See Appendix D.
S2	16398	DF_CC	Address of **PRINT** position in display file. Can be poked so that **PRINT** output is sent elsewhere.
SX2	16400	VARS	See Appendix D.
SN2	16402	DEST	Address of variable in assignment.
SX2	16404	E_LINE	See Appendix D.
SX2	16406	CH_ADD	Address of the next character to be interpreted: the character after the argument of **PEEK**, or the **ENTER** at the end of a **POKE** statement.
S2	16408	X_PTR	Address of the character preceding the marker.
SX2	16410	STKBOT	See Appendix D.
SX2	16412	STKEND	
SN1	16414	BREG	Calculator's b register.

Appendix E: The System Variables

Notes	Address	Name	Contents
SN2	16415	MEM	Address of area used for calculator's memory. (Usually MEMBOT, but not always.)
S1	16417	not used	
SX1	16418	DF_SZ	The number of lines (including one blank line) in the lower part of the screen.
S2	16419	S_TOP	The number of the top program line in automatic listings.
SN2	16421	LAST_K	Shows which keys pressed.
SN1	16423		Debounce status of keyboard.
SN1	16424	MARGIN	Number of blank lines above or below picture—31.
SX2	16425	NXTLIN	Address of next program line to be executed.
S2	16427	OLDPPC	Line number to which **CONT** jumps.
SN1	16429	FLAGX	Various flags.
SN2	16430	STRLEN	Length of string type designation in assignment.
SN2	16432	T-ADDR	Address of next item in syntax table (very unlikely to be useful).
S2	16434	SEED	The seed for **RND**. This is the variable that is set by **RAND**.
S2	16436	FRAMES	Counts the frames displayed on the television. Bit 15 is 1. Bits 0 to 14 are decremented for each frame sent to the television. This can be used for timing, but **PAUSE** also uses it. **PAUSE** resets bit 15 to 0 and puts in bits 0 to 14 the length of the pause. When these have been counted down to zero, the pause stops. If the pause stops because of a key depression, bit 15 is set to one again.
S1	16438	COORDS	x-coordinate of last point **PLOT**ted.
S1	16439		y-coordinate of the last point **PLOT**ted.
S1	16440	PR_CC	Less significant byte of address of next position for **LPRINT** to print at (in PRBUFF).
SX1	16441	S_POSN	Column number for **PRINT** position.
SX1	16442		Line number for **PRINT** position.
S1	16443	CDFLAG	Various flags. Bit 7 is on (1) during compute and display.
S33	16444	PRBUFF	Printer buffer (33rd character) is **ENTER**.
SN30	16477	MEMBOT	Calculator's memory area; used to store numbers that cannot conveniently be put on the calculator stack.
S2	16507	not used	

Appendix F: Keyword Reference Table

Tokens (Key Words & Functions)	Primary Key Name	How To Obtain
ABS	G	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
AND	2	**SHIFT** Primary Key
ARCCOS (ACS)	S	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
ARCSIN (ASN)	A	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
ARCTAN (ATN)	D	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
AT	C	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
BREAK	SPACE	Need K cursor, then press the Primary Key
CHR$	U	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
CLEAR	X	Need K cursor, then press the Primary Key

Appendix F: Keyword Reference Table

Tokens (Key Words & Functions)	Primary Key Name	How To Obtain
CLS	V	Need K cursor, then press the Primary Key
CODE	I	Need F cursor (SHIFT ENTER Key), then press the Primary Key
CONT	C	Need K cursor, then press the Primary Key
COPY	Z	Need K cursor, then press the Primary Key
COS	W	Need F cursor (SHIFT ENTER Key), then press the Primary Key
DELETE	0	SHIFT Primary Key
DIM	D	Need K cursor, then press the Primary Key
EDIT	1	SHIFT Primary Key
ENTER		Primary Key
EXP	X	Need F cursor (SHIFT ENTER Key), then press the Primary Key
FAST	F	SHIFT Primary Key
FOR	F	Need K cursor, then press the Primary Key
FUNCTION F	ENTER	SHIFT Primary Key
GOSUB	H	Need K cursor, then press the Primary Key
GOTO	G	Need K cursor, then press the Primary Key
GRAPHICS G	9	SHIFT Primary Key
IF	U	Need K cursor, then press the Primary Key
IN KEY$	B	Need F cursor (SHIFT ENTER Key), then press the Primary Key
INPUT	I	Need K cursor, then press the Primary Key
INT	R	Need F cursor (SHIFT ENTER Key), then press the Primary Key
LEN	K	Need F cursor (SHIFT ENTER Key), then press the Primary Key
LET	L	Need K cursor, then press the Primary Key
LIST	K	Need K cursor, then press the Primary Key
LLIST	G	SHIFT Primary Key
LN	Z	Need F cursor (SHIFT ENTER Key), then press the Primary Key
LOAD	J	Need K cursor, then press the Primary Key
LPRINT	S	SHIFT Primary Key
NEW	A	Need K cursor, then press the Primary Key

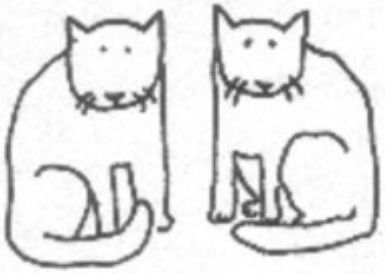
Appendix F: Keyword Reference Table

Tokens (Key Words & Functions)	Primary Key Name	How To Obtain
NEXT	N	Need K cursor, then press the Primary Key
NOT	N	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
OR	W	**SHIFT** Primary Key
PAUSE	M	Need K cursor, then press the Primary Key
PEEK	O	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
PLOT	Q	Need K cursor, then press the Primary Key
POKE	O	Need K cursor, then press the Primary Key
PRINT	P	Need K cursor, then press the Primary Key
RAND	T	Need K cursor, then press the Primary Key
REM	E	Need K cursor, then press the Primary Key
RETURN	Y	Need K cursor, then press the Primary Key
RND	T	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
RUN	R	Need K cursor, then press the Primary Key
SAVE	S	Need K cursor, then press the Primary Key
SCROLL	B	Need K cursor, then press the Primary Key
SGN	F	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
SHIFT KEY		Primary Key
SIN	Q	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
SLOW	D	**SHIFT** Primary Key
SPACE		Primary Key
SQR	H	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
STEP	E	**SHIFT** Primary Key
STOP	A	**SHIFT** Primary Key
STR$	Y	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
TAB	P	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
TAN	E	Need F cursor (**SHIFT ENTER** Key), then press the Primary Key
THEN	3	**SHIFT** Primary Key

Appendix F: Keyword Reference Table

Tokens (Key Words & Functions)	Primary Key Name	How To Obtain
TO	4	SHIFT Primary Key
USR	L	Need F cursor (SHIFT ENTER Key), then press the Primary Key
VAL	J	Need F cursor (SHIFT ENTER Key), then press the Primary Key
-	J	SHIFT Primary Key
>=	Y	SHIFT Primary Key
<=	R	SHIFT Primary Key
"	P	SHIFT Primary Key
$	U	SHIFT Primary Key
(	I	SHIFT Primary Key
)	O	SHIFT Primary Key
*	B	SHIFT Primary Key
**	H	SHIFT Primary Key
+	K	SHIFT Primary Key
,	.	SHIFT Primary Key
/	V	SHIFT Primary Key
:	Z	SHIFT Primary Key
;	X	SHIFT Primary Key
=	L	SHIFT Primary Key
?	C	SHIFT Primary Key

Appendix G: Index



Appendix G: Index

Appendix G: Index

Appendix G: Index

Appendix G: Index

Appendix G: Index

Appendix G: Index

Appendix G: Index

Appendix H: Report Codes

This table gives each report code, with a general description and a list of the statements and functions in which it can occur.

Code	Situations	Meaning
0	Any	Successful completion, or jump to line number bigger than any existing. A report with code 0 does not change the line number used by CONT.
1	NEXT	The control variable does not exist (has not been set up by a FOR statement), but there is an ordinary variable with the same name.
2	Any	An undefined variable has been used. For a simple variable this will happen if the variable is used before it has been assigned to in a LET statement. For a subscripted variable it will happen if the variable is used before it has been dimensioned in a DIM statement. For a control variable in a FOR statement and if there is no ordinary simple variable with the same name.

Appendix H: Report Codes

Code	Situations	Meaning
3	Subscripted variables	Subscript out of range. If the subscript is out of range (negative, or bigger than 65535), error B will result.
4	LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB. Sometimes during function evaluation.	Not enough room in memory. Note that the line number in the report (after the /) may not be complete on the screen, because of the shortage of memory. for instance, 4/20 may appear as 4/2. See Chapter 9.
5	PRINT, LIST, PLOT, UNPLOT	No more room on the screen. CONT will make room by clearing the screen.
6	Any arithmetic	Arithmetic overflow: calculations have led to a number great than about 10^{39}.
7	RETURN	No corresponding GOSUB for a RETURN statement.
8	INPUT	You have attempted an INPUT command (not allowed).
9	STOP	STOP statement executed. CONT will not try to reexecute the STOP statement.
A	SQR, LN, ASN, ACS	Invalid argument to certain functions.
B	RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR	Integer out of range. When an integer is required, the floating-point argument is rounded to the nearest integer. If this is outside a suitable range, error B results.
	Array access	For array access, see also Report 3.
C	VAL	The text of the (string) argument of VAL does not form a valid numerical expression.
D	At the end of any statement, or in LOAD, SAVE, LPRINT, LLIST, or COPY.	(i) Program interrupted by BREAK.
	INPUT	(ii) The INPUT line starts with STOP.
E		Not used
F	SAVE	The program name provided is the empty string.

Notes

Notes

Join The Club

Get the Most out of Your Timex Sinclair

Join the Timex Computer Club!

The Timex Computer Club is an exclusive group of Timex Sinclair Computer Owners. As a member, you will receive early notices and up-dates of Timex Computer Corporation technological advances, new hardware and software products, creative programming ideas and special offers. You will also be able to share computer ideas and achievements with other club members all over the country!

To enroll in the Timex Computer Club, simply answer the questions below and return this card promptly. Your answers will assist us in developing the kinds of products and services which will best serve your needs.

We welcome you and are looking forward to hearing from you.

CUT ALONG DOTTED LINE

1. 1 ☐ Mr. 2 ☐ Mrs. 3 ☐ Ms. 4 ☐ Miss

First Name | Initial | Last Name

Street | Apt. No.

City | State | Zip

2. Date of Purchase: Mo | Day | Yr

Date of Birth: of person listed above: Mo | Yr (19__)

3. Where purchased?
- ☐ Received as Gift
- ☐ Department Store
- ☐ Discount Store
- ☐ Drug Store
- ☐ Catalog Showroom
- ☐ Computer Store
- ☐ Camera Store
- ☐ Radio-TV-Electronic Store
- ☐ Mail Order
- ☐ Other

4. Is this the first personal computer you have owned? ☐ Yes ☐ No

5. How will you use your computer?
- ☐ Household
- ☐ Education
- ☐ Business
- ☐ Entertainment
- ☐ Word Processing
- ☐ Learn to Program
- ☐ Become Familiar with Computers
- ☐ Other

PLACE
FIRST-CLASS
STAMP
HERE

Timex Product Service Center
P.O. Box K
7004 Murray Street
Little Rock, AR 72203

— FOLD HERE —

6. Where did you learn about your Timex computer product?
   - ☐ TV Advertising
   - ☐ Magazine
   - ☐ Newspaper
   - ☐ Advertising in Store
   - ☐ From School
   - ☐ Friend
   - ☐ Salesperson
   - ☐ Other

7. Which three factors most influenced you to buy your Timex computer product?
   - ☐ Price
   - ☐ Ease of Use
   - ☐ Size
   - ☐ Advanced Technology
   - ☐ Power/Capabilities
   - ☐ Timex Reputation
   - ☐ Special Features
   - ☐ Other

8. What is the highest level of education you have completed?
   - ☐ Grade 8 or less
   - ☐ Grade 9-11
   - ☐ Graduated High School
   - ☐ 1-3 Years of College
   - ☐ Graduated College
   - ☐ Attended or Completed Graduate School

9. Which group describes your annual family income?
   - ☐ Under $10,000
   - ☐ $10,000-$14,999
   - ☐ $15,000-$19,999
   - ☐ $20,000-$24,999
   - ☐ $25,000-$34,999
   - ☐ $35,000-$49,999
   - ☐ $50,000 and over

10. Do you have children at home? ☐ Yes ☐ No

If you have comments or suggestions about our product, please write to:

TIMEX CONSUMER SERVICE DEPARTMENT
P.O. Box 2655
Waterbury, CT 06725

338-879001

TIMEX Timex Computer Corporation Waterbury, Connecticut 06720