

SHARP

POCKET COMPUTER

MODEL **PC-1500**

INSTRUCTION MANUAL



**WWW.
PC-1500
.INFO**

Do not sale this PDF !!!



AN INTRODUCTORY NOTE

Allow us to thank you for purchasing the SHARP PC-1500 Pocket Computer. We are confident that you will enjoy using this small, but powerful, new friend in your daily life. The PC-1500 is one of the world's most sophisticated hand held computers. Although it shares many features with its cousin, the SHARP PC-1211 Pocket Computer, the PC-1500 provides you with such advanced capabilities as:

- A 7 by 156 programmable dot-matrix LCD display.
- A tone generator for creating special effects under program control.
- ASCII character set with upper and lower cases.
- Scientific and mathematical functions.
- User-definable function keys.
- An extended version of BASIC which provides two-dimensional arrays, variable length strings, graphics commands, program chaining and many other advanced features.
- Up to 4K bytes of optional RAM. (Model CE-151)
- An optional Printer/Cassette Interface (Model CE-150) which allows 4 color X-Y plotting, program and data storage, and printing of programs and data in one of nine different character sizes.

This machine is capable of many of the functions which only a few years ago would have filled a warehouse with tubes, wires, and engineers. Such sophistication does not require Engineering credentials to use. On the contrary, the PC-1500, and this manual, are designed to help you gain rapid access to this new technology.

We have divided this manual into five major sections allowing the novice user to rapidly attain competence. Advanced users may explore the features of the PC-1500 through the sections on Advanced Programming, and Advanced Calculations, and through the Appendices.

The style of this manual is conversational and many examples are provided. But don't take our word for it, to see how easy it is to get started, turn to Chapter O. But first, be sure that the batteries have been loaded. If they haven't, Appendix B provides instructions.

Above all, have fun and don't hesitate to experiment!

OPERATIONAL NOTES

Since the liquid crystal display of the computer is made of glass, it must be handled with some care.

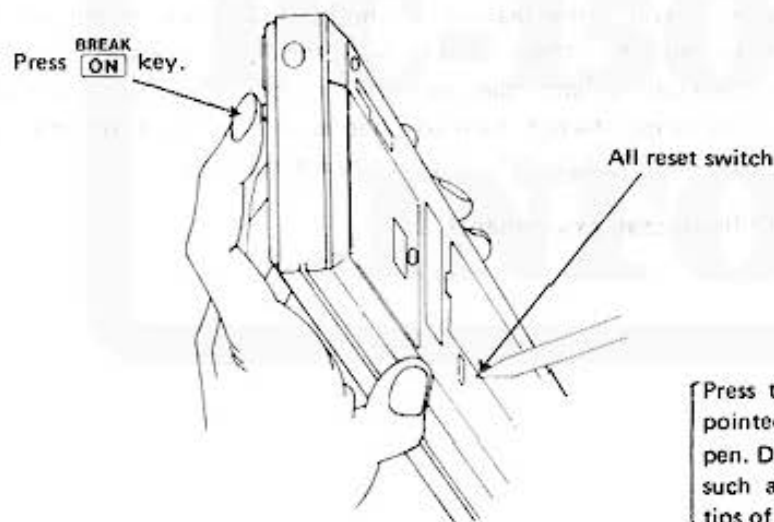
To ensure trouble-free operation of your SHARP pocket computer we recommend that:

1. You keep the computer in an area free from extreme temperature changes, moisture, or dust. During warm weather, vehicles left in direct sun light are subject to high temperature build up. Prolonged exposure to high temperature may cause damage to your computer.
2. You use only a soft, dry cloth to clean the computer. Do not use solvents, water, or wet cloths.
3. To avoid battery leakage, remove the batteries when the computer will not be used for an extended period of time.
4. If service is required, the computer be returned only to an authorized SHARP Service Center.
5. This manual be kept for further reference.

Trouble-Shooting

This unit, if subjected to strong external noise or impact during operation, may render all its keys, including **BREAK ON** key, inoperative.

Should this occur, press the ALL RESET switch on the back of the unit for approx. 15 seconds, with the **BREAK ON** key held down.



Press the all reset switch with any pointed object such as a ball-point pen. Do not use easily broken points such as mechanical pencils or the tips of needles.

Then check that **NEW0? : CHECK** is indicated on the display, and press the keys

CL N E W 0 ENTER .

If the display does not read **NEW0? : CHECK**, perform the above operation once again.

With this operation, the program, data and all the reserved contents are cleared, so do not press the ALL RESET switch except when the above trouble occurs.

PC-1500 SPECIFICATIONS

Model:	PC-1500 Pocket Computer																								
Number of calculation digits:	10 digits (mantissa) + 2 digits (exponent)																								
Calculation system:	According to mathematical formula (with priority judging function)																								
Program language:	BASIC																								
Capacity:	<table border="0"> <tr> <td>CPU:</td> <td>CMOS 8 bit</td> </tr> <tr> <td>System ROM:</td> <td>16 K Bytes</td> </tr> <tr> <td>Memory Capacity RAM:</td> <td>3.5 K Bytes</td> </tr> <tr> <td> System area</td> <td>0.9 K Bytes</td> </tr> <tr> <td> Input buffer:</td> <td>80 Bytes</td> </tr> <tr> <td> Stack:</td> <td>196 Bytes</td> </tr> <tr> <td> Others:</td> <td></td> </tr> <tr> <td> User area</td> <td>2.6 K Bytes</td> </tr> <tr> <td> Fixed memory area:</td> <td>624 Bytes</td> </tr> <tr> <td> (A~Z, AS~ZS)</td> <td></td> </tr> <tr> <td> Basic program data area:</td> <td>1850 Bytes</td> </tr> <tr> <td> Reserve area:</td> <td>188 Bytes</td> </tr> </table>	CPU:	CMOS 8 bit	System ROM:	16 K Bytes	Memory Capacity RAM:	3.5 K Bytes	System area	0.9 K Bytes	Input buffer:	80 Bytes	Stack:	196 Bytes	Others:		User area	2.6 K Bytes	Fixed memory area:	624 Bytes	(A~Z, AS~ZS)		Basic program data area:	1850 Bytes	Reserve area:	188 Bytes
CPU:	CMOS 8 bit																								
System ROM:	16 K Bytes																								
Memory Capacity RAM:	3.5 K Bytes																								
System area	0.9 K Bytes																								
Input buffer:	80 Bytes																								
Stack:	196 Bytes																								
Others:																									
User area	2.6 K Bytes																								
Fixed memory area:	624 Bytes																								
(A~Z, AS~ZS)																									
Basic program data area:	1850 Bytes																								
Reserve area:	188 Bytes																								
Calculations:	Four arithmetic calculations, power calculation, trigonometric and inverse trigonometric functions, logarithmic and exponential functions, angular conversion, extraction of square root, sign function, absolutes, integers and logical calculations.																								
Editing function:	Cursor shifting (▶ ◀) Insertion (INS) Deletion (DEL) Line up and down (↓ , ↑)																								
Memory protection:	CMOS battery back-up (program, data and reserve memories are protected)																								
Display:	Liquid Crystal 26 Character Width 7 x 156 Dot Graphics																								
Keys:	65 Keys including Alphabetic, Numeric, User-definable Function, Pre-programmed																								
Power supply:	6.0V, DC: 4 dry batteries (Type UM-3, AA or R6)																								
Power consumption:	6.0V, DC: 0.13W																								
Operating time:	Approx. 50 hours on dry batteries (Type (UM-3, AA or R6))																								
Operating temperature:	0°C ~ 40°C (32°F ~ 104°F)																								
Dimensions:	195(W) x 86(D) x 25.5(H) mm 7-11/16"(W) x 3-3/8"(D) x 1"(H)																								
Weight:	Approx. 375g (0.83 lbs.) (with batteries)																								
Accessories:	Soft case, four dry batteries, two keyboard templates, name label and instruction manual																								
Options:	Printer/cassette interface (CE-150) Expansion memory module (Plug-in type, 4K Byte RAM CE-151)																								

TABLE OF CONTENTS

	Page
An Introductory Note	1
Operational Notes	2
PC-1500 Specifications	3
Table of Contents	4
0. Instant Programming	8
A. Example 1	8
B. Example 2	9
I. Getting Acquainted	11
A. ON and OFF keys	12
B. Alphabetic keys	12
C. Numeric keys and Arithmetic Operation keys	12
D. SHIFT key	12
E. Lower-case Letters and the SMALL key	12
F. The Display	13
G. The Cursor and the Prompt	13
H. Clear key	13
I. ENTER key	13
J. Error Messages	13
K. Battery Function Indication	14
II. Taking the Plunge	15
A. MODE key	15
B. Simple Calculations	15
C. Serial Calculations	16
D. Calculations with Negative Numbers	16
E. Compound Calculations	17
F. Use of Parentheses	17
G. Logical Comparisons and Inequalities	18
H. Editing Keys and Functions	19
H.1. Left Arrow/DElete Key	19
H.2. Right Arrow/INSert Key	20
H.3. Recall Function	21
I. Variables	21
J. We Pause	24
SUMMARY	24
III. The Mysterious (?) Art of Programming	26
Foreword	26
A. What Is A Program?	26
B. How Do I Program?	26
C. COMMANDS vs. STATEMENTS	27
D. Line Numbers	27
E. Program-line Review keys	28
F. A Closer Look At Some Old Friends	29
F.1. The NEW Command	29

F.2.	The LET Statement	29
F.3.	The PRINT Statement	29
G.	The PAUSE Statement	34
H.	The INPUT Statement	35
I.	Shortcuts and Helpful Hints	39
I.1.	Abbreviations	40
I.2.	Multiple Statements Using the Colon	41
J.	Error Correction in the PROgram Mode	42
K.	The LIST Command	43
L.	The More, the Merrier	43
L.1.	The END Statement	43
L.2.	RUN line-number	44
M.	Control Statements	44
N.	IF ... THEN	44
O.	GOTO	46
P.	FOR ... NEXT	51
Q.	WAIT	54
R.	READ, DATA, RESTORE	55
S.	REM	57
T.	GOSUB and RETURN	58
	Summary of PROgram Mode Editing Features	59
IV.	Advanced Calculations	60
A.	Scientific Notation	60
B.	Range of Calculations; Overflow, Underflow	62
C.	Root, Power, and Pi	62
D.	Angular Modes	64
E.	Trigonometric Functions	64
SIN, COS, TAN, ASN, ACS, ATN		65
F.	Logarithmic Functions	65
LN, LOG, EXP		65
G.	Angle Conversion	66
DEG, DMS		66
H.	Miscellaneous Functions	66
ABS, INT, SGN		66, 67
V.	Advanced Programming	67
A.	Arrays and Subscripted Variables	67
DIM		67
B.	More On Character Strings	70
B.1.	DIMensioning Strings	70
B.2.	Concatenation	70
B.3.	String Comparison	72
C.	Functions	72
C.1.	ASC	72
C.2.	CHRS	73
C.3.	INKEYS	74
C.4.	LEN	75
C.5.	LEFT\$	75
C.6.	MIDS	76
C.7.	RIGHTS	77
C.8.	RND	77
C.9.	RANDOM	78
C.10.	STRS	78

C.11.	STATUS	79
C.12.	TIME	79
C.13.	VAL	80
D.	PRINT USING	80
E.	Computed Control Transfer.	83
	ON GOTO, ON GOSUB, ON ERROR GOTO.	83, 84
F.	Display Programming	84
F.1.	BEEP	84
F.2.	CURSOR	85
F.3.	CLS	88
F.4.	GCURSOR	88
F.5.	GPRINT	91
F.6.	POINT	94
G.	Debugging ... & SINGLE STEPPING	96
	TRON, TROFF, Arrow keys	96
H.	Hexadecimal Numbers & Boolean Functions.	98
H.1.	Hexadecimal Numbers	98
H.2.	AND Function	98
H.3.	OR Function	99
H.4.	NOT Function	99
I.	Halting Program Execution	100
	STOP, CONT	100
J.	Mode Control	100
	LOCK, UNLOCK	100
VI.	Expanding the PC-1500	101
A.	Printer/Cassette Interface (CE-150)	101
A.1.	Connecting the Computer to the Interface	101
A.2.	Power (Recharging the Batteries)	103
A.3.	Connecting a Tape Recorder to the Interface	103
A.4.	Loading the Paper	105
A.5.	Replacing the Pens.	106
B.	Using a Cassette Recorder	108
B.1.	Tape Recorder Operation	108
B.2.	Saving Programs on Magnetic Tape (CSAVE)	109
B.3.	Loading Programs from Magnetic Tape (CLOAD, CLOAD?)	109
B.4.	Saving and Loading Data Using Magnetic Tape (PRINT#, INPUT#)	110
	<i>a* convention written variables</i>	
B.5.	Editing Programs on Magnetic Tape (MERGE)	111
B.6.	Chaining Programs. (CHAIN)	113
B.7.	Using Two Tape Recorders	114
C.	Using the Printer	116
C.1.	CE-150 Printer Specifications	116
C.2.	TEST Command	116
C.3.	Printing Calculations	117
C.4.	Printer Modes	118
C.5.	Listing Programs	118
C.6.	Programmable Printer Control	120
	CSIZE	120
	ROTATE	120
	COLOR	121
	LF	121

LPRINT	122
LCURSOR	123
TAB	124
SORGN	124
GLCURSOR	124
LINE	125
RLINE	127
VII. RESERVE Mode.	129
A. Defining and Selecting Reserve Keys	129
B. Identifying Reserve Keys	130
Reserve Key Identification Template.	131
VIII. Beginning Program Execution	132
A. The DEF Key	132
A.1. Running DEFInable Programs	132
A.2. Pre-Assigned Keywords.	132
A.3. The AREAD Statement	133
B. Automatic Program Initiation	134
ARUN	134
C. Comparison of Initiation Methods	134
IX. Appendices	137
A. Abbreviations	138
B. Battery Replacement	142
C. ASCII Character Code Chart	144
E. Error Messages	145
F. Further Reading	151
O. Order of Expression Evaluation	151
X. Command Comparison: PC-1211 vs. PC-1500	153
Z. Command Reference Table	155

Name label

Write your name on the attached name label and stick it on the back of the Computer.

INSTANT PROGRAMMING

(No Water Necessary)

This section is devoted exclusively to a select group of people (the authors included) whose inquisitiveness outweighs their patience (and perhaps their common sense). For those of you who absolutely must DO SOMETHING with this miracle of modern electronics, we present a simple programming exercise. (WARNING: The timid or faint-of-heart are instructed to proceed to Chapter 1; Getting Acquainted, for a more thorough and leisurely introduction to the SHARP PC-1500).

Before you proceed, one further caution is in order. It is important to follow all of the listed steps in the given order. Contrary to popular opinion, computers are not "super-brains" and do not have the average human's ability to "figure out" what you desire. The PC-1500 simply awaits your instructions and performs them. Are you ready? Good, let's begin.

Example 1

First, find the key marked "ON" in the upper right corner of the keyboard. Pressing this key will cause the sleeping electronic genie to awaken (don't expect a puff of smoke!). The display portion of the computer should be similar to the illustration below:



Press the **MODE** key (on the far right side) until the abbreviation PRO appears in the upper portion of the display. (If you press the key too many times, just press it again until the desired result is obtained). The SHARP PC-1500 is ready to accept the series of instructions which make up a computer program.

Enter the following keystroke sequence:

1 0 A = 1 ENTER

Notice that as you press the **ENTER** key, the computer will modify what you have typed in. The display should now look like this:



Note: Throughout this manual we will use Ø for the number zero, so that you can distinguish between the letter O and the number Ø.

Continue by pressing the following keys. Do not be alarmed when each line disappears as you type its successor.

2 Ø P A U S E A ENTER

3 Ø A = A + 2 ENTER

4 Ø G O T O 2 Ø ENTER

At this point, your first program is complete. Now you must tell the computer to "execute," or carry out, the instructions it contains. This process is known as "running" the program and is performed in the RUN mode (logical huh?). Press the **MODE** button once again and the letters PRO are replaced by the letters RUN at the top of the display.

One last step: type in the letters **R U N** and press **ENTER**.

Congratulations! Your first BASIC program is now running. Your instructions are being followed and the computer is busy listing all the positive odd numbers, in order.

"So," you say to yourself, "I'm a genius. But when will it stop?"

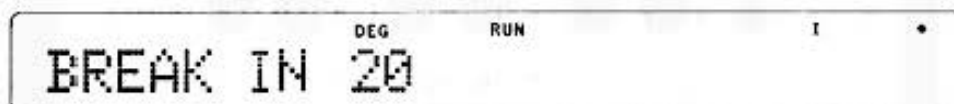
Well, . . . unfortunately, without your intervention or battery failure, this particular program will never finish. To see why, let's review our program:

```
10 A = 1
20 PAUSE A
30 A = A + 2
40 GOTO 20
```

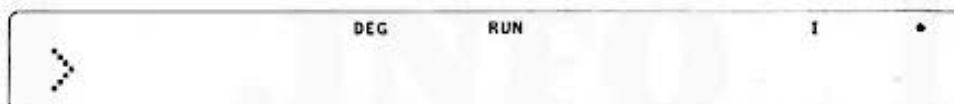
The effect of the line numbered 40 is to cause the computer to re-perform all lines after the one numbered 20. This includes line 40 which, of course, tells the computer to re-re-perform lines 20, 30, and 40 . . . and so on forever. This repetition without end is known in computer jargon as "looping".

Our program is stuck in an "infinite loop" and only you have the ability to stop this tragic expense of battery power. To do this, press the **ON** key. Since the PC-1500 is already on, you are actually selecting the BREAK function. Don't be alarmed. Despite what the name seems to indicate, this is not a self-destruct key. **As a matter of record, you cannot, in any way, hurt or damage the computer merely by pressing keys, so feel free to experiment!**

If you have pressed the BREAK key, a message similar to the following is visible in the display window:



This informs you which instruction was being "executed", or worked on, when you interrupted the computer. Press the BREAK key once more, and the computer awaits your next instruction:



For those of you who just remembered that you left the tap running at home, this is a good stopping point. (Before leaving, please press the **OFF** key to conserve batteries.) Others of you are already becoming programming addicts and will want to continue your education with our second example. (We hereby waive responsibility if you're late for dinner).

Example 2

To begin our second program it is necessary to enter the program mode by pressing the MODE key until the letters PRO (short for PROGRAM) replace the letters RUN at the top of the display. The PC-1500 will now let us submit a new program or modify an old program. Because our new program will not build on the instructions of our old program, we must clear those instructions from the computer's memory. To do this, type in the word NEW and press ENTER. After a pause the > character (called a prompt) will return.

Type the following keystrokes to enter the first line of the program:

1 0 I N P U T SHIFT " L I S T SPACE
S I Z E SHIFT / SHIFT " SHIFT + A ENTER

Notice that pressing the **SHIFT** key followed by a key which has another character inscribed above it, will enter the uppermost character. The shift key permits two characters to share the same button, and is sometimes called the "second function" key. Thus, in the first line of our program (line 10 above), a **SHIFT** keystroke followed by a **+** causes a **:** (semi colon) character to be entered. The entire line is stored in the computer as:

```

DEG          PRO          I          .
10: INPUT "LIST SIZE?":A
    
```

In this manual we will illustrate the selection of the second function character with the shift key and the character desired. For example, the line numbered 10 above will be shown as:

```

1 0 I N P U T SHIFT " L I S T SPACE
   S I Z E SHIFT ? SHIFT " SHIFT ; A ENTER
    
```

Complete the entry of our second program with the following keystrokes:

```

2 0 I F A SHIFT < = 0 G O T O 9 9 ENTER
3 0 F O R I = 1 T O A ENTER
4 0 P A U S E I SHIFT , I * I ENTER
5 0 N E X T I ENTER
9 9 B E E P 2 SHIFT : E N D ENTER
    
```

Our second program is now stored in the PC-1500's memory. Do you remember what must be done next? If you said "Run the program" you are well on the way to programming competency.

Return to the RUN mode (HINT: use the MODE key) and type in the word RUN. Press **ENTER** to begin "execution" (another word for running) of our second program.

Is the computer interrogating you in the following manner? (If not, return to the PROgram mode and re-check your typing).

```

DEG          RUN          I          .
LIST SIZE?_
    
```

Good show! Our program is asking the user (you) for information necessary to perform the task that we, as programmers, instructed it to perform. Recall the first line of our BASIC program (which you so kindly typed in for us);

```

DEG          PRO          I          .
10: INPUT "LIST SIZE?":A
    
```

The instructions within this line are currently being followed by the computer. The result is that the computer is waiting for you to "input" (type in) some information.

This program will print a list of numbers and their squares (the number multiplied by itself). First, however, it asks the user how many numbers and squares to print (LIST SIZE?). The user (you again) responds by typing a number and pressing (you guessed it!) **ENTER**.

Type **8** and press **ENTER**.

Watch closely as pairs of numbers appear briefly on the screen. The second number in the pair (on the right) will be the square of the first. Eight pairs of numbers will be displayed because that is what you asked for when you responded to the program's question.

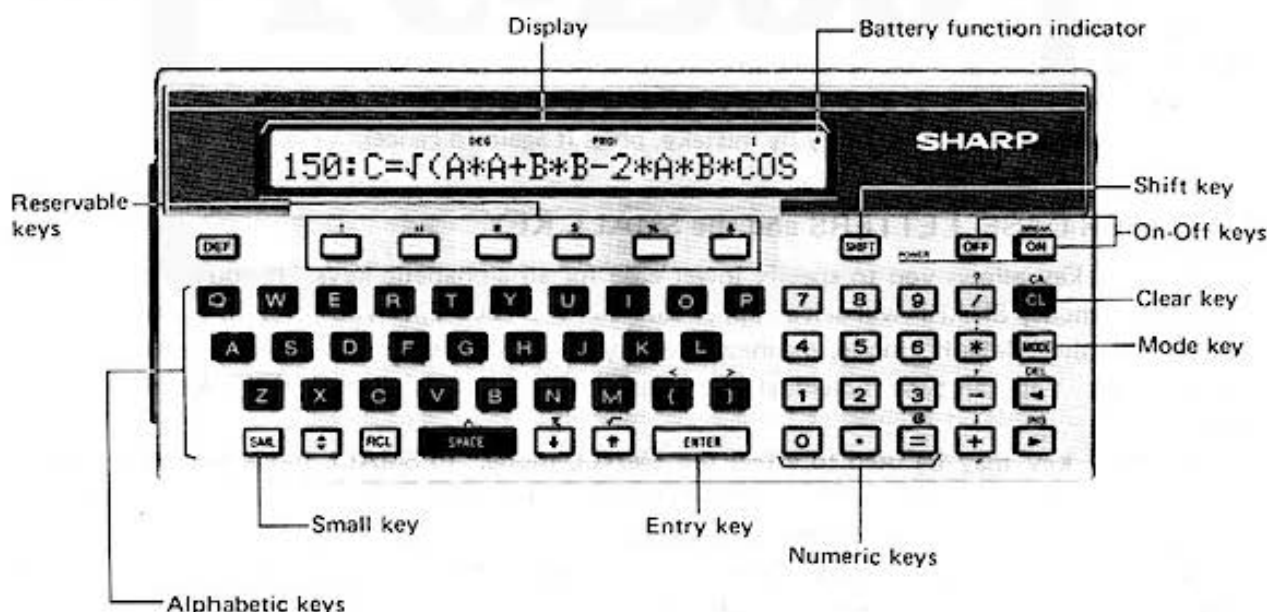
When the prompt returns, re-run the program (type RUN, press **ENTER**) and make a different response to the "LIST SIZE?" question. Run the program several times experimenting with different list sizes. You are experiencing one of the important advantages of programmable computers; they can perform a tedious task repeatedly, varying it slightly each time in response to the input.

Re-run the program once more but this time enter a zero for the list size. What happens? Yes, the program ends without having produced a list. Although this may seem odd, the PC-1500 is simply following our instructions.

This illustrates why the computer is such a powerful tool. It can be programmed to follow different sets of instructions and process the variety of information it is given. Because of the instructions in line 20, if the user's input is zero (or less) the computer skips over the computation of the list and goes to the end of the program. In effect, it has made a decision based on the user's request. As a programmer, you control what decisions are possible and when they are made. Thus, the full power of the computer is available to you to solve your specific problem in the manner you think is best.

I. GETTING ACQUAINTED

After you unpack your Sharp Pocket Computer (hereafter we'll call it SHARP) and admire your handsome new friend, you might wonder what you're staring at. Let's examine SHARP:



We will describe the display in a moment. First, even before you turn on SHARP, notice several important features of the keyboard:

A. ON AND OFF KEYS

Obviously these keys turn the power on and off. SHARP, to conserve its battery, will automatically shut off if nothing is keyed in for a period of about seven minutes, unless a program is being executed. Observe inscribed above the **ON** key the phrase BREAK. The **ON** key can be used to BREAK, or interrupt, the execution of a program. This function is described in more detail later in this manual.

B. ALPHABETIC KEYS

The Alphabetic keys allow the computer user (you) to give instructions and enter data. In addition, these keys may be used to designate "storage areas" within the computer's memory into and from which you will be able to save or retrieve data. This use will be covered in the section on variables. Lower case letters are available through the use of the **SHIFT** and **SML** (SMALL) keys (described below).

C. NUMERIC KEYS and ARITHMETIC OPERATION KEYS

With these you enter numbers for calculation. The **+**, **-**, *****, and **/** keys tell SHARP to add, subtract, multiply and divide, respectively. The **E** key allows the entry of numbers in "scientific notation". The use of this notation and other sophisticated functions are described in the chapter on Advanced Calculations.

D. SHIFT

This key delivers the secondary functions inscribed above many non-alphabetic keys. For instance, to type a colon, press **SHIFT** and then the ***** (asterisk) key. When the SHIFT key is followed by an alphabetic key, the lower-case letter is displayed. (NOTE: In the SMALL mode, the SHIFT preceding the alphabetic key will produce an upper-case letter).

When the SHIFT key is activated, the word SHIFT appears in the upper left corner of the display. The shift mode is only active for one keystroke at a time.

The six keys at the top of the keyboard, directly below the display window, are called **RESERVABLE KEYS**. Using the shift in a manner we will describe later on, you can assign frequently typed commands or other operations to these keys.

NOTE: If you press the **SHIFT** key by mistake, press it again to cancel.

E. LOWER-CASE LETTERS and the SMALL KEY

The **SML** key allows you to specify lower case for all alphabetic keys. If you do not specify the SMALL mode, SHARP will select upper-case for you each time you press an alphabetic key. (We call this the "default" mode, meaning the way in which the machine operates unless you tell it otherwise.) You can type individual lower-case letters by pressing the **SHIFT** key before the letter.

The **SML** key may be used to effect the SMALL mode. In SMALL mode lowercase letters result from pressing an alphabetic key and individual upper-case letters are displayed by pressing the **SHIFT** key first. When the computer is in SMALL mode, the word SMALL will appear on the top portion of the display window. Once you have placed the computer in SMALL mode, it will remain in this mode until you press the **SML** key again.

NOTE: We recommend that you restrict your use of lower-case for the moment. This is because SHARP only recognizes instructions in upper-case letters. When you learn to program, you will find lower-case letters handy.

F. THE DISPLAY

Press ON. The "glass window" part of the computer is called the "display." It looks something like this:



On the display you should see an > (called a "prompt"), several words or abbreviations, and a dot (indicating that the battery is functioning). Do not be concerned if the specific abbreviations appearing on your display are not those of our illustration. These symbols change as you operate SHARP.

G. The CURSOR and the PROMPT

At the far left of the display, find the prompt symbol (>); it prompts you to talk to SHARP. When the "prompt" appears it means SHARP has no immediate plans and awaits your bidding. Type a letter of your choice. It replaces > at the left of the display, while to the right of your letter appears a _ (underline symbol). This is a cursor. As you press each key, the cursor inches its way across the display, indicating where the next symbol you type will appear. Type your name and note the movement of the cursor.

If you type more than 25 characters, the limit that can be displayed at one time the entire line shifts left. (Try it!). Characters "pushed off" the screen are not lost: they remain in SHARP as part of the typed line, up to a maximum of 80 characters for any single line. We will see how to "recall" and how to change these characters in a later section.

H. CLEAR

(The red **CL** key in the upper right corner)

Push this button and you **C**lear the display of its contents. Use it to erase the characters that you just typed in. Notice that the prompt has returned, indicating that the computer is again waiting for your commands.

The clear key is also used to cancel an incorrect command. (See the section on Error messages below).

I. ENTER

As you type into the computer, the letters or numbers appear on the display. SHARP will take **NO ACTION**, however, until you signal that you have finished typing (after all, it can't read your mind). This is done by pressing the ENTER key after your other keystrokes. At this point, the computer will scan the characters you have typed for correct form. Certain errors, but by no means all errors, will cause your input to be rejected.

REMEMBER: Press the **ENTER** key each time you wish to enter an instruction or item of data into the machine.

J. ERROR MESSAGES

Press the following keys:



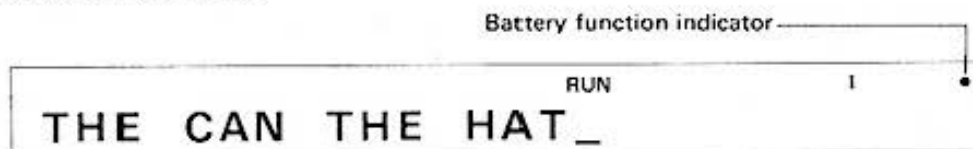
Now press **ENTER**.

The answer should be displayed. Three, right? No? You get "ERROR 1" as your answer? Is your computer defective? Never! There's a mistake in the form of the command. "ERROR

1" is an error code which tells you that you have incorrectly performed a calculation. (For the curious, a complete list of error messages is included as an appendix). We'll take the blame for this first error and in later sections we'll show you how to use other keys to correct an erroneous command. For now, you may use the CLear key to erase the error message.

K. BATTERY FUNCTION INDICATION

When this dot disappears, the time has come to rejuvenate SHARP by replacing its batteries. See Appendix for instructions.



QUIZ – CHAPTER

Match each item in column A with one from column B. (Answers on bottom of page).

HINT: Column B contains some silly possibilities.

A

- a) SHIFT key
- b) The Display
- c) The Cursor
- d) ENTER key
- e) BREAK key
- f) CLEAR key
- g) The Prompt
- h) The red HERRING key

B

- 1) A malcontent given to constant swearing.
- 2) A key which cleans the display window and erases any results from previous computations.
- 3) A key which selects one of 2 characters which share the same key.
- 4) Ain't no such key.
- 5) A character, appearing in the display, which informs the user that the computer awaits his/her command.
- 6) The key which signals the computer that the User is finished typing.
- 7) A key which interrupts a program in the process of computing.
- 8) The name of a rare tropical fish.
- 9) The character, on the display, which indicates where the next typed character will appear.
- 10) The glass window on which information appears.

Answers: A-3, B-10, C-9, D-6, E-7, F-2, G-5, H-4

II. TAKING THE PLUNGE...

Actually, this chapter is probably misnamed; learning to use the PC-1500 is not nearly as shocking as plunging into a pool of water. However, it does require that you approach the computer without fear. As we have said before, **you cannot hurt or damage the computer merely by pressing the keys.**

In this chapter we will explore the fundamental features of SHARP upon which programs and more advanced calculations are based. Take the time necessary to work through the examples in each section. A good understanding of the basics will allow you to exploit this machine's full potential.

Although we do not recommend it, if you feel that you are sufficiently advanced, you may skip ahead to the summary at the end of this chapter.

A. MODE

Let's begin with a key we have been ignoring until now. At the right of the keyboard find the very important button labelled MODE. Press it repeatedly. Notice, each time you press the key, the changes in the abbreviations at the upper right. Changing the machine's mode in this way can be likened to shifting gears on a car. As each new mode or gear is engaged, machinery which appears outwardly unchanged performs differently. Like a car, SHARP must be placed in the proper mode to function according to plan. And, also like a car, when you attempt to operate SHARP in an improper "gear", the computer will quickly notify you of your mistake.

By repeatedly pressing the MODE button you will be introduced to the two most important of SHARP's three modes: RUN and PROgram. A third mode, RESERVE, is activated by pressing the SHIFT key before the MODE key. Later chapters of this manual will describe how each of these modes contributes to the smooth running of SHARP. For now, remember that in order to use SHARP as a calculator, you must be in the RUN mode. Later, in the PROgram mode, you will write and change programs. With the RESERVE mode you can assign frequently used commands to a single key. This is explained in greater detail in Chapter 7.

Note that the first time you turn the computer on after inserting batteries, it will settle into the PRO mode. At other times the computer will come on in the mode in which it last operated before being shut off.

B. SIMPLE CALCULATIONS

With SHARP set in the RUN mode, let's test fundamental mathematical computations. It is necessary before each calculation to press **[CL]**. This clears the display of any previous data which might interfere with a new calculation. Find answers to the following simple problems:

<u>Input</u>	<u>Display</u>
5 + 2 ENTER	7
5 - 2 ENTER	3
5 / 2 ENTER	2.5
5 * 2 ENTER	10

NOTE: Do not type an equal sign. Remember from Chapter I that it is the ENTER key which informs SHARP that you have finished typing and wish to have your command or calculation performed.

C. SERIAL CALCULATIONS

You can utilize an answer from one calculation in a following calculation by proceeding directly to the second calculation. (Do not press CL between calculations). Thus, if you are balancing a checkbook, you will operate SHARP in this fashion:

Input	Display
161.16 - 47.50 <input type="button" value="ENTER"/>	113.66
- 12.33	113.66 - 12.33
<input type="button" value="ENTER"/>	101.33

As you will observe, the result of the first calculation jumps to the left of the display as you begin the second calculation.

NOTE: DO NOT type dollar signs or commas when entering numbers in calculations. These symbols have special meaning in the BASIC language (and therefore to SHARP).

Other operations can be performed similarly. Try these:

Input	Display
5 + 3 <input type="button" value="ENTER"/>	8
8 + 3 - 1 <input type="button" value="ENTER"/>	10
10 * 3 - 1 <input type="button" value="ENTER"/>	29
29 / 3 - 1 <input type="button" value="ENTER"/>	8.666666667

D. CALCULATIONS WITH NEGATIVE NUMBERS

Imagine you have presented your computer science teacher, Mr. Onoff, with two apples. You have remaining an inventory of five apples, and you wonder, "How many apples would I now have if I hadn't been so generous to Mr. Onoff?". To find the answer, you might imagine the subtraction from inventory negated; that is, you would subtract the subtraction. Type into SHARP: 5 -- 2 . Seven would be your hypothetical inventory. Try these similar calculations with negatively signed numbers:

5 * - 2
 5 + - 2
 5 / - 2
 - 5 - 2.3
 - 5 + - 2
 - 5 / - 2

REMEMBER: To press **CL** between calculations to CLear previous results.

E. COMPOUND CALCULATIONS

You can string together a sequence of calculations before asking SHARP for an answer. For example, you and two friends, Hob and Nob, wish to share 5 apples twice a day for a week. How many apples should you buy to last the week? 5 apples \div 3 friends \times 2 per day \times 7 days:

Keystrokes	Display
5 \div 3 \times 2 \times 7 ENTER	23.33333333

(Buy 24 apples and a parrot, to whom you can feed the extra 1/3 apple). Run the following calculations, (but this time invent your own stories as to what they represent):

Input	Display
$5 \times 2 - 3$ ENTER	6.325
$5 / 3 \times 6.2 + 7 - 47$ ENTER	-29.66666667

F. USE OF PARENTHESES

A problem which emerges as we investigate compound calculations is that of priority. For instance, the expression $5 - 3 / 4$ can be read two ways: (5 minus 3) divided by 4, in which case the answer is .5, or 5 minus (3 divided by 4), in which case the answer is 4.25.

Located in the first row of keys are the parentheses which you can use to clarify such ambiguities. Run the following calculations:

Input	Display
$5 - 3 / 4$ ENTER	4.25
$5 - (3 / 4)$ ENTER	4.25
$(5 - 3) / 4$ ENTER	0.5

SHARP is predisposed (has "default" priorities) to perform some calculations before others (for a complete listing of the order in which SHARP calculates, see Appendix 0). Division and multiplication will be carried out before subtraction and addition, unless parentheses are used to direct a different ordering. Thus SHARP is built to interpret the first equation above as being equal to the second rather than to the third. To make sure that the answer SHARP gives you is the one you need, use parentheses to indicate the proper order in which it should perform calculations.

SHARP can interpret several layers of parentheses as in this problem:

Input	Display
$((6 - 4) / 2) * ((3 - 1) / 4) * 6$ ENTER	3

The equations within the innermost set of parentheses will always be calculated first.

REMEMBER: When in doubt, use parentheses to clarify the order of your arithmetic operations.

G. LOGICAL COMPARISONS AND INEQUALITIES

SHARP will allow you to compare two values or equations and will indicate to you the result of the comparison. This ability is basic to designing programs which make decisions. The manner in which this is done will be recognizable to students of the "New Math" as an inequality (don't despair if you weren't raised on New Math; the author wasn't either).

An inequality may be thought of as a comparison which is either true or false. For instance, the statement "six divided by three is equal to two" is a comparison which happens to be true. On the other hand, the statement, "six divided by three is greater than five" is a false comparison.

Computers and mathematicians use the following symbols for the possible types of comparisons:

<	less than
>	greater than
=	equal to
<=	less than OR equal to
>=	greater than OR equal to
<>	not equal to

Thus, we can restate the above inequalities symbolically, as: $6 / 3 = 2$ and $6 / 3 > 5$ respectively.

Given an inequality, SHARP will determine whether the comparison is true or false. In keeping with current computer design practice, SHARP will indicate a true statement with a 1 and a false statement with a 0. For example if you type:

$$6 / 3 = 2$$

SHARP will respond with a 1 (for true). Typing the sequence:

$$6 / 3 > 5$$

will elicit a response of 0 (for false).

Try the following tests of SHARP's judgment (be sure you are in RUN mode):

Keystrokes	Result
4 SHIFT > 5 ENTER	0
5 SHIFT > 4 ENTER	1
2 * 2 SHIFT < 2 * 3 ENTER	1
1 8 SHIFT < SHIFT > 1 8	0
2 = 2 ENTER	1
5 SHIFT < = 5 ENTER	1
2 5 * 2 SHIFT > = 1 0 0 / 3 + 4 ENTER	1

As you may have observed, the equations which are compared may be as complex as necessary (the only limit is the restriction of 80 symbols to a line).

Here is a very simple problem to illustrate a practical use of inequalities:

Nob enters Nails 'n Stuff hardware store. He finds that cement comes in 4, 8, and 12 pound bags. The money he has brought with him will buy either two 12 pound bags or three 4 pound and one 8 pound bag. He wonders in which case he will get more cement. He asks SHARP if

$$2 * 12 > (3 * 4) + 8$$

SHARP replies 1; Nob buys two 12 pound bags.

Experiment with inequalities applying them to your own problems.

H. EDITING KEYS AND FUNCTIONS

Most humans (except us genius-types) have a tendency to make mistakes. Recognizing human fallibility, the designers of the PC-1500 have incorporated several features which facilitate changes and corrections.

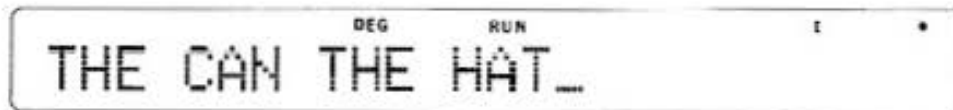
H.1. Left Arrow/DELeTe key ^{DEL} ◀

By now, several of you may have discovered the Left Arrow key at the lower right side of the keyboard. This key acts like the backspace key on most modern typewriters; it allows you to move back over previously typed characters.

In the RUN mode, starting with a clear display (i.e. with the prompt showing) type in the following characters:

T H E SPACE C A N SPACE T H E
SPACE H A T

The display should appear as follows:



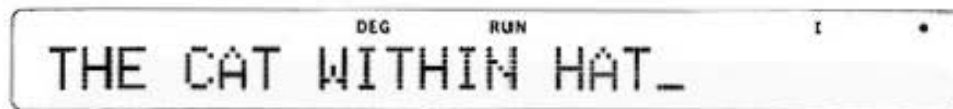
Press the Left Arrow key once and notice the manner in which the cursor changes. This "Flashing grid" form of the cursor allows you to see the character at the current cursor position. Press the Left Arrow key repeatedly (or hold it down) until cursor is positioned over the N. (If you accidentally pass the N, move the cursor forward again with the Right Arrow key).

Type a T. The T replaces the N and the cursor moves forward. This should not be surprising if you remember that the cursor indicates where the next character will be placed.

Type these characters:

`[SPACE] [W] [I] [T] [H] [I] [N] [SPACE] [H] [A] [T]`

Now displayed is:



As the display illustrates, characters which have been typed over are gone forever.

In addition to backspacing, the Left Arrow key has a second use, the DELEte function whose abbreviation is inscribed above the key. To delete a character, place the cursor on the doomed character and press `[SHIFT] [DEL]`.

Let's try it: move the cursor back to the W and press the sequence `[SHIFT] [DEL]` four times. The display now shows:



H.2. Right Arrow/INSert key `[INS]`

As we have already seen, (sorry to ruin the surprise) the Right Arrow key moves the cursor forward without erasing characters. Like the Left Arrow, the cursor will move repeatedly if the key is held down.

Move the cursor to the end of the line.

The second function for the Right Arrow key gives us the ability to INSert characters within a typed line. This feature is handy for those of us who tend to forget little things (like letters and words).

Move the cursor back to the H in HAT. Type the sequence `SHIFT INS` four times and watch the present characters shift to the right. The new box-like characters which have appeared can be thought of as "placeholders." These "place holders" can be filled with new information.

Type:

`[T] [H] [E] [SPACE]`

Presto! You have just inserted a word. (I wish my typewriter could do this).

H.3. RECALL FUNCTION

Up to now we have been discussing ways to correct statements which had not yet been entered (that is to say, you had not pressed ENTER after your keystrokes). Once you press ENTER, however, SHARP immediately attempts to perform your calculation. If the computer is successful, the result replaces your equation on the display. The equation, however, is not lost and may be recalled (redisplayed) by pressing either the Left Arrow or the Right Arrow key.

Clear the display and type in the equation of your choice. Press ENTER to compute the result. Recall your equation. Note that to see the result again, it is necessary to re-compute the equation (using the **ENTER** key).

Fortunately, the Recall function will also work if SHARP encounters an error while trying to evaluate (make sense of) your input. This allows you to recall and correct the erroneous equation using any of the editing features you have just learned.

To test this, enter the following incorrectly phrased expression:

45 * 63 / * 2 **ENTER**
 ↑

When the error message appears (ERROR 1) press either of the Arrow keys to recall the expression. You should see the Flashing-grid cursor positioned over the second multiplication symbol (asterisk). This is SHARP's way of indicating the point at which it became puzzled (and for good reason in this case). From here you may proceed to make any correction you deem appropriate.

I. VARIABLES

The ability to work abstractly through variables is one of SHARP's most powerful features. Variables may be thought of as a group of little boxes, each of which may be filled with a single item of data such as a number or a name.

You might remember variables from high school algebra. You learned (at least you were taught) that if $5A = 30$ then A must be 6, but if $5A = 35$ then A is equal to 7. The A in this case is a variable which holds a single number (not always the same and therefore a "varying" number) called the "value" of the variable. The ability to use a letter (such as A) in an equation instead of a specific number is very useful. Let's see why with the following example:

General Duffer has his heart set on purchasing a set of Deluxo chrome-plated golf clubs. The set consists of 5 clubs at \$12 each, a bag at \$21.99 and three balls as \$1.56 each. The local Army PX is offering this set at a 10% discount but they have an \$8 polishing and delivery charge. Five Finger Discount Bernie's Golf Goods is offering the same set with a 5% discount and free delivery.

To compute which is the better deal, the General decides to enlist the aid of his SHARP PC-1500. The General calculates the basic cost of the set and saves this result into a variable G (for golf, not general):

$G = (5 * 12) + 21.99 + (3 * 1.56)$ **ENTER**

The result is displayed and can be recalled from storage by entering the name of the variable;

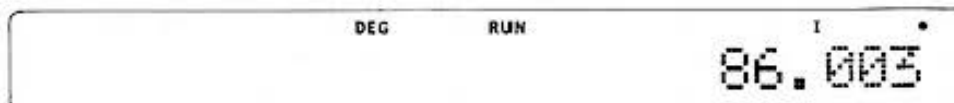
G **ENTER**

DEG RUN

1
86.67

Now the General calculates the actual purchase price at the PX:

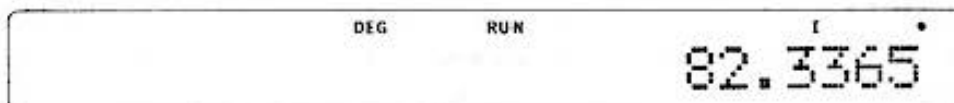
$$G - (G * .10) + 8 \text{ ENTER}$$



A calculator display showing the result of the calculation. The display is rectangular with a dark background and white text. It shows "DEG" and "RUN" on the left side, and "86.003" on the right side. There is a small "I" and a dot above the "3".

and the purchase price at Discount Bernie's:

$$G - (G * .05) \text{ ENTER}$$



A calculator display showing the result of the calculation. The display is rectangular with a dark background and white text. It shows "DEG" and "RUN" on the left side, and "82.3365" on the right side. There is a small "I" and a dot above the "5".

Clearly, Discount Bernie's has the better bargain. (Clever readers may have noticed that the General could have worked this problem using SHARP's unique recall feature (see Chapter 2), though not quite as easily).

There are several lessons to be learned from the preceding example . . .
Observe that the General's first calculation had the following form:

$$\text{variable-name} = \text{expression}$$

An instruction with this form is known as an **Assignment Statement**. It is important not to confuse an Assignment statement with an inequality. Unlike Assignment statements, inequalities are not used separately, but form parts of other programming instructions.

The Assignment statement instructs SHARP to store the result, obtained by calculating the expression, in the memory location associated with the given variable name. Thereafter, using the name of the variable (G in our example) is just like using the result itself. Notice also that a variable may be used as many times as needed in the same calculation.

Variables can be used for other neat tricks, too. The value of one variable can be assigned to another, as in the statement:

$$H = G$$

which will copy our previous result into H. G has not been altered; the same result is now stored in two different variables.

Variables which hold numbers can be incremented or decremented in one statement as in this example:

$$G = G + 5$$

This instruction will cause SHARP to recall the value of G, add 5 to that value, and store the new value back into G. This capability is useful for all kinds of calculations:

The cost of a widget is stored in variable X. Assuming a 6.5% sales tax, what is the purchase price of the widget? :

$$X = X * 1.065$$

Of course, we could have written this example as $P = X * 1.065$ in order to leave X unchanged. If we had stored the tax rate in a variable T, we could have said $X = X * T$ or $P = X * T$.

Until now, we have been using single letters as variable names. This provided us with 26 variables (A through Z). In actuality, SHARP allows the use of over 950 variable names for single numbers. An additional group of over 950 variables may be used to store up to 16 characters each. Finally (as if that weren't enough), using more advanced techniques, users may create variables which hold as many numbers or characters as desired, limited only by the amount of memory available in the computer.

The naming-scheme for variables is simple and easy to learn. Names of numeric variables (ones used to hold numbers) may be chosen using the following rules:

- The name may be a letter; A through Z.
- The name may be a letter, followed by a single digit (0 through 9) or by another letter.

Thus, the following are valid names for numeric variables:

S, Q1, TX, MM, Z9, R0, E.

NOTE: Due to conflicts with abbreviations which have other meanings in the BASIC language, SHARP does not allow the use of these variable names: LF, IF, LN, PI, TO.

Names of character variables (those used to hold characters) follow the same rules as above **except** that the name ends with a \$ (dollar sign). The \$ alerts SHARP to the fact that the variable holds character information. The following are examples of valid character variable names:

T\$, P2\$, T7\$, AAS\$, YR\$, X\$, ZH\$, B5\$.

NOTE: Due to conflicts with words which are part of the BASIC language, SHARP does not allow the use of these variable names: LF\$, IF\$, LN\$, PI\$, TOS\$.

It is important to understand that a variable A and a variable A\$ are **two different variables**, the first one can only hold a number and the second one can only hold characters. As we shall see later, SHARP's BASIC includes instructions to convert characters to numbers and numbers to characters.

To store characters in character variables we use a variation of our friend the Assignment Statement:

```
character-variable-name = "characters"
```

As an example type the following:

```
D$ = "DAVY JONES"
```

Now recall the contents of D\$ by typing

```
DAVY JONES
```


Notice that the space between Y and J was stored and that the " (double quote) characters were not. The double quotes serve as "delimiters," a guide to indicate what other characters will be stored. Any character (including spaces), except the double quote itself, may be stored. This sequence of characters enclosed by double quotes is called a character "string" and character variables are often referred to as "string variables." Each character variable can hold up to 16 characters. Some caution must be used in order not to exceed this limit or information will be lost. This is demonstrated by the following assignment which unintentionally normalizes John's eating habits. Type:

F\$ = "JOHN EATS BUTTERFLIES"

Now recall the information: [F] [\$] [ENTER]. Some change!

One final point to remember about variables: they have the memory of an elephant. Information remains stored in a variable until:

- 1) Another Assignment statement is executed for the same variable.
- 2) A NEW or CLEAR command is given.
- 3) A program is run using the RUN command.
- 4) The computer's batteries are changed.

The value of a variable is even retained when the power is turned off! Test this by turning SHARP off, then on. In RUN mode, ask for the value of G (Type [G] [ENTER]). Impressive, eh? As you begin to program in the next chapter, you will find variables are indispensable.

J. We Pause. . . (without station identification)


Congratulations! If you have persevered to this point, you now know the fundamentals of the SHARP PC-1500 well enough to perform a wide variety of calculations. Because the PC-1500 is so versatile, each reader will find many uses for it in his own fields of interest. No matter what your application is, however, you will eventually want to learn programming in order to fully exploit the power of this amazing machine.

At this point you have a choice. For those whose hearts have quickened, whose breathing is more rapid at the thought of programming (and for anyone not yet asleep), we request that you to go back and read Chapter 0, integrating your new knowledge with the information there. Then proceed to Chapter 3.

Other readers, who have an immediate need to use such features as scientific notation and trigonometric functions, may proceed directly to the chapter on Advanced Calculations.

SUMMARY

- 1) **Modes** — The SHARP PC-1500 operates in one of three different modes: RUN, PROgram, and RESERVE. Each change in mode causes a slightly different change in internal function, analogous to shifting gears in a car. The RUN mode is for calculations and is the mode in which you must "run" (execute) programs. In the PROgram mode, all writing and editing (additions and corrections) of programs is performed.
- 2) **Calculations** — SHARP performs common arithmetic calculations in the RUN mode. The CL (clear) key should be pressed before each calculation to clear the results of a previous calculation. **NOT** pressing the CLear key between calculations allows a series of calculations utilizing the result of the immediately preceding calculation.

- 3) **Special Characters** – The \$ (dollar sign) and the , (comma) have special meanings in BASIC and may not occur as part of a number within a calculation.
- 4) **Negative Numbers** – are denoted by preceding the number with a – (minus sign) as, for example: $-5 + 2 = > -3$
- 5) **Compound Calculations** – follow common algebraic laws. Parentheses are used to indicate the correct meaning of a given expression. Complete information on the order of expression evaluation is given in Appendix 0.
- 6) **Inequalities** – Inequalities using the symbols $<$, $>$, $<=$, $>=$, and $<>$ (for less than, greater than, less than or equal to, greater than or equal to, and not equal to) will return 0 for FALSE and 1 for TRUE.
- 7) **Left Arrow key** – The Left Arrow key  acts as a non-destructive cursor backspace. Holding this key will cause automatic repetition. The Underline cursor changes to a Flashing grid when placed over a previously typed character. Pressing SHIFT followed by the Left Arrow key invokes the DELETE function, which deletes the character on which the cursor is positioned.
- 8) **Right Arrow key** – The Right Arrow key moves the cursor forward non-destructively. Holding this key down will cause automatic repetition. The sequence SHIFT, Right Arrow will invoke the INSERT function; inserting a “place holder” character at the current cursor position. This character can then be overwritten with the information to be inserted.
- 9) **Recall function** – After ENTER is pressed, and the result of a calculation is displayed, the original equation may be recalled by pressing either the Left Arrow or the Right Arrow key. At this point, changes may be made and the modified equation re-entered. The Recall function will also work for any non-programmed expression which produces an error. In this case, the cursor will be positioned at the point of error detection.
- 10) **The Use of Variables** in the RUN mode greatly increases the PC-1500's computing power and provides brevity in complex expressions.
- 11) **Assignment Statements** using the forms:
variable-name = expression
character-variable-name = “characters”
allow the storage of a single number or a string of up to 16 characters, respectively. Any printable character may be used within the string except a double quote.
- 12) **Variable Names** for numeric variables are:
 1. A letter (A through Z)
 2. A letter followed by a digit (0 through 9) or another letter.Character variable names follow the same rules with the addition of the \$ (dollar sign) as a suffix to all names.
- 13) **Exceptions** – The following are exceptions to the naming scheme and may not be used as variable names: IF, LF, LN, PI, TO, IF\$, LF\$, LN\$, PI\$, TO\$.
- 14) **Life-span of Information** – Information within variables is retained until:
 - 1) A CLEAR or NEW command is given.
 - 2) A program is run using the RUN command.
 - 3) Another Assignment statement is executed for the same variable.
 - 4) The computer's batteries are changed.Turning the computer off does not affect values stored in variables.

III. THE MYSTERIOUS (?) ART OF PROGRAMMING

The art of programming has been needlessly shrouded in a veil of mystery for so long that most people associate it with wizardry or mathematical genius. The fact is that no special talent for pulling rabbits out of hats is required. Nor is it necessary that you be adept at solving partial differential equations. Your greatest assets will be your patience, your logical reasoning abilities, your attention to detail, and your eagerness to learn. A willingness to accept challenges is also useful (we won't kid you: at times programming is very challenging, that's the fun of it).

Programming is an art, and as such requires a little skill, a little training, and a lot of practice. It is not our intent in this manual to make a seasoned programmer out of you. We will familiarize you with the basic operations and concepts of programming. To be a competent programmer requires more, just as good driving involves more than knowing how to steer and shift gears.

Many good books on programming already exist and we strongly urge you to patronize your local computer dealer and library. Several good books on programming in general and the BASIC language in particular, are listed in Appendix F.

A. What Is A Program?

You may be surprised to discover that a program is just a set of instructions that the computer follows one at a time. These instructions must be given to the computer in a language it "understands". The SHARP PC-1500 "speaks" a dialect of BASIC, a widely used and very popular programming language. Like other languages, BASIC has a special vocabulary and grammar rules which are combined to form statements. If you speak to SHARP "ungrammatically", or in unfamiliar vocabulary, the computer will alert you to your error. But it is not difficult to correctly instruct SHARP. The BASIC language was originally developed to teach programming principles and many of its statements contain English words and other familiar symbols.

B. How Do I Program?

As you use SHARP to program, you will follow a certain routine. The instructions which make up a program are entered in the PROgram mode. These instructions are known as "statements" in the BASIC language. To begin execution of these statements, it is necessary to switch to the RUN mode, and then to instruct SHARP to proceed by typing the RUN command. For you "experts", who already have the two programs of Chapter 0 under your belt, this will be familiar. For those who are peeking ahead, let's try entering and running a program:

Switch to the PROgram mode and issue the BASIC "command" (more on commands vs. statements later):

N **E** **W** **ENTER**

This will erase any previous statements which may be left in memory. Type the following line:

Program Listing:

```
10 PRINT "GOOD SHOW!"
```

Keystrokes:

1 **O** **P** **R** **I** **N** **T** **SHIFT** **"** **G** **O** **O** **D** **SPACE**
S **H** **O** **W** **SHIFT** **:** **SHIFT** **"** **ENTER**

Our one-line program is complete. Change to the RUN mode and type:

R **U** **N** **ENTER**

This command instructs SHARP to begin processing the statements (or, in this case, the statement) in our program. Following orders, SHARP prints:

```
DEG      RUN      I      •
GOOD SHOW!
```

on the display. (Press ENTER to inform SHARP when you are through reading).

To make any additions, changes or deletions to our program we must return to the PROgram mode. If our program had contained an error and had not completed successfully, it would have been necessary to return to the PROgram mode to correct the offending statement. Thus, work on programs is accomplished in the PROgram mode, while the execution and testing of programs is performed in the RUN mode.

C. COMMANDS VS. STATEMENTS

You may have noticed in the previous example that we communicated our desires to SHARP by two different methods. The instructions NEW and RUN were performed immediately after we pressed ENTER. This type of instruction is known as a **command**. The PRINT instruction, on the other hand, was somewhat different. It was entered in the PROgram mode, was preceded by a number (10), and was not executed immediately. This type of instruction is known as a **statement**.

In some sense, the commands tell SHARP what to do with the statements. For example, the NEW command will erase all currently saved statements. It is important to remember that commands may not be used within a program whereas statements almost (but not quite) always are grouped to form a program.

D. LINE NUMBERS

A BASIC program consists of a series of numbered lines each containing one or more statements. The "line numbers" are used by SHARP to maintain the correct sequence during execution but do not become part of the output when the program is run. Statements may be entered in any order but are processed by the computer sequentially by line number (subject to modification, as we shall see later).

As a demonstration, let's add a statement to our previous one-line program. Switch to the PROgram mode and type:

Keystrokes:

5 **P** **R** **I** **N** **T** **SHIFT** **"** **J** **O** **L** **L** **Y** **SPACE**
SHIFT ***** **SHIFT** **:** **ENTER**

Now run the revised program. What happens? Press **ENTER** after "JOLLY" appears.

Notice in this example both what SHARP has done for you and what you have had to do for yourself. SHARP has arranged and executed lines 5 and 10 in the proper order. Yet, to nudge SHARP on from line 5 to 10, to see the result of line 10, you have to press ENTER between outputs.

Although a line number may be any number from 1 to 65,279, we strongly advise you to number your lines by increments of 10 (i.e. 10, 20, 30, ... etc.). This allows you to insert up to 9 statements between your current statements (11 through 19, for instance). Some programmers recommend an even greater gap; numbering by 20's. Although, with careful program design and writing, you will rarely need to insert statements, do not count on this! It is far easier to number by 10's now than to renumber many statements later.

Remember that no two lines may have the same line number. If this condition should occur, the oldest line (the one entered first) will be lost. This feature can be exploited to delete unwanted lines merely by keying the line number of the line to be deleted and pressing **ENTER**. Thus, an empty, yet numbered, line will effectively delete an existing line with the same number.

If duplicate line numbers are used unintentionally, however, trouble will result. To demonstrate this, enter the following line (in PROgram mode, of course):

Keystrokes:

1 **0** **P** **R** **I** **N** **T** **SHIFT** **"** **A** **W** **F** **U** **L**
SHIFT **"** **ENTER**

Now run the program in the same way you did before. It is "JOLLY AWFUL" to lose a program line, isn't it? And since line loss can lead to some very subtle errors, you should exercise caution in writing your programs.

E. PROGRAM-LINE REVIEW keys

"But", you might ask yourself, "How can I remember what lines I have entered?". Fear not intrepid programmer! This need to review has been anticipated and provided for with the **↑** (Up Arrow) and **↓** (Down Arrow) keys. You may think of these as the Program-line Review keys. By pressing the appropriate key (in PROgram mode), one may "move" up or down through the lines of the current program (a process known appropriately as "scrolling").

Switch to the PROgram mode and use the Down Arrow key to review the lines of our program in ascending order. Now use the Up Arrow key to move back to the top of the program (line 10). Note that if either Arrow key is held down, successive lines will be displayed automatically. (Unfortunately, this feature is not easily seen with a two line program).

Once you have reached a given line using the Program-line Review keys, you may then proceed to edit that line using the Right or Left Arrow keys. You will be delighted (relieved?) to discover that the operation of these keys in the PROgram mode is identical to their operation in the RUN mode (see Chapter 2). The DELeTe and INSert functions are also available for use in statement editing, and are invoked in the same manner as before.

NOTE: That after any changes are made to a program line, you must press **ENTER** in order to affect those changes. DO NOT use the Up or Down Arrow keys to move on to the next adjacent line without pressing **ENTER** or any editing you may have performed will be lost.

F. A Closer Look At Some Old Friends

Now that we know how to enter, run, and edit programs, we shall expand our vocabulary of useful statements and commands. Let's begin by examining some old friends: the NEW command, the LET statement, and the PRINT statement.

F.1. The NEW Command

As we saw in our previous programming examples, the NEW command deletes all program lines currently in memory. We will use the NEW command (in PROgram mode) before each sample program to insure that the only instructions in SHARP's memory are the instructions of our current program. Although it is possible (and often desirable) to have several programs in memory simultaneously, we will postpone the use of this feature to avoid confusion.

In PROgram mode, issue the NEW command. What effect do the Up and Down Arrow keys have now?

F.2. The LET Statement

Don't be alarmed if you don't recognize the LET statement as an "old friend"; I sneaked this one in on you. Actually, the LET statement is nothing but the Assignment statement in disguise. (If you don't recognize the Assignment statement either, please re-read the section on variables, in Chapter 2, immediately!).

In the early days of BASIC every statement began with a "keyword" (like PRINT, INPUT, etc.) which indicated what the instruction did. LET was the keyword which identified the fact that a value was to be stored into a variable. Today, it is generally agreed that the word LET is not really needed. As a result, the keyword LET is optional in PC-1500 BASIC. Thus, a statement which stores the number 7 into a variable S can be written in either of the following ways:

S = 7 \longrightarrow or \longrightarrow

\longleftarrow LET S = 7

The one exception, the place where LET must be used, is an assignment which occurs as part of an IF statement. Although we have not yet discussed the IF statement, this should be possible to explain (cross your fingers).

Using the IF statement you can write an instruction like the following:

IF expression THEN statement

If the statement following the THEN is an Assignment statement, the keyword LET must be used. This would result in a statement with the form:

IF expression THEN LET variable-name = expression

NOTE: Omitting the LET in this case would produce an ERROR 19 (or other error message).

F.3. The PRINT Statement

For most of the programs you write, the instructions will follow a basic pattern (no pun intended). There will be instructions to read in raw data, instructions to process the data, and instructions to print or display the results. This pattern has been referred to as the Input, Processing, Output cycle.

The PRINT statement is the main statement used to produce output. It is not surprising, therefore, that the PRINT statement has several different variations. The general format of the PRINT statement is the word PRINT followed by an item or a list of items to be printed. These include character strings, expressions, or names of variables whose values will be printed. Each item in a list is separated by a comma or a semicolon.

Enter the following program to demonstrate the printing of single items. (Remember to issue the NEW command before you begin.)

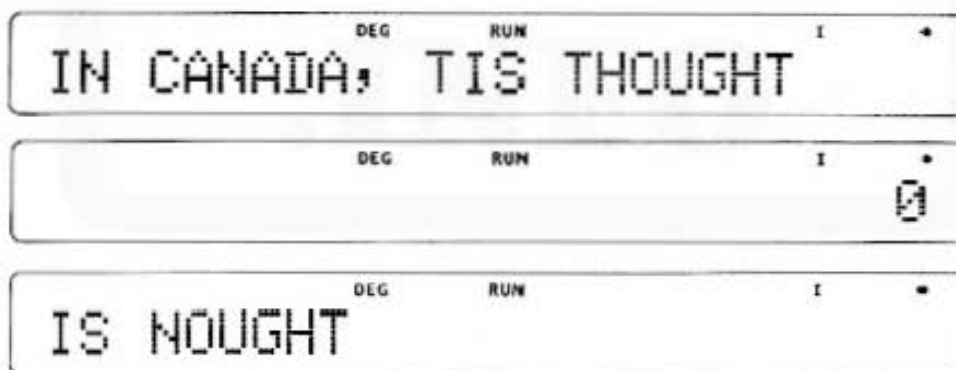
Program Listing:

```
10 Z$ = "IS NOUGHT"  
20 ZZ = 0  
30 PRINT "IN CANADA, TIS THOUGHT"  
40 PRINT ZZ  
50 PRINT Z$
```

Keystrokes:

```
1 1 0 Z SHIFT $ = SHIFT " I $ SPACE  
N O U G H T SHIFT " ENTER  
2 2 0 Z Z = 0 ENTER  
3 3 0 P R I N T SHIFT " I N SPACE  
C A N A D A SHIFT +  
SPACE T I S SPACE T H O U G H T  
SHIFT " ENTER  
4 4 0 P R I N T Z Z ENTER  
5 5 0 P R I N T Z SHIFT $ ENTER
```

When run, this program should produce three lines of output as follows (press **ENTER** after reading each line):



The first of our PRINT statements (line 30) displays a character string or "literal". Notice that the double quote marks are NOT printed as part of the output. They are necessary to "delimit", or mark, the beginning or end of the sequence of characters which you wish to print. This sequence can include any character except the double quote character itself.

The items in the second and third PRINT statements (lines 40 and 50) should be recognizable to all readers as variables. When a variable name is used within a print list, the value of the variable is printed. In this case, we knew what the values of ZZ and Z\$ would be since they were specified by lines 10 and 20. Printing an "empty" variable will result in a zero or a blank depending on whether the variable is numeric or character.

Clever readers will have observed that character strings and the values of character variables are printed beginning on the left side of the display. This is known as "left justified". By contrast, numbers and the values of numeric variables are "right justified".

It is also possible to print the result of an expression which is contained in the PRINT statement. The following one-line program illustrates this:

Program Listing:

```
10 PRINT (1982 - 1956) * 365.25
```

Keystrokes:

1 0 P R I N T (1 9 8 2 - 1 9 5 6)
* 3 6 5 . 2 5 ENTER

As you might expect, the result, a number, is right justified.

For the sake of efficiency, it is best to avoid computing expressions within PRINT statements unless that statement (and its associated expression) will be executed only one time in a program.

Because most programs compute several results at once, the printing of several items simultaneously is a common practice.

Perhaps the simplest of the multiple-item PRINT statements divides the display window into two sections. Each section contains one of two items specified in the print list. The items are separated in the list by a comma. Explore this format using the following program:

Program Listing:

```
10 A = 2 * PI
20 PRINT "2 TIMES PI =", A
```

Keystrokes:

1 0 A = 2 * P I ENTER
2 0 P R I N T SHIFT " 2 SPACE
T I M E S SPACE P I =
SHIFT " SHIFT , A ENTER

Run the program. As in the single-item PRINT statements, numbers are right justified and characters are left justified. In this case, the justification (alignment) occurs within each of the two sections of the display.

Using your editing skills, alter line 20 to read:

```
20 PRINT A, "= 2 TIMES PI"
```

Although there are several ways to accomplish this editing, one set of keystrokes could be:

MODE ↓ ↑ ↓ ▶ ▶ SHIFT INS SHIFT INS A
SHIFT , ▶ SHIFT INS =
▶ ▶ ▶ ▶ ▶ ▶ ▶ ▶ ▶ ▶
SHIFT DEL ▶ SHIFT DEL SHIFT DEL ENTER

The output from this modified version should help you to identify the two sections of the display.

More than two items may be displayed on the same line through a variation of the PRINT statement which utilizes the semi colon. Build and check the following program:

Program Listing:

```
10 BS = " BE "  
20 T = 2  
30 PRINT T;BS; "OR NOT"; 12/3 - 2; BS
```

Keystrokes:

```
1 0 B SHIFT $ = SHIFT " SPACE B E SPACE  
  SHIFT * ENTER  
2 0 T = 2 ENTER  
3 0 P R I N T T SHIFT : B  
  SHIFT $ SHIFT :  
  SHIFT " O R SPACE N O T SHIFT *  
  SHIFT : 1 2 / 3 - 2 SHIFT :  
  B SHIFT $ ENTER
```

Running this electronic composition should produce the following output:

```
                DEG      RUN                I      *  
2 BE OR NOT 2 BE
```

Shakespeare it ain't, but it does illustrate the action of the PRINT statement when the printable items are separated by semicolons. In this format, the items are adjacently displayed with a minimum separation between them. This capability is very handy for creating "natural" looking output (i.e. output which flows together).

NOTE: If the length of the information displayed exceeds the space available on the display (25 characters), the items at the end of the print list will not be seen.

Another use of the hard-working semicolon is at the very end of a PRINT statement. In this capacity, the semicolon indicates that the output currently on the display should be saved and that any new output (from the next PRINT statement) should join the old output on the same line. This process is easier to program than to describe, so let's experiment with the following program (don't forget the NEW command):

Program Listing:

```
100 PRINT "HUMPTY ";  
110 PRINT "DUMPTY"
```

Keystrokes:

```

1  Ø  Ø  P  R  I  N  T  SHIFT "  H  U  M  P  T  Y
    SPACE  SHIFT "  SHIFT :  ENTER
1  1  Ø  P  R  I  N  T  SHIFT "  D  U  M  P  T  Y
    SHIFT "  ENTER
    
```

Now run it. As usual, you must press **ENTER** after the first PRINT statement has displayed "HUMPTY ". Because of the semicolon "HUMPTY " is retained and "DUMPTY" shares the display with it. Contrast this with the execution of the following program which does not utilize the semicolon and the differences will become clear:

Program Listing:

```

10 PRINT "HUMPTY "
20 PRINT "DUMPTY"
    
```

Keystrokes:

```

1  Ø  P  R  I  N  T  SHIFT "  H  U  M  P  T  Y
    SPACE  SHIFT "  ENTER
2  Ø  P  R  I  N  T  SHIFT "  D  U  M  P  T  Y
    SHIFT "  ENTER
    
```

The next sample program illustrates that the semicolon may join as many lines of output as will fit on the display. Printing too much information on the display is a mistake which SHARP will not signal. It is up to you, as a programmer, to insure that this will not happen. Try the following educational program:

Program Listing:

```

20 PRINT "DO ";
40 PRINT "RE ";
60 PRINT "MI ";
80 PRINT "FA ";
100 PRINT "SOL ";
120 PRINT "LA ";
140 PRINT "TI ";
160 PRINT "DO "
    
```

Keystrokes:

```

2 0 P R I N T SHIFT " D O SPACE
    SHIFT " SHIFT : ENTER
4 0 P R I N T SHIFT " R E SPACE
    SHIFT " SHIFT : ENTER
6 0 P R I N T SHIFT " M I SPACE
    SHIFT " SHIFT : ENTER
8 0 P R I N T SHIFT " F A SPACE
    SHIFT " SHIFT : ENTER
1 0 0 P R I N T SHIFT " S O L SPACE
    SHIFT " SHIFT : ENTER
1 2 0 P R I N T SHIFT " L A SPACE
    SHIFT " SHIFT : ENTER
1 4 0 P R I N T SHIFT " T I SPACE
    SHIFT " SHIFT : ENTER
1 6 0 P R I N T SHIFT " D O SHIFT "
    ENTER

```

Now run the program, pressing `ENTER` repeatedly while joyously singing each note in the scale (okay, okay . . . you don't have to sing, but you still have to press `ENTER`).

G. The PAUSE Statement

The PAUSE statement is a semi-automatic form of the PRINT statement. It displays the various items in its associated list for a fixed and brief time period. The user is thus freed from the burden of prodding SHARP to continue by pressing ENTER. Think of the PAUSE as a PRINT statement followed by a countdown. When the countdown is over the program continues.

The formats of the PAUSE statement are identical to those of the PRINT statement. All of the various techniques which we have discussed for the PRINT statement are applicable to the PAUSE statement, although the resulting output will, of course, vary slightly. To illustrate one use of the PAUSE statement, let's re-write our musical scale program:

Program Listing:

```

10 PAUSE "DO ";
20 PAUSE "RE ";
30 PAUSE "MI ";
40 PAUSE "FA ";
50 PAUSE "SOL ";
60 PAUSE "LA ";
70 PAUSE "TI ";
80 PAUSE "DO"

```

Keystrokes:

```

1  Ø P A U S E SHIFT * D O SPACE
   SHIFT * SHIFT : ENTER
2  Ø P A U S E SHIFT * R E SPACE
   SHIFT * SHIFT : ENTER
3  Ø P A U S E SHIFT * M I SPACE
   SHIFT * SHIFT : ENTER
4  Ø P A U S E SHIFT * F A SPACE
   SHIFT * SHIFT : ENTER
5  Ø P A U S E SHIFT * S O L SPACE
   SHIFT * SHIFT : ENTER
6  Ø P A U S E SHIFT * L A SPACE
   SHIFT * SHIFT : ENTER
7  Ø P A U S E SHIFT * T I SPACE
   SHIFT * SHIFT : ENTER
8  Ø P A U S E SHIFT * D O
   SHIFT * ENTER

```

Notice what happens after the last "note" is printed; the program ends and the display returns to the prompt. This happens because there are no other statements following the last PAUSE. To "Freeze" the display after the last note is printed, we can change line 80 to:

```
80 PRINT "DO"
```

Keystrokes:

```

8 Ø P R I N T SHIFT * D O
   SHIFT * ENTER

```

After all, there's no reason why we can't mix PRINT and PAUSE statements throughout our program. Try this for yourself.

H. INPUT

By utilizing the various forms of the PRINT statement, alone or in combination, information can be nicely presented to the computer user. Most of the items which will be printed, however, are the result of processing some initial data. This initial data is given to the program by that same computer user. The instruction which controls this process is the INPUT instruction.

In its simplest form; INPUT variable-name, the INPUT instruction merely prints a ? (question mark) and then waits for the user to type in the required information. What is "required" depends on the variable which appears as part of the INPUT instruction. For example, if the variable is numeric, the user should enter a number, which will be stored in the variable.

Do you see a problem with this form of the INPUT statement? Good. As some of you may realize, unless the user of the program is also the programmer (and probably not even then), he will not know what type of data to enter when he sees the question mark. It is up to the programmer to keep the user informed at all times.

As an example of a program which fails to do this, enter the following:

Program Listing:

```
10 A = 0
20 INPUT A
30 PRINT A * PI
```

Keystrokes:

```
1 0 A = 0 ENTER
2 0 I N P U T A ENTER
3 0 P R I N T A * P I ENTER
```

Now imagine the user as he runs the program . . . The first thing that appears on the display is a question mark. The knowledgeable user will realize that some data is required but he will have no idea what that data should be. Suppose he takes a wild guess, and luckily, enters a number. Suddenly, a long and complicated number appears. What does it mean? He presses ENTER to continue, but the program ends. From his viewpoint, the whole experience has been a waste of time. Why? Because of poor programming.

One solution to this problem is to use PRINT or PAUSE statements to help the user along. With this idea we can re-write our program as follows:

Program Listing:

```
10 A = 0
20 PAUSE "ENTER ANY NUMBER"
30 INPUT A
40 AP = A * PI
50 PRINT A; " TIMES PI = "; AP
```

Keystrokes:

```
1 0 A = 0 ENTER
2 0 P A U S E SHIFT " E N T E R SPACE
  A N Y SPACE N U M B E R SHIFT " ENTER
3 0 I N P U T A ENTER
4 0 A P = A * P I ENTER
5 0 P R I N T A SHIFT ; SHIFT " SPACE
  T I M E S SPACE P I SPACE =
  SPACE SHIFT * SHIFT ; A P ENTER
```

This version is much more useful because it "prompts" (or provokes) the user for input and because it identifies the output.

The operation of prompting (printing a request for input) is so common that more advanced forms of the INPUT statement have been created which incorporate it. The first of these is written with a semicolon:

INPUT "characters" ; variable-name

Astute readers will recognize our old friend the character string. This string will be printed on the display as a prompt. The cursor, replacing the question mark, will follow on the same line, and SHARP will wait for the user's input. This form of the INPUT statement allows us to re-write our previous example as:

Program Listing:

```
10 A = 0
20 INPUT "ENTER ANY NUMBER "; A
30 AP = A * PI
40 PRINT A; " TIMES PI = "; AP
```

Keystrokes:

```
1 0 A = 0 ENTER
2 0 I N P U T SHIFT " E N T E R SPACE
  A N Y SPACE N U M B E R SPACE
  SHIFT " SHIFT ; A ENTER
3 0 A P = A * P I ENTER
4 0 P R I N T A SHIFT ; SHIFT " SPACE
  T I M E S SPACE P I SPACE =
  SPACE SHIFT " SHIFT ; A P ENTER
```

As you run this example, notice how the use of the INPUT statement differs from the PAUSE statement.

The second prompting input form is almost identical to the first except that it uses a comma instead of a semicolon:

INPUT "characters", variable-name

When this form of the statement is executed, the associated character string is displayed and the computer again waits for input. This time, however, as the user begins to type, the prompt message is cleared and the user's data appears in its place. This allows data entry after a long prompt without running off the end of the display.

Change the semicolon to a comma in line 20 of our program and re-run the program.

Of course, the entry of data is not limited to numbers, as our examples so far have implied. To enter characters we merely specify a character variable in the INPUT statement as in this brilliant, deductive program:

Program Listing:

```

10 INPUT "ENTER YOUR LAST NAME "; LS
20 INPUT "ENTER YOUR FIRST NAME "; FS
30 IS = LEFTS (FS, 1) + " ." + LEFTS (LS, 1) + " ."
40 PAUSE "GEE,"
50 PRINT "YOUR INITIALS ARE "; IS

```

```

1 0 I N P U T SHIFT " E N T E R SPACE
   Y O U R SPACE L A S T SPACE
   N A M E SPACE SHIFT " SHIFT :
   L SHIFT $ ENTER

```

```

2 0 I N P U T SHIFT " E N T E R SPACE
   Y O U R SPACE F I R S T SPACE
   N A M E SPACE SHIFT "
   SHIFT : F SHIFT $ ENTER

```

```

3 0 I SHIFT S = L E F T SHIFT $ I F
   SHIFT $ SHIFT , 1
   ) + SHIFT " . SHIFT " +
   L E F T SHIFT $ I L SHIFT $
   SHIFT , 1 ) + SHIFT " .
   SHIFT " ENTER

```

```

4 0 P A U S E SHIFT " G E E SHIFT ,
   SHIFT " ENTER

```

```

5 0 P R I N T SHIFT " Y O U R SPACE
   I N I T I A L S SPACE A R E SPACE
   SHIFT " SHIFT : I SHIFT $ ENTER

```

NOTE: Don't worry about line 30; it uses advanced techniques which you will learn later.

In addition to accepting a single item of data, the INPUT instruction may be used to gather and store several such values. To program such a process, one merely lists the variables which will receive data within the INPUT instruction. Each variable in the list is separated by a comma.

As an example of multiple-item input, build this statistical program:

Program Listing:

```

10 W = 0 : X = 0 : Y = 0 : Z = 0
20 INPUT "ENTER AGES OF 4 PEOPLE", W, X, Y, Z
30 S = W + X + Y + Z : A = S/4
40 PAUSE "TOTAL YEARS = "; S
50 PRINT "AVERAGE AGE IS "; A

```

Keystrokes:

```

1  Ø W = Ø SHIFT : X = Ø SHIFT : Y = Ø
   SHIFT : Z = Ø ENTER
2  Ø I N P U T SHIFT " E N T E R SPACE
   A G E S SPACE O F SPACE 4 SPACE
   P E O P L E SHIFT " SHIFT ↵
   W SHIFT ↵ X SHIFT ↵ Y SHIFT ↵
   Z ENTER
3  Ø S = W + X + Y + Z SHIFT : A = S / 4
   ENTER
4  Ø P A U S E SHIFT " T O T A L SPACE
   Y E A R S SPACE = SPACE SHIFT "
   SHIFT : S ENTER
5  Ø P R I N T SHIFT " A V E R A G E SPACE
   A G E SPACE I S SPACE SHIFT "
   SHIFT : A ENTER

```

When run, this somewhat indiscrete program will prompt you to enter 4 numbers. The first number you type will replace the prompt because we used a comma directly after the character string in the INPUT statement. When the first number is completely typed you must press `ENTER`. A question mark will precede every successive number each of which must also be followed by pressing `ENTER`.

Had we used a semicolon in the INPUT statement, the first number entered would not have replaced the prompt but would have shared the line with it. All successive inputs would have been the same as with the comma. Try this for yourself by altering line 20 to read:

```
20 INPUT "ENTER AGES OF 4 PEOPLE"; W, X, Y, Z
```

Keystrokes:

```

2  Ø I N P U T SHIFT " E N T E R SPACE
   A G E S SPACE O F SPACE 4 SPACE
   P E O P L E SHIFT " SHIFT :
   W SHIFT ↵ X SHIFT ↵ Y SHIFT ↵
   Z ENTER

```

I. SHORTCUTS AND HELPFUL HINTS

Since you have been so diligent and patient, I have provided this section as a small reward. I know it's not as good as money but the information contained here may save you some effort.

I.1. Abbreviations

Those of you whose golden fingers do not fly swiftly over the computer's keys will have noticed that our sample programs have been growing steadily larger. In a burst of merciful forethought, SHARP's clever designers have anticipated your difficulties. They have enabled SHARP to recognize abbreviations for frequently used statements and commands.

The general form of an abbreviation is one or more designated letters followed by a period. The period is crucial to avoid confusion with variable names. As an example, enter the following program whose statements are abbreviated:

Program Listing:

```
15 PA. "HELLO, HUMAN."
25 I. "WHAT IS YOUR NAME?"; NS
35 P. "GLAD TO MEET YOU, "; NS
```

Keystrokes:

```
1 5 P A . SHIFT " H E L L O SHIFT ,
   H U M A N . SHIFT " ENTER
2 5 I . SHIFT " W H A T SPACE I S SPACE
   Y O U R SPACE N A M E SHIFT ? SHIFT "
   SHIFT ; N SHIFT $ ENTER
3 5 P . SHIFT " G L A D SPACE T O SPACE
   M E E T SPACE Y O U SHIFT , SPACE
   SHIFT " SHIFT ;
   N SHIFT $ ENTER
```

As you complete the entry of each line (by pressing the ENTER key), notice that SHARP expands any abbreviations on that line. The resulting clarity is immensely valuable to you later, as you check the program for errors. An additional hidden benefit is that the space used to store the abbreviations for the statements is no more (or less) than the space used to store the full statement. Thus, the abbreviation facility is strictly a convenience for you, the programmer.

To enhance readability we will not use these abbreviations in our sample program listings (although you may, of course, when entering the programs). The following are the allowable abbreviations for commands and statements already introduced:

PRINT	P. PR. PRI.
PAUSE	PA. PAU.
INPUT	I. IN. INP.
RUN	R.

A complete list of abbreviations is included in the back of this manual as an Appendix.

1.2. Multiple Statements and the Colon

As mentioned in the section on line numbers, several statements may share the same line. SHARP will execute the statements in sequence from left to right. In order for SHARP to be able to distinguish the end of one statement from the beginning of the next, some "signalling" character must be used. This character is the colon (:). By analogy, the function of the colon is similar to the function of the periods which separate the sentences on this page; it tells one when to stop reading.

The use of the colon has already been illustrated in several of the last few sample programs. Its general form is as follows:

line-number statement 1 : statement 2 : statement 3 (etc)

The question of when to use the colon is a matter of programming style. Indiscriminate use of the colon to write very compact programs produces programs which are very difficult to read, correct, or expand. Since the prime advantages of the PC-1500 are that it is personal and interactive, it is highly desirable that you are able to modify and extend programs to suit your own needs. We therefore recommend that you exercise restraint in your use of the colon.

If you do use the colon, it is very advantageous to group only those statements which are conceptually related. That is, place only those few statements which accomplish a single task, out of the many possible tasks that comprise the program, on one line.

For example, consider the following program statements which read in three numbers, find their average, compute the difference of each number from the average, and print the sum of these differences:

Program Listing:

```
10 N1 = 0 : N2 = 0 : N3 = 0
20 INPUT "ENTER 3 NUMBERS", N1, N2, N3
30 A = (N1 + N2 + N3) / 3
40 D1 = N1 - A : D2 = N2 - A : D3 = N3 - A
50 SD = D1 + D2 + D3
60 PRINT "SUM OF DIFFERENCES = "; SD
```

Keystrokes:

```
1 [ ] [ ] N 1 = [ ] [ ] SHIFT : N 2 = [ ] [ ] SHIFT : N 3
  = [ ] [ ] ENTER
2 [ ] [ ] I N P U T SHIFT " E N T E R [ ] [ ] SPACE 3
  SPACE N U M B E R S SHIFT " SHIFT ,
  N 1 SHIFT , N 2 SHIFT , N 3 ENTER
3 [ ] [ ] A = [ ] [ ] N 1 + N 2 + N 3 [ ] [ ] / [ ] [ ] 3 ENTER
4 [ ] [ ] D 1 = N 1 - A SHIFT : D 2 = N 2 - A
  SHIFT : D 3 = N 3 - A ENTER
5 [ ] [ ] S D = D 1 + D 2 + D 3 ENTER
6 [ ] [ ] P R I N T SHIFT " S U M SPACE
  O F SPACE D I F F E R E N C E S
  = SPACE
  SHIFT " SHIFT : S D ENTER
```

Notice how each line in the program accomplishes a single complete and necessary step. Line 10 erases any previous values which might be contained in the variables N1, N2, and N3 (this is a precaution in case the user fails to enter all three numbers). Line 20 collects the data from the user. Line 30 averages the numbers. Line 40 computes the three differences. Line 50 sums the differences and line 60 prints the result.

It is no accident that the instructions on each line correspond to the English language description of the program. Instead, this is one of the principles of good programming which you should try to follow.

Unlike the use of abbreviations, the use of the colon does affect the amount of memory used to store the program. This is the chief justification for using the colon to place several statements on one line. Each line number reserves several program "steps" (a unit of storage) and therefore the fewer the line numbers in a program, the smaller memory size of the program.

The final statement about the use of the colon is that each programmer must balance each program's readability and changeability against the storage needs of his application.

J. Error Correction in the Program Mode

Although abbreviations and colons help us to easily enter programs, they cannot prevent the best of us from making mistakes. Even professional programmers fail to catch errors when reviewing their own programs. What this means is that sooner or later you will encounter an error while running your program (if you haven't already). Most of these errors are easily corrected if you simply accept them as a puzzle to be solved and carefully track down the problem. Several features of the PC-1500 will assist you in this.

Upon discovering an incorrect statement SHARP will halt and indicate the problem with a terse message such as:

```

                                RUN
ERROR 1 IN 20
    
```

Let's create a program with a deliberate error for purposes of illustration. Enter:

```

Program Listing:
25 PAUSE "HUMPTY DUMPTY"
50 PRINT "WAS AN EGGHEAD"
    
```

Keystrokes:

```

[2][5][P][A][U][S][E][SHIFT][*][H][U][M][P][T][Y][SPACE]
[D][U][M][P][T][Y][SHIFT][*][ENTER]
[5][O][P][R][I][M][T][SHIFT]["][W][A][S][SPACE][A][N]
[SPACE][E][G][G][H][E][A][D][SHIFT][*][ENTER]
    
```

Now run the program. When the error message appears depress the **[↑]** (Up Arrow) key. As long as you hold this key, the display will show the line on which SHARP became confused. The flashing grid may provide a hint as to the nature of the problem.

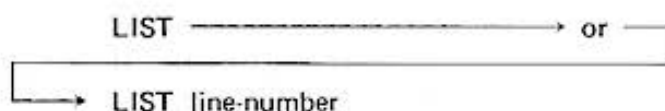
To correct the bad statement, press **[CL]** to quit the program and switch to the PROgram mode. Press the Up Arrow key, but do not hold it, and the display will again show the erroneous line:

50 PRINT "WAS AN EGGHEAD"

You may now proceed to edit the line using the familiar INSert, DElete, and Arrow keys. When you have finished editing, you must press **ENTER** to instruct SHARP to store the corrected line.

K. The LIST Command

Another way to display a particular line of a program is to use the LIST command whose form is:



If no line-number is given, the first program line will be displayed.

If a line-number is specified, the program line with that number is displayed. If no program line has the given number, the next program line whose number is larger is displayed. For example, assuming the following program:

```
15 PRINT "MOTHER GOOSE"  
30 PRINT "WAS A";  
45 PRINT "QUACK"
```

the command LIST 40 will display line 45. The command LIST will display line 15, and the command LIST 30 will display line 30 (what else?).

NOTE: If you specify a line-number which is larger than any existing program line-number, an ERROR 11 will occur.

L. The More, the Merrier

As we have mentioned previously, it is possible to maintain more than one program in memory at the same time. The trick to doing this is to give each program its own range of line-numbers. For example, one program could have line-numbers of 10 to 200 while the lines of a second program would be numbered from 300 to 500. You must, of course, be careful not to accidentally intermix statements from the various programs (by incorrect numbering) or unpredictable results could follow.

L.1. The END Statement

Another problem that occurs while storing several programs simultaneously is that each program is just a group of numbered statements and statements are executed in ascending order. Therefore, how will SHARP know when it has finished executing statements from a given program? The answer is that it won't unless you tell it. The statement used to signal the computer that it has reached the end of a program is the END statement.

Until now we have not used, nor needed, the END statement. SHARP has simply executed all of our program lines in ascending order until it ran out of statements. When it ran out, the computer decided that the program was finished and returned to wait for our next command. Now, however, we must tell SHARP to stop executing instructions before it continues on to the next program.

To illustrate the use of multiple programs enter the following lines:


```
10 PAUSE "HUMPTY DUMPTY"  
20 PAUSE "HAD A GREAT FALL"  
30 PRINT "BUT NOT A GOOD SPRING"  
40 END  
200 PAUSE "THE OLD MANS GLASSES"  
210 PRINT "WERE FILLED WITH SHERRY"  
220 END
```

L.2. RUN line-number

Well, now that you have two programs in memory, how are you going to run each of them separately? If you are adventurous enough to have tried the normal RUN command, you will have discovered that it is perfectly adequate to initiate the first program. (If you are not adventurous, try this now). To begin the second program we need to use a variation of the RUN command which tells SHARP on what line to start. This is the RUN line-number command. To start the second program type:

```
R U N 2 0 0
```

and voila, it is done!

Like most helpful commands, the RUN line-number command can cause problems. Because it instructs SHARP where to begin following instructions, and because SHARP is such an obliging servant, programs can be started in the middle. This, as you might guess, should not be done by choice. If it is done, it will produce some, ah . . . unusual results. Try it on our first program by issuing the command:

```
RUN 30
```

This is a mild mistake compared to what could happen in a more complex program.

M. CONTROL STATEMENTS

Our programs so far have been a consecutive sequence of instructions each performed one time by the computer. Those with experience giving instructions may realize that this is not always the best way to accomplish a task. Often you allow your listener to choose one of several plans. Sometimes you "condense" your instructions by including a command like, "Now repeat the last 3 steps until you are finished."

These capabilities have been incorporated into statements in BASIC called control statements. These determine when, if, and how often other statements will be executed. Control statements enable you to build very powerful and versatile programs. In the next few sections we will discuss the major control statements of BASIC; the IF . . . THEN, the GOTO, and the GOSUB.

N. IF and THEN

The potential for choice within a BASIC program is provided by the IF statement. A program equipped with an IF statement can evaluate your input and make decisions:

Program Listing:

```

10 PAUSE "ARE YOU ASLEEP?"
20 INPUT "TYPE YES OR NO "; SX$
30 IF SX$ = "YES" THEN PRINT "OH, SORRY TO
   DISTURB YOU"
40 END

```

Keystrokes:

```

1  O P A U S E SHIFT " A R E SPACE
   Y O U SPACE A S L E E P SHIFT ?
   SHIFT " ENTER
2  O I N P U T SHIFT " T Y P E SPACE
   Y E S SPACE O R SPACE N O SPACE
   SHIFT " SHIFT ; $ X SHIFT $ ENTER
3  O I F S X SHIFT $ = SHIFT " Y E S SHIFT "
   T H E N P R I N T SHIFT " O H
   SHIFT ↵ SPACE S O R R Y SPACE T O
   SPACE D I S T U R B SPACE Y O U
   SHIFT " ENTER
4  Q E N D ENTER

```

This soporific program will only produce output if you answer affirmatively. The general form of the IF statement, which line 30 illustrates, is:

IF condition THEN statement

During execution, the test embedded in the IF clause is performed. Whether the statement is performed, or not, depends on the result of the test. The test is usually an inequality and is called a "condition." Remember that inequalities are comparisons which are either True or False. If the inequality is True then the statement is executed. If the inequality is False then the statement is skipped.

In our sample program the test is whether the variable SX\$ is equal to (contains) the character string "YES". If it does, and only if it does, then the PRINT instruction is followed. If SX\$ is not equal to "YES", then the PRINT instruction is ignored. **In either case, no matter whether the PRINT statement is executed or ignored, SHARP will proceed, as normal, to the next line.** (In our sample program the next line is line 40).

Notice that we could have reversed our test by altering several lines in the following manner:

```

30 IF SX$ = "NO" THEN END
40 PRINT "OH, SORRY TO DISTURB YOU"

```

This program, however, is not quite the same as the original. It allows our apologetic computer to talk with anyone who mistypes or who doesn't answer with a "NO". In addition, this program really has two endpoints; the END statement in line 30 and the implied END beyond line 40.

Several endings are not the result of good programming practice. Our reversal of these statements does demonstrate though, that correct ordering of statements and proper testing are necessary for a program to operate correctly. In the next section, we will observe a third way to phrase our program which involves the GOTO statement and solves the problems of the second version.

Although the condition of an IF statement is usually an inequality, it is not necessarily so. In PC-1500 BASIC any expression which evaluates to a positive, non-zero, number is considered to be True. Any expression, such as 5-9, which evaluates to zero or a negative number is considered to be False. This explains why inequalities work as conditions: remember that an inequality which is True returns a 1 and inequalities that are False return a 0. This method of representing True and False is not standard on all computers and can produce obscure programs. Use this facility judiciously.

One more reminder is in order. If the statement which follows the THEN is an assignment statement, the LET keyword **MUST** be used. Failure to do so will cause an ERROR condition to occur. This was discussed in the section on the LET statement.

O. GOTO

You may have noticed, in the last section, that our options were limited after the test was made in the IF statement. We were only allowed to execute one statement if the condition was True. For convenience, we would like to execute several statements. The GOTO statement allows us to do this.

The GOTO statement modifies the "flow" of statement execution. It tells SHARP to "go to" a line other than the next one and begin executing statements sequentially from there. The effect of this "jump" is that some statements may be skipped entirely. For example review the following program:

Program Listing:

```
10 PAUSE "ESCHEW ";
20 GOTO 50
30 PRINT X * 3 / 4 + 2
40 PRINT "A BOOK WHICH EMPLOYS";
50 PRINT "OBFUSCATION!"
60 END
```

Keystrokes:

```
1 [ ] [ ] P A U S E [SHIFT] ["] E S C H E W [ ]
   [SPACE] [SHIFT] ["] [SHIFT] [ ; ] [ENTER]
2 [ ] [ ] G O T O 5 [ ] [ ] [ENTER]
3 [ ] [ ] P R I N T X * 3 / 4 + 2 [ENTER]
4 [ ] [ ] P R I N T [SHIFT] ["] A [SPACE] B O O K [ ]
   [SPACE] W H I C H [SPACE] E M P L O Y [ ]
   [S] [SHIFT] ["] [SHIFT] [ ; ] [ENTER]
5 [ ] [ ] P R I N T [SHIFT] [*] O B F U S C [ ]
   [A] [T] [I] [O] [N] [SHIFT] [!] [SHIFT] ["] [ENTER]
6 [ ] [ ] E N D [ENTER]
```

Normally, of course, this program would execute in ascending line-number sequence. The effect of the GOTO in line 20, however, is to cause SHARP to proceed immediately to line 50 and to begin in ascending sequence from there. Lines 30 and 40 are never executed.

The general form of the GOTO statement is:

GOTO expression

where:

expression evaluates to a number which is a valid program line-number (i.e. 1 through 65279).

NOTE: Specifying a line-number which does not exist will result in an ERROR 11.

If we wish to have several instructions executed as a result of a certain choice we use GOTO statements in conjunction with the IF:

```
10 IF test THEN GOTO 100
20 { statements here are }
  { performed only if the }
  { test is False. }
90 GOTO 200
100 { statements here are }
   { performed only if the }
   { test is True. }
200 { statements here are }
   { always performed. }
999 END
```

The logic of this program is applicable to many situations. Any number of statements can be inserted in each of the sections which are formed by the GOTO statements.

As an example of this structure in practice, the following segment from a checkbook program determines whether a given input amount is a deposit (positive number) or a withdrawal (negative number). An initial balance is entered by the user:

Program Listing:

```
10 INPUT "INITIAL BALANCE?", B
20 INPUT "TRANSACTION AMOUNT?", TA
25 IF TA = 0 THEN GOTO 80
30 B = B + A
40 IF TA < 0 THEN GOTO 70
50 PRINT "DEPOSIT OF $"; TA; " POSTED"
60 GOTO 80
70 PRINT TA; " DOLLARS WITHDRAWN"
80 PRINT "FINAL BALANCE = "; B
90 END
```

Keystrokes:

```

1  Ø I N P U T SHIFT " I N I T
    I A L SPACE B A L A N C E
    SHIFT ? SHIFT " SHIFT , B ENTER
2  Ø I N P U T SHIFT " T R A N S
    A C T I O N SPACE A M O U N T SHIFT ?
    SHIFT " SHIFT , T A ENTER
2  5 I F T A = O T H E N G O T O
    B O ENTER
3  Ø B = B + A ENTER
4  Ø I F T A SHIFT < Ø T H E N
    G O T O 7 Ø ENTER
5  Ø P R I N T SHIFT " D E P O S
    I T SPACE O F SPACE SHIFT $
    SHIFT " SHIFT ; T A SHIFT ;
    SHIFT " SPACE P O S T E D SHIFT "
    ENTER
6  Ø G O T O B Ø ENTER
7  Ø P R I N T T A SHIFT ; SHIFT "
    SPACE D O L L A R S SPACE W I T
    H D R A W N SHIFT " ENTER
8  Ø P R I N T SHIFT " F I N A L
    SPACE B A L A N C E = SPACE
    SHIFT " SHIFT ; B ENTER
9  Ø E N D ENTER

```

Observe that this program ends only if the transaction amount is zero. The second IF statement is the one which illustrates the structure we mentioned before. Notice that in addition to the two separate actions associated with the IF statement (lines 50 and 70), there are some statements (lines 80 and 90) executed regardless of the result of the IF testing.

We can now write a third version of the program from the previous section:

Program Listing:

```

10 PAUSE "ARE YOU ASLEEP?"
20 INPUT "TYPE YES OR NO "; SX$
30 IF SX$ <> "YES" THEN GOTO 99
40 PRINT "OH, SORRY TO DISTURB YOU"
99 END

```


Keystrokes:

```

1  Ø P A U S E SHIFT " A R E SPACE
   Y O U SPACE A S L E E P SHIFT ?
   SHIFT " ENTER
2  Ø I N P U T SHIFT " Y Y P E SPACE
   Y E S SPACE O R SPACE N O SPACE
   SHIFT " SHIFT : S X SHIFT $ ENTER
3  Ø I F S X SHIFT $ SHIFT < SHIFT >
   SHIFT " Y E S SHIFT " T H E N
   G O T O 9 9 ENTER
4  Ø P R I N T SHIFT " O H SHIFT ↵
   SPACE S O R R Y SPACE T O SPACE
   D I S T U R B SPACE Y O U SHIFT "
   ENTER
9 9 E N D ENTER
    
```

This version reverses the conditional test by specifying some action to be performed if the user's input is NOT equal to "YES". Thus, it eliminates the problems of the second version, and could be expanded into the larger form of the IF statement described above. Re-arranging statements in this fashion is often a useful maneuver when programming. Take some time to experiment on your own and you will be rewarded with better programs.

Another very common use of the GOTO statement is to cause repeated execution of a sequence of statements. This process is known as "looping". A simple example of looping is illustrated by this program:

Program Listing:

```

10 WAIT 30
20 PRINT "CHUG CHUG CHUG CHUG"
30 PRINT "    CHUG CHUG CHUG CHUG"
40 GOTO 20
    
```

Keystrokes:

```

1  Ø W A I T 3 0 ENTER
2  Ø P R I N T SHIFT " C H U G SPACE
   C H U G SPACE C H U G SPACE C H
   U G SHIFT " ENTER
3  Ø P R I N T SHIFT " SPACE SPACE SPACE
   SPACE SPACE C H U G SPACE C H U G
   SPACE C H U G SPACE C H U G SHIFT "
   ENTER
4  Ø G O T O 2 0 ENTER
    
```

Unfortunately, this program will just keep "chugging" along forever. (You may stop the program by using the BREAK key). We should provide a way for such programs to end. We can do this by using a "counter" along with an IF statement. A counter is a variable in which we keep a record of how many times we have done something (i.e. we count with it). With this technique the IF statement will test whether we have performed our PRINT statements a predetermined number of times. Of course, it is entirely up to us to determine the number of repetitions. Let's pick 10 times for each PRINT statement (after 10 times I get bored).

Using our counter and IF statement we can write:

Program Listing:

```
10 WAIT 30
15 C = 1
20 PRINT "CHUG CHUG CHUG CHUG"
30 PRINT "      CHUG CHUG CHUG CHUG"
40 C = C + 1
50 IF C <= 10 THEN 20
60 END
```

Keystrokes:

```
1 0 W A I T 3 0 ENTER
1 5 C = 1 ENTER
2 0 P R I N T SHIFT " C H U G SPACE
  C H U G SPACE C H U G SPACE C H
  U G SHIFT " ENTER
3 0 P R I N T SHIFT " SPACE SPACE SPACE
  SPACE SPACE C H U G SPACE C H U G
  SPACE C H U G SPACE C H U G SHIFT "
  ENTER
4 0 C = C + 1 ENTER
5 0 I F C SHIFT < = 1 0 T H E N
  2 0 ENTER
6 0 E N D ENTER
```

Follow the operation of the counter as each loop is executed. Notice that in line 15 we must assign the counter an initial value. On line 40 we increment this value to reflect one more execution of statements 20 and 30.

There are many other ways to use counters and loops within programs. Unfortunately, we do not have the space to describe them here. We suggest that you pursue your education with one of the books listed in Appendix F.

The GOTO instruction is a command as well as a statement. As a command its use is naturally different from its use as a program statement. Issued as a command, in the RUN mode, GOTO begins program execution in a manner similar to the RUN command. The difference lies in certain internal preparations which are made before instructions of the program are performed. (For a

comparison of the methods used to begin a program, see the Chapter entitled Beginning Program Execution). Unlike the RUN command the GOTO command will not clear values from any variables before it begins execution of the program.

To begin program execution with the GOTO command ENTER:

GOTO line-number

where:

line-number is the number of the first line of the program to be executed.

NOTE: Specifying a line-number which does not exist will result in an ERROR 11.

P. FOR and NEXT

As we saw in the previous section, the ability to repeat a series of instructions is very useful. In fact this feature is so often used that BASIC incorporates several statements to automate the process. These are the FOR statement and its partner, the NEXT statement. Together, the FOR and the NEXT statements enclose a series of instructions which are repeated a number of times. The FOR statement has an associated counter variable and a built-in test condition. It also allows the specification of the initial value and the increment value of the counter variable.

The form for all of this information is:

FOR counter-variable = initial-value TO final-value STEP increment-value

where:

counter-variable is the name of the variable used to hold the loop count.

initial-value is the value stored in the counter-variable before the first time through the loop.

The allowable range for this value is -32768 through 32767.

final-value is the number which is used in the test. If the counter-variable contains a value greater than final-value, the looping is ended. The legal range for this number is -32768 through 32767.

STEP increment-value is an optional clause. The increment-value indicates by how much to increase or decrease the counter-variable each time through the loop. This must be an integer in the range - 32768 through 32767. If the entire clause is omitted, then the increment-value is assumed to be one.

This is a lot to handle, so let's observe the behavior of some simple, sample programs. The first program is similar to the version of the CHUG CHUG program which used a counter. Instead of "chugging" we print the value of the counter-variable C:

```
15 FOR C = 1 TO 10
30 PAUSE C
50 NEXT C
```

(To make this program "CHUG" as before, simply insert statements 10, 20, and 30 from that program). Notice that this version is neater and more concise using fewer statements to accomplish the same counting and looping functions than the older version.

In case there is still some residual confusion about what the FOR and NEXT statements are doing, we present a comparison of a FOR . . . NEXT loop with the equivalent statements:

<pre> 10 FOR I = 1 TO 8 20 [some statements : to be repeated] 90 NEXT I 100 END </pre>	<pre> 10 I = 1 12 IF I > 8 THEN 100 20 [some statements to : be repeated] 90 I = I + 1 92 GOTO 12 100 END </pre>
---	--

Our second sample program illustrates how one may program a "general loop", whose repetitions are controlled by the value of a variable. We begin by asking the user how many numbers he/she plans to enter. We store this number in a variable N and we proceed to loop through the statements which take in a number for processing. The loop is executed N times, unless N is less than or equal to zero, in which case we END without further processing.

```

Program Listing:
10 N = 0 : V = 0 : T = 0 : A = 0
20 WAIT 0
30 INPUT "HOW MANY VALUES? "; N
40 IF N = 0 THEN GOTO 999
50 FOR I = 1 TO N
60 CLS : CURSOR 0
70 INPUT V
80 T = T + V
90 NEXT I
100 WAIT : CLS : CURSOR 0
110 A = T / N
120 PAUSE "TOTAL = "; T
130 PRINT "AVERAGE = "; A
999 END
    
```

Keystrokes:

- 1 [0] [N] [=] [0] [SHIFT] [:] [V] [=] [0] [SHIFT] [:] [T] [=] [0]
[SHIFT] [:] [A] [=] [0] [ENTER]
- 2 [0] [W] [A] [I] [T] [0] [ENTER]
- 3 [0] [I] [N] [P] [U] [T] [SHIFT] ["] [H] [O] [W] [SPACE] [M] [A] [N]
[Y] [SPACE] [V] [A] [L] [U] [E] [S] [SHIFT] [?] [SPACE]
[SHIFT] ["] [SHIFT] [:] [N] [ENTER]
- 4 [0] [I] [F] [N] [=] [0] [Y] [H] [E] [N] [G] [O] [T] [O]
[9] [9] [9] [ENTER]
- 5 [0] [F] [O] [R] [I] [=] [I] [T] [O] [N] [ENTER]
- 6 [0] [C] [L] [S] [SHIFT] [:] [C] [U] [R] [S] [O] [R] [0] [ENTER]
- 7 [0] [I] [N] [P] [U] [T] [V] [ENTER]

```

8 0 T = T + V ENTER
9 0 N E X T I ENTER
1 0 0 W A I T SHIFT : C L S SHIFT :
    C U R S O R 0 ENTER
1 1 0 A = T / N ENTER
1 2 0 P A U S E SHIFT " Y O T A L
    SPACE = SPACE SHIFT " SHIFT :
    T ENTER
1 3 0 P R I N T SHIFT " A V E R A G E
    SPACE = SPACE SHIFT " SHIFT :
    A ENTER
9 9 9 E N D ENTER
    
```

The FOR . . . NEXT loop need not always increment the counter-variable by 1 nor always initialize it to 1. Using the STEP clause, the programmer may specify the size of the increment (or decrement). Our next example program demonstrates this and brings back memories of high school cheerleading:

Program Listing:

```

10 WAIT 30
20 FOR HS = 2 TO 8 STEP 2
30 PRINT HS; " ! ";
40 NEXT HS
50 WAIT 60 : CLS : CURSOR 0
60 PRINT "WHO DO WE APPRECIATE?"
70 PRINT "SHARP PC-1500!"
80 END
    
```

Keystrokes:

```

1 0 W A I T 3 0 ENTER
2 0 F O R H S = 2 T O 8 S T E P
    2 ENTER
3 0 P R I N T H S SHIFT : SHIFT "
    SHIFT ! SPACE SHIFT " SHIFT :
    ENTER
4 0 N E X T H S ENTER
5 0 W A I T 6 0 SHIFT : C L S SHIFT :
    C U R S O R 0 ENTER
6 0 P R I N T SHIFT " W H O SPACE D O
    SPACE W E SPACE A P P R E C I
    A T E SHIFT ? SHIFT " ENTER
    
```



```

7 0 P R I N T SHIFT * S H A R P SPACE
P C - 1 5 0 0 SHIFT ! SHIFT " ENTER
8 0 E N D ENTER

```

In addition to counting up, the STEP clause also allows SHARP to count down. This is done by reversing the initial and final values and specifying a negative increment. The following program, (dedicated to consumers everywhere), illustrates this:

The Coverup Carpet Emporium is offering rugs to the LAQ INN at the amazing, low price of \$.99 per square foot. The rugs range in radius from 40 feet (for the hotel lobby) to 1 foot (for the bathrooms). Between each rug and the next smaller size is a gap of three feet in radius. Mr. Crafty Consumer decides to price each rug using a program on his SHARP computer. He writes the following program:

```

10 FOR R = 40 TO 1 STEP -3
20 P = .99 * (PI * R ^ 2)
30 PRINT R; " FOOT MODEL IS $"; P
40 NEXT R
50 END

```

and discovers that the rugs range in price from \$4976.2 to \$3.11.

Q. WAIT

The WAIT statement allows the programmer to change the operation of the print statement. Information displayed by a PRINT statement will remain on the display for the time period specified by the WAIT statement.

The format for the WAIT statement is:

WAIT argument

The argument is optional. If no argument is specified, the default time period is "infinite"; that is, the information will remain on the display until the user presses **ENTER**. This is the mode of operation used in most of our programs to this point.

If an argument is given all subsequent PRINT statements will "hold" their information on the display for a time period proportional to the number specified as an argument. This type of printing is similar to the PAUSE statement, except that the time period of the PAUSE statement is fixed. Notice that the WAIT statement has no effect on the operation of the PAUSE statement.

The number (or expression which results in a number) given as an argument must be in the range 0 to 65535. WAIT 0 causes information to be displayed so fast that it is virtually unreadable. WAIT 65535 will cause each PRINT statement to display its information for about 17 minutes! More practically, WAIT 64 gives a period of about a second, and WAIT 3840 about a minute. To reset the operation of the PRINT statement, so that it waits until the user presses enter, use the instruction WAIT with no argument.

Demonstration Program

This program illustrates the effect of the WAIT statement on subsequent PRINT statements. Here we vary the WAIT time from 0 to 102 by increments of 2 while printing periods in a loop. The BEEP is solely to aid you in comprehending the time interval, since some of the action happens too fast to be seen.

Program Listing:

```

10 FOR W = 0 TO 102 STEP 2
20 WAIT W
30 BEEP 1,5
40 PRINT ". ";
50 NEXT W
60 END

```

Keystrokes:

```

1 0 F O R W = 0 T O 1 0 2 S T E P
  2 ENTER
2 0 W A I T W ENTER
3 0 B E E P 1 SHIFT → 5 ENTER
4 0 P R I N T SHIFT " . " SHIFT " "
  SHIFT ; ENTER
5 0 N E X T W ENTER
6 0 E N D ENTER

```

R. READ, DATA and RESTORE

Not all data within a program must be entered by the user of the program. Often, useful data is relatively static, such as tax tables within a financial application or stress constants in an engineering application. These types of information may be embedded within a program, and utilized when needed, through the use of the DATA, READ, and RESTORE statements. These statements act in concert to specify the data used by a program, to transfer the data into variables, and to repeat the process as necessary.

The DATA statement consists of the keyword DATA followed by a list of data items. These include numbers, in real or scientific notation, and character strings. The items in the list are separated by commas. Data statements may appear anywhere within a program, but many programmers prefer to group them at the beginning of the program. This enables them to be found more easily when the program is read.

A typical DATA statement might resemble the following:

```
10 DATA "MOBY DICK", 20000, "WHITE", "M", 112
```

The READ statement consists of the keyword READ followed by a list of variable names. These may be numeric or character variable names. The variable names within the list are separated by commas. The READ statement causes an item, or items, of data to be "read" from a DATA statement and stored in the associated variables. A READ statement corresponding to our previous DATA statement is:

```
120 READ N$, WT, C$, SX$, L
```

SHARP insists that every time a READ statement is executed there be a corresponding data item within a DATA statement. Thus, the following program will produce an error on line 30 because all of the data items have been "used up" by the READ statement on line 20:

```
10 DATA 1, 2, 3
20 READ A, B, C
30 READ D
```

To correct this situation, we may add a data item to line 10:

```
10 DATA 1, 2, 3, 65
```

or we may use a separate DATA statement, anywhere within the program:

```
10 DATA 1, 2, 3
20 READ A, B, C
30 READ D
40 DATA 65
```

This illustrates that SHARP views all of the DATA statements within a program as a single list of data items. As the computer encounters each variable name within a READ statement, it assigns the next data item from the list to that variable. If SHARP cannot fulfill a request for a data item it stops the program and signals an error. Extra items which are unused when the program finishes in a normal manner, are ignored.

If the type (character or numeric) of the next item does not match the type of the variable to be filled, an error will occur. Good programmers group data items into separate DATA statements, each of which corresponds to its READ statement within the program. This is illustrated in the following program which reads three data items four times:

```
10 DATA 1, "A", 1
20 DATA 2, "B", 3
30 DATA 5, "C", 8
40 DATA 13, "D", 21
50 FOR I = 1 TO 3
60 READ A, A$, Z
70 T = T + A * Z
80 NEXT I
```

Lines 10 through 40 could have been written as:

```
10 DATA 1, "A", 1, 2, "B", 3, 5, "C", 8, 13, "D", 21
```

or even as:

```
10 DATA 1
20 DATA "A"
30 DATA 1
40 DATA 2
  ⋮
(etc)
```

Both of these alternate forms obscure the fact that three data items are read, each time, by the READ statement. The alternate forms also make it more difficult to verify the types of the data

items since the pattern of: number, character, number is not immediately discernible.

Sometimes it is desirable to re-use some or all of the values in the DATA statements. The RESTORE statement allows us to do this. RESTORE causes SHARP to re-use data items beginning at the first DATA statement of the program. Thus, any READ statements performed after the RESTORE will receive data items which were used previously.

The RESTORE statement may also specify a more selective "recycling" of data items. The statement:

```
RESTORE line-number
```

will cause data items to be re-assigned beginning at the DATA statement on the specified line. For example:

```
10 DATA .8, 4
15 DATA 1, 3, 6, 8E2
100 READ X, Y
110 READ M, N, O, P
120 RESTORE 15
130 READ A, B
```

When this program finishes, the values of the variables A and B will be 1 and 3, respectively.

In addition to line numbers, DATA statements may be "labeled" with a single character. The RESTORE statement may then be used to re-issue data items beginning at the DATA statement with the given label. An example of a labeled DATA statement is:

```
20 "A" : DATA 1, -12.2, 8
```

The following program segment RESTORES back to the DATA statement labeled "X":

```
10 DATA 4.2, 3, 1
20 "X" : DATA -2, 0, 3, 5
100 READ Q, Y, Z
110 READ MA, MB, MC, MD
120 RESTORE "X"
130 READ N, Z
```

At the end of the program N contains -2 and Z contains 0.

S. REM (REMARK)

The REM statement provides the capability to insert comments among the statements of a program. Although these comments are ignored by SHARP, they are extremely important because they assist other human beings to read your program. The more easily your program is read and understood, the more other programmers will wish to use it and, perhaps, improve it.

To lessen the typing burden, we have omitted comments in this manual. We recommend that you do NOT follow our example. Comments are probably most important when you are printing, saving, or sharing your programs. Do not rely on your mental ability to remember what the meaning of each variable in a program is, use a comment! If you don't, six months from now you will have forgotten.

Comments, following the REM keyword, may be inserted on their own line or at the end of another statement or series of statements. Comments may not appear before or in the middle of executable statements. The reason for this is that when SHARP sees the REM keyword it ignores any characters on the rest of the line. If REM were used at the beginning of a line, valid program statements would be ignored.

T. GOSUB... RETURN

As you begin to design programs, you will find that you utilize certain functions repeatedly within a single program. For example, such functions might include calculating the area of a circle or accepting and checking a number given by the user. Such repetition causes duplication of the statements which perform the function. To avoid this wasteful duplication, the programmer may use the GOSUB statement.

The GOSUB statement allows a group of other statements, which are used in several places within the program, to be set aside. This group of statements is called a "subroutine" (hence the term GOSUB). At each place in the program where the group of statements would occur, a GOSUB instruction is inserted.

The GOSUB statement instructs SHARP to begin executing the group of statements which have been set aside. This process is known as "calling a subroutine". Because the GOSUB statement causes a change in the normal sequential flow of execution, it is similar to the GOTO statement. The difference, however, is that before SHARP begins to perform the statements of the subroutine it "remembers" where it was. When the computer finishes performing the subroutine it returns to the point where it left off. This is known as "returning" from a subroutine.

But how does SHARP discern the end of the statements which form the subroutine? The answer is that you must inform it with the RETURN statement. The form of the GOSUB statement is:

GOSUB line-number

where line-number is the number of the first line of the subroutine. The form of the RETURN statement is simply:

RETURN

As an example of a subroutine in action, consider a program to compute and compare the area of two rectangles given the length and width of the sides:

Program Listing:

```
10 REM READ IN LENGTH AND  
20 REM WIDTH OF TWO RECTANGLES  
30 FOR I = 1 TO 2  
40 PAUSE "RECTANGLE "; I; ":"  
50 INPUT "ENTER LENGTH, WIDTH", L, W  
60 GOSUB 200  
70 IF I = 1 LET A1 = A  
80 IF I = 2 LET A2 = A  
90 NEXT I
```



```
100 REM PRINT AND COMPARE THE
110 REM AREAS OF THE RECTANGLES.
120 PRINT "AREA OF RECTANGLE 1 "; A1
130 PRINT "AREA OF RECTANGLE 2 "; A2
140 IF A1 > A2 THEN 170
150 PRINT "AREA OF 2 IS > AREA 1"
160 GOTO 180
170 PRINT "AREA OF 1 IS > AREA 2"
180 END
200 REM SUBROUTINE TO COMPUTE AREA
210 REM OF RECTANGLE GIVEN LENGTH
220 REM OF SIDES IN L AND W
230 A = L * W
240 RETURN
```

Notice where the subroutine is located. All subroutines should be placed after the final END statement of the main program. This prevents their accidental use by the normal process of sequential execution. A RETURN statement **MUST** terminate each and every subroutine.

A subroutine may include any legal statement and may perform any kind of processing desired. Good BASIC programmers design their programs as a set of pieces, or "modules". Usually a subroutine is used to encode each module. The main program is then used to control the order in which the subroutines are executed. For further information on this approach refer to one of the books on structured programming listed in Appendix F.

SUMMARY OF PROGRAM MODE EDITING FEATURES

1) Program lines should be numbered in intervals of, at least, ten. This will allow an additional line to be inserted between any two existing lines merely by choosing a line number which lies in between the line numbers of the existing lines. For example, to insert a line between lines 40 and 50 give the new line a number from the set 41 through 49.

2) To delete an existing line, simply type its line number and press **ENTER** .

NOTE: Because several program statements may be grouped together on a single line, great care must be exercised when deleting lines.

3) The Up Arrow and Down Arrow keys may be used to scroll (move up or down) through the current program a line at a time. Holding either Arrow key down will cause automatic repetition of the movement.

4) The LIST command may be used to proceed directly to a given line. The command LIST will display the first line of the first program in memory. LIST N (where N is a number), will display line N or, if no line numbered N exists, the first line whose number is greater than N is displayed.

5) Once a line is displayed, the Left and Right arrow keys may be used to position the cursor anywhere on the line. The INSert and DELete functions may then be used to affect changes.

NOTE: If changes are made to a line, it is necessary to press ENTER before displaying the next line or all changes will be nullified (not made).

- 6) If an error is encountered in a program during execution, the Up Arrow key will "remember" the line on which the error occurred. To recall the line, switch to the PROgram mode and press the Up Arrow key.

Note: The program data area is used for both a program and data. When the sum of program area and data area could be more than the capacity of the program data area, enter 65279 END in programming.

IV. ADVANCED CALCULATIONS

A. Scientific Notation

To enter a number in scientific notation ($A \times 10^B$), enter the mantissa, press the letter **E** and enter the exponent.

Example 1: To key in 6.7×10^8 :

Keystrokes	Display
[6] [.] [7]	6. 7_
[E]	6. 7E_
[8]	6. 7E8_

Example 2: To key in: -9.12×10^{-34} :

Keystrokes	Display
[-] [9] [.] [1] [2]	-9. 12_
[E]	-9. 12E_
[-] [3] [4]	-9. 12E-34_

Only the first 10 digits of the mantissa are significant (see example). For a number smaller than 1 but larger than -1 the data is accurate to a maximum of 10 digits.

Example 3: Key in 1234567898765:

Keystrokes	Display
[1] [2] [3] [4] [5] [6] [7] [8]	
[9] [8] [7] [6] [5]	1234567898765_
[ENTER]	1. 234567898E 12

Example 4: Key in 9.87654321234:

Keystrokes

9 . 8 7 6 5 4

3 2 1 2 3 4

ENTER

Display

9. 87654321234_

9. 876543212

Example 5: Key in 0.0000000002345678:

Keystrokes

. 0 0 0 0 0 0

0 0 0 0 2 3

4 5 6 7 8

ENTER

Display

. 0000000002345678_

2. 345678E-10

Example 6: Key in $0.00001234567 \times 10^{24}$:

Keystrokes

. 0 0 0 0 1 2

3 4 5 6 7 E 2 4

ENTER

Display

. 00001234567E24_

1. 234567E 19

Notice that for exponents, only the last two digits typed are effective.

Example 7: Key in 3×10^{123} :

Keystrokes

3 E 1 2 3

ENTER

Display

3E123_

3E 23

Example 8: Key in 4×10^{-3210} :

Keystrokes

4 E - 3 2 1 0

ENTER

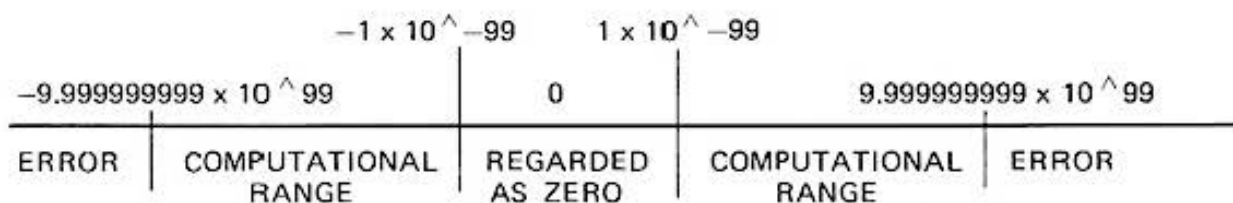
Display

4E-3210_

4E-10

B. Range of Calculations

Most machines have a range of numbers with which they can operate. In the PC-1500, this range is any number between $9.999999999 \times 10^{99}$ and $-9.999999999 \times 10^{-99}$. When a number exceeds this range it becomes too large for the computer to handle and an "overflow" condition (signalled by error 37) occurs. There is also an "underflow" condition; the point at which a number becomes too small. An underflow condition will not be signalled; no error message or halt will occur. Any number which falls into the range of -1×10^{-99} to 1×10^{-99} will be regarded as zero. This is illustrated by the following chart:



Example 1: If you try to solve the equation $(5.67 \times 10^{55}) * (8.90 \times 10^{65})$, you will cause an overflow:

Keystrokes

(5 . 6 7 E 5 5) *

(8 . 9 0 E 6 5)

ENTER

Display

(5. 67E55) * (8. 90E65)_

ERROR 37

Error 37 indicates calculation overflow.

C. Root, Power, Pi

Root

Example 1: To find the square root of 73:

Keystrokes

SHIFT $\sqrt{\quad}$ 7 3

ENTER

Display

$\sqrt{73}_$

8. 544003745

Example 2: To find $\sqrt[4]{256}$:

Keystrokes

SHIFT $\sqrt{\quad}$ SHIFT $\sqrt{\quad}$ 2 5 6

ENTER

Display

$\sqrt{\sqrt{256}}_$

4

Example 3: To find $\sqrt{3^2 + 4^2}$:

<u>Keystrokes</u>	<u>Display</u>
SHIFT $\sqrt{\quad}$ (3 * 3 +	
4 * 4)	$\sqrt{(3*3+4*4)}$ _
ENTER	5

This equation can also be computed in the following manner:

Example 4:

<u>Keystrokes</u>	<u>Display</u>
SHIFT $\sqrt{\quad}$ (3 SHIFT \wedge 2 + 4	
SHIFT \wedge 2)	$\sqrt{(3^2+4^2)}$ _
ENTER	5

Power

The power, or exponentiation function, permits you to raise a number to a power.

Example 1: Calculate 4^3 (= 4 x 4 x 4):

<u>Keystrokes</u>	<u>Display</u>
4 SHIFT \wedge 3	4^3 _
ENTER	64

Example 2: Calculate $3^{3.2} \times 4^{-2.4}$:

<u>Keystrokes</u>	<u>Display</u>
3 SHIFT \wedge 3 . 2 *	
4 SHIFT \wedge - 2 . 4 *	$3^{3.2} * 4^{-2.4}$ _
ENTER	1. 207380162

Example 3: Calculate $4^{(3^2)}$:

<u>Keystrokes</u>	<u>Display</u>
4 SHIFT \wedge 3 SHIFT	
\wedge 2	4^{3^2}
ENTER	262144

PI π

The value of Pi (3.141592654) is stored in both the symbols PI and π as a fixed constant. Either symbol can be used in calculations where the value of Pi is needed.

As an example, to find the area of a rug which has a diameter of five feet, in RUN mode key:

Keystrokes	Display
P I * (5 / 2) ^ 2	PI * (5/2) ^2_
SHIFT ^ 2	19. 63495408
ENTER	

D. Angular Modes

The PC-1500 allows angular functions to be calculated in any of three angular modes as follows:

To set the PC-1500 into Degree mode type:

DEG. **ENTER**
(DEG will appear at top of the display)

To set the PC-1500 into Radians mode type:

RAD. **ENTER**
(RAD will appear at top of the display)

To set the PC-1500 into Grads mode type:

GRA. **ENTER**
(GRAD will appear at top of the display)

E. Trigonometric Functions

The six trigonometric functions provided on the PC-1500 are SIN, COS, TAN, ASN, ACS, and ATN. Each function can be calculated in either the GRAD, DEG, or RAD mode. Execution is as follows:

DEG. ENTER	→	Sets mode to degree
SIN 30 ENTER	→	SIN 30 in degrees
CL	→	Clears display
GRA. ENTER	→	Sets mode to grad.
SIN 30 ENTER	→	SIN 30 in grad
CL	→	Clears display
RAD. ENTER	→	Sets mode to radians
SIN 30 ENTER	→	SIN 30 in radians

In the above examples, the sine of 30 is computed in each mode, producing three different (but equivalent) answers. Inverse trigonometric functions can also be performed as follows:

RAD.	<input type="button" value="ENTER"/>	—————→	Sets mode
ASN -0.5	<input type="button" value="ENTER"/>	<input type="text" value="-5. 235987756E-01"/>	Arcsine of -.5
DEG.	<input type="button" value="ENTER"/>	—————→	Sets mode
ASN -0.5	<input type="button" value="ENTER"/>	<input type="text" value="-30"/>	Arcsine of -.5
ACS (-.5 + .1)	<input type="button" value="ENTER"/>	<input type="text" value="113. 5781785"/>	Arccosine
ATN (7/3)	<input type="button" value="ENTER"/>	<input type="text" value="66. 80140949"/>	Arctangent

F. Logarithmic Functions

LN, LOG

The function LN will compute the natural logarithm (base e) while the function LOG will compute the common logarithm (base 10). These are executed in the RUN mode as follows:

LN 7.4	<input type="button" value="ENTER"/>	<input type="text" value="2. 00148"/>
LOG 7.4	<input type="button" value="ENTER"/>	<input type="text" value="8. 692317197E-01"/>
LN 25	<input type="button" value="ENTER"/>	<input type="text" value="3. 218875825"/>
LOG 100	<input type="button" value="ENTER"/>	<input type="text" value="2"/>

EXP

The reverse function of LOG is a number raised to a power of 10. For example:

LOG 100	<input type="button" value="ENTER"/>	<input type="text" value="2"/>			
10	<input type="button" value="SHIFT"/>	<input type="button" value="^"/>	2	<input type="button" value="ENTER"/>	<input type="text" value="100"/>

Because the natural log (LN) is not based on a power of 10 but on a power of e, a reverse function is necessary. This function is EXP.

Example:

LN 7.4	<input type="button" value="ENTER"/>	<input type="text" value="2. 00148"/>
EXP 2.00148	<input type="button" value="ENTER"/>	<input type="text" value="7. 399999998"/>

G. Angle Conversion

The PC-1500 performs conversions of angles from DMS (Degree, Minutes, Seconds) to DEG (Decimal Degree) form. When converting Decimal Degrees to the Degree, Minutes, Seconds equivalent, the answer is comprised of an integer portion representing degrees, and a fractional portion of which the 1st and 2nd decimal digits represent the minutes, and the 3rd and 4th decimal digits the seconds. The 5th through end decimal digits are decimal degrees. To convert an angle given in degree, minutes, seconds into decimal degree form, it must be entered in integer, decimal order.

Example 1: Convert 16.1932 Decimal Degrees into DMS form:

DMS 16.1932 16. 1 13552

Example 2: Convert 32.2513 DMS into Decimal Degree form:

DEG 32.2513 32. 42027778

H. Miscellaneous Functions

ABS

The ABS function derives the absolute value of a numerical value or variable.

Example 1:

ABS (25 - 86) 61

Normally, $25 - 86 = -61$. The ABS function takes the actual difference of the numbers to get 61.

INT

The INT function rounds a numeric value to the largest integer not larger than the numeric value itself.

Example 1:

$(25/3) + 7$ 15. 33333333

INT $(25/3) + 7$ 15

Example 2:

$(31.62 + 21.18)$ 52. 8

INT $(31.62 + 21.18)$ 52

In Example 2, the answer is not rounded up from 52.8 to 53 because that would make the answer larger than the original value.

NOTE: Don't forget the order of evaluation. Parentheses should be used if you wish the INT of the resultant expression. For example, let us alter the previous example, leaving off all parentheses:

Example 3:INT 31.62 + 21.18

The computer takes that INT of the first number 31.62 (result: 31) and adds that to 21.18 to get 52.18. This is slightly different from the first computation, eh?

SGN

For any number X the SGN function returns a value indicating if the number is negative, zero, or positive. The values are the following:

1 if $X > 0$
 0 if $X = 0$
 -1 if $X < 0$

Examples:5 - 10 SGN (5 - 10) 12 - 4 SGN (12 - 4) 15 - 15 SGN (15 - 15)

V. ADVANCED PROGRAMMING

A. ARRAYS and the DIM Statement

Most of our sample programs to date have used a small number of variables. As you begin to utilize the full processing potential of the PC-1500 you will discover that variables which hold a single number can have drawbacks. Perhaps, for example, you are considering a program which reads in fifty numbers and sorts them. You may quickly conclude that, although the PC-1500 has more than enough potential variables, there must be an easier way. There is, and it is called an "array variable".

An array is simply a group of consecutive storage areas, or "locations", with a single name. Each storage area can hold a single number or each storage area can hold a character string. All of the storage areas within a given array must hold the same type of data.

The number of locations within a single array may be as many as 256 and is determined by your specification. Thus, if you define a numeric array with 50 locations, this allows you to store up to 50 numbers in association with a single name. If you define an array of strings (called a

"character array") you may also specify the size of the strings, up to a maximum of 80 characters per string. Creating character strings of varying sizes is described in Section B.1.

To define an array, the DIM (short for dimension) statement is used. Arrays must always be "declared" (defined) before they are used. (Not like the single-value variables we have been using.) The form for the numeric DIMension statement is:

DIM numeric-variable-name (size)

where:

numeric-variable-name is a variable name which conforms to the normal rules for numeric variable names previously discussed.

size is the number of storage locations and must be a number in the range 0 through 255. Note that when you specify a number for the size you get one more location than you specified.

Examples of legal numeric DIMension statements are:

```
DIM X (5)
DIM AA (24)
DIM Q5 (0)
```

The first statement creates an array X with 6 storage locations. The second statement creates an array AA with 25 locations. The third statement creates an array with one location and is actually rather silly since (for numbers at least), it is the same as declaring a single-value numeric variable.

It is important to know that an array-variable X and a variable X are separate and distinct to SHARP. The first X denotes a series of numeric storage locations, and the second a single and different location.

Now that you know how to create arrays, you might be wondering how it is that we refer to each storage location. Since the entire group has only one name, the way in which we refer to a single location (called an "element") is to follow the group name with a number in parentheses. This number is called a "subscript". Thus, for example, to store the number 8 into the fifth element of our array X (declared previously) we would write:

```
X (4) = 8
```

If the use of 4 is puzzling, remember that the numbering of elements begins at zero and continues through the size number declared in the DIM statement.

The real power of arrays lies in the ability to use an expression or a variable name as a subscript. For example, to create a table containing the squares of the numbers 0 to 9 we could write the following statements:

```
10 DIM SQ (9)
20 FOR I = 0 TO 9
30 SQ (I) = I * I
40 NEXT I
```


In this example, the variable I is used to select which storage location will hold the result and it is also being used to compute the result.

To declare a character array a slightly different form of the DIM statement is used:

DIM character-variable-name (size) * length

where:

character-variable-name is a variable name which conforms to the rules for normal character variables as discussed previously.

size is the number of storage locations and must be in the range 0 through 255. Note that when you specify a number, you get one more location than you specified.

*length is optional. If used, it specifies the length of each of the strings that comprise the array. Length is a number in the range 1 to 80. If this clause is not used, the strings will have the default length of 16 characters.

Examples of legal character array declarations are:

```
DIM X$ (4)
DIM NMS (10) * 10
DIM IN$ (1) * 80
DIM R$ (0) * 26
```

The first example creates an array of five strings each able to store 16 characters. The second DIM statement declares an array NM with eleven strings of 10 characters each. Explicit definition of strings smaller than the default helps to conserve memory space. The third example declares a two element array of 80-character strings and the last example declares a single string of twenty-six characters (see Section B.1.).

Besides the simple arrays we have just studied, the PC-1500 allows "two-dimensional" arrays. By analogy, a one-dimensional array is a list of data arranged in a single column. A two-dimensional array is a table of data with rows and columns. The two-dimensional array is declared by the statement:

DIM numeric-variable-name (rows, columns)

or

DIM character-variable-name (rows, columns) * length

where:

rows specifies the number of rows in the array. This must be a number in the range 0 through 255. Note that when you specify the number of rows you get one more row than the specification.

columns specifies the number of columns in the array. This must be a number in the range 0 through 255. Note that when you specify the number of columns you get one more column than the specification.

The following diagram illustrates the storage locations that result from the declaration DIM T (2, 3) and the subscripts (now composed of two numbers) which pertain to each storage location:

	column 1	column 2	column 3	column 4
row 0	T (0, 0)	T (0, 1)	T (0, 2)	T (0, 3)
row 1	T (1, 0)	T (1, 1)	T (1, 2)	T (1, 3)
row 2	T (2, 0)	T (2, 1)	T (2, 2)	T (2, 3)

NOTE: Two-dimensional arrays can rapidly eat up storage space. For example, an array with 25 rows and 35 columns uses 875 storage locations!

Arrays are very powerful programming tools. For a more complete treatment of arrays, we recommend supplementary reading.

B. MORE ON CHARACTER STRINGS

B.1. DIMensioning Strings

Character strings are limited, by default, to sixteen characters in length. By dimensioning a character string it is possible to create a string whose length is up to 80 characters. Reductions in string length, to conserve memory space, are also possible.

The length of a string is specified in the DIMension statement as follows:

DIM variable-name (bound) * length

where:

variable-name is the name of the character string array.

bound is the maximum subscript of the array.

length is the length of each string within the array.

If only one string is needed, an array with one element (subscripted as element zero) may be specified to conserve memory space. This is illustrated by the following declaration of a 26 character string:

```
DIM AS (0) * 26
```

B.2. Concatenation

Several character strings (or characters within character variables) can be joined to form a single string. This "adding" of character strings is called "concatenation". The form for concatenation is:

$$\text{variable} = \frac{\text{character-string}}{\text{character-variable}} + \frac{\text{character-string}}{\text{character-variable}}$$

Example 1:

```
10 SS = "SUPER"
20 TS = SS + "MAN"
30 PRINT TS
```

Output:

SUPERMAN

In line 20, the contents of the variable S\$ ("SUPER") is "added" to the string "MAN". Notice that no space is inserted during concatenation. Several strings may be concatenated in the same expression as in the following example:

Example 2:

```
10 AS = "ER"
20 BS = "AND"
30 CS = "MOTH"
40 DS = "GR"
50 SS = "WRITE YOUR "+DS+BS
60 PRINT SS + CS + AS
```

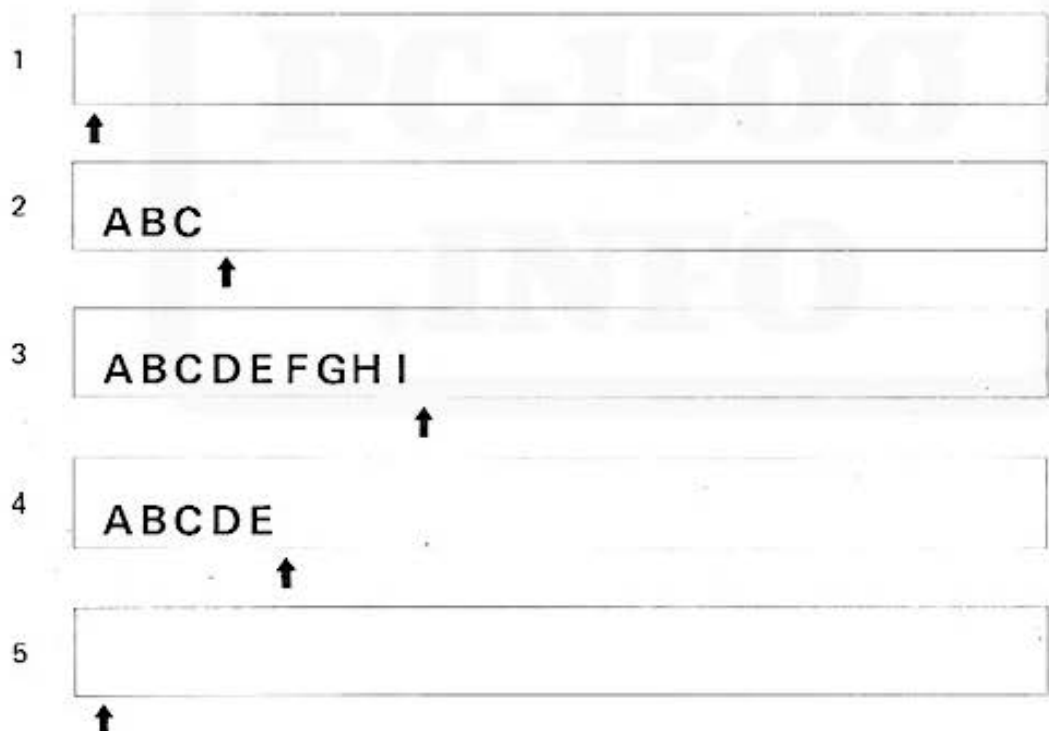
Output:

WRITE YOUR GRANDMOTHER

When concatenation operations are performed, an internal temporary character storage area is used to build the new string. This storage area has a capacity of 80 characters. If the new string exceeds this length, an ERROR 15 will occur. An illustration of this area during a concatenation operation follows:

Example 3:

```
X = LEN ("ABC" + LEFT$ ("DEFGHI", 2)) 
```



NOTE: The ↑ symbol represents an internal character pointer which keeps track of the amount of storage used.

- 1) At the start of execution, the storage area will be cleared and the character pointer will be reset to the starting position.

- 2) "ABC" is entered into the area, taking up the first three positions.
- 3) "DEFGHI" is added to the character storage area, following "ABC".
- 4) The LEFT\$ function acts upon the string "DEFGHI" to extract "DE", which then replaces "DEFGHI" in the area.
- 5) The assignment is performed and the storage area is again cleared.

B.3. String Comparison

Character strings may be compared to determine which string is "greater" or "less than" the other. These determinations are based on the Collating Sequence (given in Appendix C) which is the order of all the characters recognized by the computer.

If the strings contain an unequal amount of characters, the shorter string is "padded" (filled out) with NULL characters (ASCII 0). The operators which are legitimate for comparison of strings are:

- = True if the two strings are equal in length and contain the same characters in the same order.
- <> True if the two strings differ in length, characters, or ordering of characters.
- > True if the characters of the first string are "greater" (occur later in the ordering) than the characters in the second string.
- < True if the characters of the first string are "less than" (occur first in the ordering) than the characters in the second string.

The format for string comparison is:

$$\frac{\text{character string}}{\text{character variable}} \quad | \quad \text{OP} \quad | \quad \frac{\text{character string}}{\text{character variable}}$$

where OP is one of the comparison operators listed above.

Examples:

"MARY" > "MARI" is True
 "MARY" = "MARY " is False
 "abc" <> "ABC" is True
 "DATA 1" < "DATA 2" is True
 "?" < "#" is False

Note: The form of AS <= BS, AS >= BS cannot be used for comparing character strings. Comparison is possible however, in the forms of (AS < BS) OR (AS = BS) and (AS > BS) OR (AS = BS).

C. FUNCTIONS

C.1. ASC

There are two functions used in the conversion of characters to and from the ASCII code. The function ASC converts a single character into its ASCII decimal code. The reverse function CHR\$ converts the ASCII decimal code into a single character string.

ASC { "character"
char variable name

The argument in this function is any character string or a character string variable name. The value returned by this function is the corresponding ASCII code for the first character of the specified string.

Example 1:

```
10 LET XS = "PATTI"  
20 LET A = ASC XS  
30 PRINT A
```

Output:

```
                                RUN                                •  
                                                                80
```

In the above example XS is assigned the value of the character string "PATTI". The ASC function takes the first character (P) and converts it to its ASCII code (80).

Example 2:

```
10 PRINT ASC "K"
```

Output:

```
                                RUN                                •  
                                                                75
```

ASC "K" returns the ASCII code for "K" which is 75.

C.2. CHR\$

CHR\$ is the complement of the ASC function. The CHR\$ function takes an ASCII decimal code, from 0 through 127, and returns the character string equivalent. (Note: Some codes represent special characters which do not print)

CHR\$ { ASCII decimal code
numeric variable

Example 1:

```
10 PRINT (CHR$ 67) + "OP"
```

Output:

```
COP                                RUN                                •
```


Example 2:

```
10 Z = 65
20 PRINT CHR$ Z
```

Output:

```

A                                     RUN

```

The first example shows the use of an ASCII decimal code as an argument. The result is concatenated to the string "OP" producing "COP". The second example assigns a numeric value to the variable Z. The variable name is then used as the argument, resulting in the character "A".

Our next sample program converts the upper case characters in a text string to lower-case using both the ASC and the STR\$ functions:

Example 3:

```

5 WAIT 0
10 INPUT "ENTER MESSAGE", MS
20 FOR I = 1 TO LEN (MS)
30 T$ = MIDS (MS, I, 1)
40 L = ASC (T$)
45 IF (L < 65) OR (L > 90) THEN 60
50 T$ = CHR$ (L + 32)
60 PRINT T$;
70 NEXT I
80 WAIT : PRINT

```

C.3. INKEY\$

This function takes in any character from the keyboard and stores it in the specified variable. There is no need to press **ENTER** because the character will be automatically accepted.

```
variable = INKEY$
```

During execution of this statement a prompt character is not displayed unless a previous PRINT statement is used. The input character is not echoed back to the display and the display remains unaffected.

Example:

```

1Ø WAIT Ø
2Ø A$ = INKEY$
3Ø IF A$ = " " THEN PRINT "NO KEY": GOTO 2Ø
4Ø PRINT A$
5Ø GOTO 2Ø

```

This function will only accept one character. If more than one is keyed, only the first character will be read in; all others will be ignored.

C.4. LEN

While manipulating characters, it is desirable to know the number of characters in a string. This can be done by the use of the LEN function. It returns the number of characters in a specified expression or character variable.

```
LEN { "character string"
      character variable name
```

Example 1:

```
10 AS = "CATHY"
20 C = LEN AS
30 PRINT C
```

Output:

```

                                     RUN                                     I .
                                                                                   5
```

Example 2:

```
10 C = LEN "CAT"
20 PRINT C
```

Output:

```

                                     RUN                                     I .
                                                                                   3
```

If LEN is used on an empty string (i.e. nothing is enclosed in the quotation marks), zero will be returned.

Example 3:

```
10 A = LEN " "
20 PRINT A
```

Output:

```

                                     RUN                                     I .
                                                                                   0
```

C.5. LEFT\$

There are three functions used to select or extract specified sections of a character string. LEFT\$ extracts characters from the left, RIGHT\$ from the right and MID\$ from the middle.

```
LEFT$ { ("character string", number)
        (character variable name, number)
```

The "number" argument specifies how many characters to extract beginning from the left side.

Example 1:

```
10 A$ = LEFT$ ("DRESSER", 5)
20 PRINT A$
```

Output:

```
                RUN                I  •
DRESS
```

Example 2:

```
10 B$ = "THINK BIG"
20 A$ = LEFT$ (B$, 4)
30 PRINT A$
```

Output:

```
                RUN                I  •
THIN
```

In both examples, starting from the left side of the character string, characters are extracted and stored in A\$. Printing A\$ results in "DRESS" and "THIN" respectively.

C.6. MID\$

To extract the middle portion of a character string the function MID\$ is used.

```
MID$ { ("character string", expression, expression)
      ( character string variable, expression, expression)
```

Example 1:

```
10 A$ = "I NEED HELP"
20 B$ = MID$ (A$, 3, 4)
30 PRINT B$
```

Output:

```
                RUN                I  •
NEED
```

Example 2:

```
10 T$ = MID$ ("(415) 743 - 1602", 6, 3)
20 PRINT T$
```

Output:

```
                RUN                I  •
743
```

The first argument to this function is a character string or a character string variable. The second argument is a number representing the first character to be extracted. The third argument is the total number of characters, including the first, to be extracted.

In the first example, "I NEED HELP" is stored in A\$. MID\$ extracts four characters beginning at the third, and places these into B\$. When B\$ is printed, it is found to contain "NEED".

In the second example, a string containing a telephone number is the first argument. The sixth character ("7") is located and, together with the following two characters, is stored in the variable TS. When printed, the result is "743".

C.7. RIGHTS

The RIGHTS function works much like LEFT\$, the only difference being that it starts from the opposite (right) end of the string. The arguments are the same as LEFT\$:

$$\text{RIGHT\$} \left\{ \begin{array}{l} (\text{"character string", number}) \\ (\text{character variable, number}) \end{array} \right.$$

The "number" argument to this function specifies how many characters to extract from the character string beginning on the right side.

Example 1:

```
10 XS = "READ ONLY MEMORY"
20 YS = RIGHTS (XS, 6)
30 PRINT YS
```

In this program the RIGHTS function takes six characters from the right end of the string and stores them in the variable Y\$. The content of Y\$ is now "MEMORY".

C.8. RND

There may be times when you want to provide your program with random numbers. The RND functions allows the computer to generate random numbers in a range from one to a specified number. (Note: The range always starts with one)

Example 1:

```
10 A = RND 5
```

In this example A could have any one of the values one through five, inclusive. If you want random numbers in a range which begins with a number other than 1, (for example 40 to 50) you will have to simulate this by generating random numbers from 1 to 10 and adding a constant (39 in our example):

Example 2:

```
10 FOR I = 1 TO 5
20 B = 40 + RND 10
40 PRINT B:
50 NEXT I
```

Output:

RUN
42 44 47 48 42

C.9. RANDOM

Random numbers are generated by a mathematical formula and are accessible by using the RND function. Whenever the computer is turned ON, a series of random numbers is generated by the computer. This list remains unchanged unless the RANDOM function is used. This means that a program will use the same series of "random" numbers each time the computer is turned on. To prevent this, the RANDOM function resets the "seed" used by the formula to generate its random numbers.

Example 1:

```
10 FOR I = 1 TO 5
15 A = RND 3
20 PRINT A;
30 NEXT I
```

Output: 1 2 2 3 1Example 2:

```
10 FOR I = 1 TO 5
15 A = RND 3
20 PRINT A;
30 NEXT I
```

Output: 1 2 2 3 1

To get true random numbers in this case the function RANDOM should appear before the RND statement. This function sows a new seed in the generation of random numbers and thus causes the numbers to differ. Accordingly, a program run under identical conditions will produce varied output:

Example 1:

```
10 RANDOM
15 FOR I = 1 TO 5
20 A = RND 3
30 PRINT A;
40 NEXT I
```

Output: 3 1 2 2 3Example 2:

```
10 RANDOM
15 FOR I = 1 TO 5
20 A = RND 3
30 PRINT A;
40 NEXT I
```

Output: 2 3 1 3 2

C.10. STR\$

STR\$ works in a manner opposite to VAL. It will convert an internal numeric variable back to its character string representation.

Example:

```
10 INPUT "ENTER A NUMBER "; I
20 SS = STR$ (I)
30 PAUSE "THAT NUMBER IS"
40 PRINT "THE STRING "; SS
```

The above program accepts a number, forms the character representation of that number and stores it in the character variable SS. Because the internal numeric representation cannot be displayed it is automatically converted back into a character string by the PRINT statement.

C.11. STATUS

To display how much program memory you have left available or how much memory a program uses, the STATUS function is used. STATUS 0 will display how many program "steps" are still available. STATUS 1 displays the number of steps already used. The maximum number of steps available is 1850.

Example:

STATUS 0

Output:

RUN	1 770
-----	-------

STATUS 1

Output:

RUN	81
-----	----

The instruction STATUS 0 is equivalent to the MEM command of the PC-1211. MEM remains a valid command on the PC-1500 also.

C.12. TIME

To display or set the month, date, and hour the TIME function is used in the following manner:

Setting: TIME = MMDDHH.MMSS

Display: TIME

where: MM represents two digits for the month, DD two digits for the day, and HH two digits for the hour. The fractional portion defines the minutes (MM) and seconds (SS).

When displaying TIME, the output will be in the same format just given. The result of the function can be handled in the same manner as a number and can also be used freely in expressions.

The following program simulates a clock. Be sure to set the time before running the program.

Program Listing:

```

10 WAIT 0
20 AS = STR$ TIME
30 IF TIME > 99999 THEN 50
40 AS = "0" + AS
50 MS = LEFT$ (AS, 2)
60 DS = MID$ (AS, 3, 2)
70 HS = MID$ (AS, 5, 2)
80 DSS = MS + "/" + DS + "/" + HS
90 DS = VAL (MS + DS + "00")
100 PRINT DSS;
110 T = TIME - DS
    
```

```
120 IF T >= 1 THEN 140
130 T = T + 12
140 IF T > 23.5959 GOTO 20
150 CURSOR 18 : PRINT T
160 GOTO 110
```

C.13. VAL

VAL and STRS are complementary functions which convert character strings to and from a numeric variable. The function VAL converts a string containing the character representation of a number into a number, which is then stored in a variable.

NOTE: When anything other than a digit 0 thru 9, . (decimal point), + (positive sign), - (negative sign), or E (scientific notation) is used in the expression, conversion will end with the illegal character.

Example 1:

```
10 ZS = VAL "-37"
```

Result:

ZS contains the number -37

Example 2:

```
10 A = VAL "237.6"
```

Result:

A contains the number 237.6

D. PRINT USING

The USING statement allows a programmer to rigidly control the format of information on the display. This allows standardized displays and prevents loss of information.

When the USING clause appears, alone or within a PRINT or PAUSE statement, it defines the format for all subsequent PRINT or PAUSE statements until the next USING clause is encountered in the program.

Several USING clauses may appear within a single PRINT or PAUSE statement. In this case each one defines the format to be used to print the listed variables until the next USING clause is encountered.

A format is specified via a string of special characters called an "editing string". The characters within the editing string define the areas of the display available for information and restrict the type of information which may be printed in these areas. This scheme is the same general scheme employed by other languages such as COBOL and PL/I.

An editing string may be stored in a string variable. The variable's name would then replace the editing string within the USING clause. This allows multiple formats which are selected under program control.

The characters which may be employed within editing strings are summarized below:

TABLE OF EDITING CHARACTERS

<u>Character</u>	<u>Use</u>
#	Specifies a numeric field. Numbers are right-justified within this field. If the field width is not sufficient to hold the number, an ERROR 36 will occur. Leading zeroes are converted to blanks.
*	Specifies Asterisk Fill of the specified positions of a numeric field which do not contain data.
.	Causes a decimal point to be displayed within a numeric field.
,	Used at the beginning of a numeric field to specify the insertion of commas after every three digits.
^	Used within a numeric field to cause the number to be displayed in scientific notation.
+	Used in a numeric field to force printing of the sign of the data.
&	Specifies a character field. Characters are left-justified within the field. If the field width is not sufficient to hold the data string, the string is truncated.

NOTE: The width of a numeric field must always be one more than the width of the data to allow for the sign of the data. This is true regardless of whether you use the + editing character or not.

NOTE: The use of the comma requires that you insert one extra # for each comma in the editing string.

Examples

```
X = PI  Y = 1234  AS = "ABCDEF"
```

```
PRINT USING "###"; X
```

3

```
PRINT USING "+###.###"; X
```

+3.141

```
PRINT USING "###.##^"; X
```

3.14E 00

PRINT USING "###.^"; X

3.E 00

PRINT USING "*#####"; Y

**1234

PRINT USING "***##"; Y

1234

PRINT USING "&&&&&#####"; AS; Y

ABCDEF 1234

PRINT USING "&&&"; AS

ABC

10 US = "*#####.##"

20 USING US

30 PRINT Y; "S"

**1234.00\$

PRINT X; "\$"

*****3.14\$

PRINT USING; AS; X

ABCDEF 3.141592654

PRINT USING "###, ###, ###"; 246813

246, 813

Note: Use the number of # (including *) marks for variable (integer) designation in the following range:

With a 3-digit punctuation (,) not used: Within 11 (including sign)

With a 3-digit punctuation (,) used: Within 14 (including sign)

This computer has 10 significant digits for the numbers.

When the format exceeding 10 integers is designated by the USING statement and the figure exceeding 10 integers is displayed (printed) by the PRINT (LPRINT) statement, the displayed (printed) number may be incorrect.

Example:	RUN mode USING "#####" PRINT 888888888888 (LPRINT 888888888888)	Display → > → 888888888800 → 88888888880)
----------	--	---

12 digits

E. Computed Control Transfer

In addition to the basic control statements described in Chapter III, the PC-1500 provides two other control statements of great utility. These are the ON GOSUB and the ON GOTO. As you might guess from their names, these statements act like the GOSUB and GOTO statements discussed previously. The difference, however, lies in their ability to transfer control (i.e. to execute statements at a different location) automatically. That is, the GOTO or GOSUB functions will "GO" to one of several statements (or subroutines) depending on the value of a numeric variable. This dependence on a variable for guidance is what gives the ON statements the nickname "computed control statements".

The ON statements have the form:

<u>ON</u> expression	{	GOTO line #1, line #2, line #3, ... (etc) GOSUB " " "
----------------------	---	--

The expression which follows the ON keyword must evaluate to a positive integer greater than zero and less than the number of line-numbers listed after the GOTO or GOSUB keyword. During execution, when the computer encounters an ON statement, it transfers the flow or execution to the line-number which corresponds to the value of the expression.

A typical ON statement might be:

ON TX GOSUB 100, 200, 250, 300

In this case, the variable TX MUST contain a number in the range 1 through 4 because there are only four line-numbers listed. Any other number in TX will result in an error since there is no corresponding line-number. For this reason, it is important to include sufficient tests (IF statements) to insure that your expressions result in a valid number.

The ON statements are very useful for automating a series of choices. For example, consider the following program fragment which allows the user to select one of several tax tables. Without the ON statement this might be written:


```
10 PAUSE "SPECIFY TAX TABLE TO USE:"
20 PAUSE "(1) SINGLE,"
30 PAUSE "(2) MARRIED,"
40 INPUT "(3) BUSINESS ?"; TT
50 IF (TT < 1) OR (TT > 3) THEN 10
60 REM USE APPROPRIATE TABLE
70 IF TT = 1 THEN 220
80 IF TT = 2 THEN 300
90 IF TT = 3 THEN 450
(etc)
```

Using the ON statement we can consolidate lines 70, 80, and 90 into a single statement:

```
70 ON TT GOTO 220,300,450
```

ON ERROR GOTO

Using another form of controlled transfer allows a program to detect when an error occurs. After detection, the program may execute statements which attempt to recover from the error. Such statements may inform and instruct the user, or they may save valuable data.

The ON ERROR GOTO statements instructs SHARP where to go upon detecting the occurrence of an error. The form of this statement is:

```
ON ERROR GOTO line-number
```

where line-number is the number of a program line containing instructions to be followed in the event of an error.

F. DISPLAY PROGRAMMING

The display window incorporated into the PC-1500 is a remarkably flexible output device. To allow programmers to exploit the full power of the display several new statements have been added to the dialect of BASIC used by the PC-1500. These extensions are described in this section.

The display itself utilizes liquid crystal technology to display up to 26 characters at a time. Each character in the computer's character set occupies a 5 x 7 dot matrix. Utilizing the GPRINT command, programmers may develop and display their own characters.

For graphic purposes, the entire display field may be utilized as a 7 x 156 dot matrix. Individual dots within any of 156 columns may be energized to create graphics, figures, or special symbols. The POINT command allows "sensing" of any column to discover which dots are currently energized.

A speaker and tone generator allow the programmer to add the dimension of sound to the man-machine interaction. Tones may be created at any of 256 frequencies (range 230Hz to about 7KHz). Automatic repetition of a tone and control of the duration of a tone are also possible under program control.

F.1. BEEP

The BEEP statement allows the programmer to create tones for game playing, error signalling, and other interactive applications. The format of the BEEP statement which creates sound is:

```
BEEP expression 1 , expression 2 , expression 3
```

where:

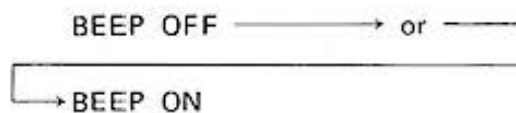
expression 1 is the only required parameter and specifies how many times the beeping tone is repeated. The allowable range is 0 to 65535 repetitions.

expression 2 is optional and specifies the frequency of the tone(s). This is a number between 0 and 255.

expression 3 is also optional and specifies the duration of each tone. This duration is specified as a number in the range 0 to 65279.

The BEEP statement can also be used to turn off and on the PC-1500's internal speaker. Thus, the noise made by a cassette SAVE or LOAD operation may be eliminated.

The format of the BEEP statement which controls the internal speaker is:



NOTE: When the PC-1500 is turned OFF and then ON the speaker is restored to an active mode.

Demonstration Program

```
10 D = 60  
20 DATA 14  
30 DATA 245, 1, 245, 1, 160, 1, 160, 1  
40 DATA 143, 1, 143, 1, 160, 2  
50 DATA 180, 1, 180, 1, 195, 1, 195, 1  
60 DATA 220, 1, 220, 1, 245, 2  
100 READ X  
110 FOR I = 1 TO X  
120 READ N, S  
130 BEEP 1, N, (D * S)  
140 NEXT I  
150 END
```

F.2. CURSOR

The CURSOR statement positions the cursor at one of the 26 character positions available on the display. The form of this statement is:

CURSOR position-expression

where:

position-expression evaluates to a number in the range 0~25 which specifies to where the cursor will move.

The normal use of the CURSOR command is to position the cursor preparatory to printing some information. Used in this manner, it allows the programmer to define his own separation between data items as in the following statements:

Program Listing:

```

10: WAIT 20
20: X = 8
30: PRINT "A"
40: CURSOR 4
50: PRINT "B"
60: CURSOR X
70: PRINT "C"
80: CURSOR ((X^2) / 4) - 4
90: PRINT "D"
100: END
    
```

Keystrokes:

```

1  [ ] [ ] W [ ] A [ ] I [ ] T [ ] 2 [ ] [ ] ENTER
2  [ ] [ ] X [ ] = [ ] 8 [ ] ENTER
3  [ ] [ ] P [ ] R [ ] I [ ] N [ ] T [ ] SHIFT [ ] " [ ] A [ ] SHIFT [ ] " [ ] ENTER
4  [ ] [ ] C [ ] U [ ] R [ ] S [ ] O [ ] R [ ] 4 [ ] ENTER
5  [ ] [ ] P [ ] R [ ] I [ ] N [ ] T [ ] SHIFT [ ] " [ ] B [ ] SHIFT [ ] " [ ] ENTER
6  [ ] [ ] C [ ] U [ ] R [ ] S [ ] O [ ] R [ ] X [ ] ENTER
7  [ ] [ ] P [ ] R [ ] I [ ] N [ ] T [ ] SHIFT [ ] " [ ] C [ ] SHIFT [ ] " [ ] ENTER
8  [ ] [ ] C [ ] U [ ] R [ ] S [ ] O [ ] R [ ] ( [ ] ( [ ] X [ ] SHIFT [ ] ^ [ ] 2 [ ] ) [ ]
   [ ] / [ ] 4 [ ] ) [ ] - [ ] 4 [ ] ENTER
9  [ ] [ ] P [ ] R [ ] I [ ] N [ ] T [ ] SHIFT [ ] " [ ] D [ ] SHIFT [ ] " [ ] ENTER
10 [ ] [ ] [ ] E [ ] N [ ] D [ ] ENTER
    
```

This program will cause the letters A, B, C, and D to appear in positions 0, 4, 8, and 12, respectively on the display:

```

A   B   C   D   RUN
    
```

NOTE: Specifying a cursor position greater than 25 or less than 0 will result in an ERROR 19.

Demonstration Program

```

10 WAIT 0
20 DIM AS (0) * 13
30 INPUT "ENTER YOUR NAME", AS (0)
40 C = 0 : CLS
50 FOR I = 1 TO (LEN AS (0) - 1)
60 CURSOR C
70 PRINT MIDS (AS (0), I, 1)
80 C = C + 2
90 NEXT I
100 WAIT
110 CURSOR C
120 PRINT RIGHTS (AS (0), 1)
130 END
    
```

Keystrokes:

```

1 0 W A I T 0 ENTER
2 0 D I M A SHIFT S ( 0 ) * 1 3 ENTER
3 0 I N P U T SHIFT " E N T E R SPACE
  Y O U R SPACE N A M E SHIFT *
  SHIFT , A SHIFT S ( 0 ) ENTER
4 0 C = 0 SHIFT : C L S ENTER
5 0 F O R I = 1 T O ( L E N A SHIFT S ( 0 )
  - 1 I ENTER
6 0 C U R S O R C ENTER
7 0 P R I N T M I D S ( A S ( 0 )
  SHIFT , I SHIFT , 1 ) ENTER
8 0 C = C + 2 ENTER
9 0 N E X T I ENTER
1 0 0 W A I T ENTER
1 1 0 C U R S O R C ENTER
1 2 0 P R I N T R I G H T SHIFT S
  ( A SHIFT S ( 0 ) SHIFT , 1 )
  ENTER
1 3 0 E N D ENTER
    
```

F.3. CLS (Clear Screen)

The CLS statement erases the display by turning off all of the dots on the window. It is used before other statements to remove any residual data from the display. The cursor is repositioned to the left of the screen by the CLS command. The format is simply CLS.

Demonstration Program

```
10 GPRINT "7F7F7F7F7F"
20 CLS
30 PRINT "NEW INFO"
```

Keystrokes:

```
1 0 G P R I N T SHIFT " 7 F 7 F 7 F
7 F 7 F SHIFT " ENTER
2 0 C L S ENTER
3 0 P R I N T SHIFT * N E W SPACE
I N F O SHIFT " ENTER
```

F.4. GCURSOR

The GCURSOR statement specifies one of the 156 columns of dots, available on the display, as the beginning column for any subsequent display of information. The format of the GCURSOR instruction is:

GCURSOR position-expression

where:

position-expression evaluates to a number in the range 0 to 155. This number specifies one of the 156 7-dot columns of the display.

NOTE: If a position-expression results in a number which is less than 0 or greater than 155, an ERROR 19 will occur.

The GCURSOR statement is usually used, in conjunction with the GPRINT statement, for the purpose of creating graphic displays (this will be illustrated more thoroughly in the next section). Other types of instructions may follow the GCURSOR statement since this statement does not write any information. GCURSOR merely indicates in which column subsequent information will be written. This is illustrated by the following lines:

Program Listing:

```
10 GCURSOR 50
20 PRINT "A"
30 GCURSOR 80
40 PRINT 26/3
```


Keystrokes:

```

1  Ø  G  C  U  R  S  O  R  5  Ø  ENTER
2  Ø  P  R  I  N  T  SHIFT "  A  SHIFT "  ENTER
3  Ø  G  C  U  R  S  O  R  8  Ø  ENTER
4  Ø  P  R  I  N  T  2  6  /  3  ENTER
    
```

which produces normal looking output at positions 50 and 80:

```

                                RUN
                                .
                                A      8. 6666666667
    
```

What happens if we alter line 30 to begin printing at GCURSOR position 93? Try this by substituting the following line:

```
30 GCURSOR 93
```

Surprise! The second output has been "truncated" (chopped off) because it ran off the end of the display.

As a final example of the GCURSOR statement, we present an advanced program which creates an unusual effect by overlapping characters:

Program Listing:

```

10 WAIT 0
20 DIM AS (0) * 10
30 AS (0) = " " : C = 0
40 INPUT "ENTER MESSAGE (< 10 CHARS)", AS (0)
50 CLS
60 FOR I = 1 TO (LEN AS (0))
70 GCURSOR C
80 FOR J = 1 TO 3
90 GCURSOR C
100 PRINT MID$ (AS (0), I, 1)
105 C = C + 3
110 NEXT J
120 C = C + 5
130 NEXT I
140 WAIT
150 GCURSOR 155
160 GPRINT "00"
170 END
    
```

Keystrokes:

- 1 0 W A I T 0 ENTER
- 2 0 D I M A SHIFT \$ (0) * 1 0 ENTER
- 3 0 A SHIFT \$ (0) = SHIFT " SHIFT "
- SHIFT : C = 0 ENTER
- 4 0 I N P U T SHIFT " E N T E R SPACE
- M E S S A G E SPACE I SHIFT < 1 0
- SPACE C H A R S) SHIFT " SHIFT ,
- A SHIFT S (0) ENTER
- 5 0 C L S ENTER
- 6 0 F O R I = 1 T O I L E N A SHIFT \$
- I 0)) ENTER
- 7 0 G C U R S O R C ENTER
- 8 0 F O R J = 1 T O 3 ENTER
- 9 0 G C U R S O R C ENTER
- 1 0 0 P R I N T M I D SHIFT \$ I A
- SHIFT \$ (0) SHIFT , I SHIFT ,
- 1) ENTER
- 1 0 5 C = C + 3 ENTER
- 1 1 0 N E X T J ENTER
- 1 2 0 C = C + 5 ENTER
- 1 3 0 N E X T I ENTER
- 1 4 0 W A I T ENTER
- 1 5 0 G C U R S O R 1 5 5 ENTER
- 1 6 0 G P R I N T SHIFT " 0 0 SHIFT " ENTER
- 1 7 0 E N D ENTER

F.5. GPRINT

The GPRINT statement provides direct, programmable control over the dots of the display window. Since the GPRINT statement sets and resets dots within any 7-dot column, it is normally used in conjunction with the GCURSOR instruction. The GCURSOR statement selects the appropriate column for modification and the GPRINT statement manipulates the dots within that column. The GPRINT statement is also capable of printing several contiguous columns of information in a single statement.

In order to understand the format of the GPRINT statement, it is necessary to understand how the dots within a column are controlled. The pattern of energized dots within a column may be specified either as a decimal number or as a hexadecimal character string. If the decimal system is used, then each row may be visualized as being numbered, from the top down, by a power of two. This is illustrated below:

```

1  -----
2  -----
4  -----
8  -----
16 -----
32 -----
64 -----

```

With 7 dots to a column, each of which may either be on or off, there are 128 possible dot patterns. Thus, to specify a particular pattern one uses the format:

```
GPRINT pattern-expression ; pattern-expression 2 .. (etc) ..
```

where:

pattern-expression evaluates to a number in the range 0 to 127 and specifies the pattern of energized dots. Several pattern expressions may optionally be specified and must be separated by either a semicolon or a comma. If a comma is used, a blank column will be left between every printed column.

Let us illustrate the utility of the GPRINT instruction by creating a new character; an Up Arrow. First, we design our character on a grid representing the rows and columns:

```

1  ----- * -----
2  -- * - * - * --
4  * --- * --- *
8  ----- * -----
16 ----- * -----
32 ----- * -----
64 ----- * -----
      1   2   3   4   5

```

Caution

Observe the following when correcting the PRINT command in a program for a GPRINT command:

Example:

Rewrite "PRINT" entirely into "GPRINT".

```
20 PRINT AS
```

Inserting only "G" does not allow the computer to judge it as "GPRINT".
(This is regarded as "G" and "PRINT".)

The same can apply when the CURSOR command is corrected to a GCURSOR command, or to the printer commands.

Because our character is 5 columns wide we will need 5 separate numbers in the pattern-expression list of the GPRINT statement. The numbers representing the columns 1 and 5 must each specify a single dot in the row labeled 4. Similarly, the numbers representing columns 2 and 4 must each specify a single dot in row 2. The final statement is:

```
GPRINT 4;2;127;2;4
```

The specification of the third column (127) is the only number whose derivation may not be immediately obvious. The number 127 is the sum of a dot in the first row (1), a dot in the second row (2), a dot in the third row (4), and so on. Thus, 127 is $1 + 2 + 4 + 8 + 16 + 32 + 63$ and specifies all 7 dots in the column. Any pattern may be created by specifying a row or a sum of several rows.

If the hexadecimal addressing scheme is used, the 7 rows of the display are conceptually divided into a lower group of 3 rows and an upper group of 4 rows. Each group is numbered, from its top row, by powers of two as illustrated below:

```
1  -----
2  -----
4  -----
8  -----
1  -----
2  -----
4  -----
```

Thus, it is possible to represent all the patterns of a group by a single hexadecimal digit. Because the lower group has only 3 rows, the range of allowable digits for this group will be from 0 to 7. Of the two hexadecimal digits required to specify an entire column, the first digit will represent the lower group and the second digit will represent the upper group.

The form of the hexadecimal GPRINT is:

```
GPRINT "hexadecimal-string"
```

where:

hexadecimal-string is a string consisting of hex digits, each pair specifying the dot pattern of a single column.

Using this format to create our Up Arrow character from the previous example would give us the statement:

```
GPRINT "04027F0204"
```

Table of Hex Characters

1	-----	---*---	-----	---*---
2	-----	-----	---*---	---*---
4	-----	-----	-----	-----
8	-----	-----	-----	-----
	0	1	2	3

1	-----	---*---	-----	---*---
2	-----	-----	---*---	---*---
4	---*---	---*---	---*---	---*---
8	-----	-----	-----	-----
	4	5	6	7

1	-----	---*---	-----	---*---
2	-----	-----	---*---	---*---
4	-----	-----	-----	-----
8	---*---	---*---	---*---	---*---
	8	9	A	B

1	-----	---*---	-----	---*---
2	-----	-----	---*---	---*---
4	---*---	---*---	---*---	---*---
8	---*---	---*---	---*---	---*---
	C	D	E	F

Demonstration Program

This program prints all possible dot patterns, in order, from 0 to 127:

```

Program Listing:
10 WAIT 0 : CLS
20 FOR I = 0 TO 127
30 GCURSOR I
40 GPRINT I
50 NEXT I
60 WAIT
70 GCURSOR 155 : GPRINT "00"
80 END
    
```

Keystrokes:

```

1 0 W A I T 0 SHIFT : C L S ENTER
2 0 F O R I = 0 T O 1 2 7 ENTER
3 0 G C U R S O R I ENTER
4 0 G P R I N T I ENTER
5 0 N E X T I ENTER
6 0 W A I T ENTER
7 0 G C U R S O R 1 5 5 SHIFT :
  G P R I N T SHIFT " 0 0 SHIFT " ENTER
8 0 E N D ENTER
    
```


F.6. POINT

The POINT function returns a number which represents the pattern of activated dots within the given column. Thus, the POINT function allows the "sensing" of any column on the display under program control.

The format of the POINT function is:

POINT position-expression

where:

position-expression evaluates to a number in the range 0 to 155 and represents the column to be investigated.

The value returned by the POINT function is a number in the range 0 to 127. The interpretation of this number is a sum of powers of 2 as explained in the section on GPRINT.

As an illustration assume that on the display is a capital I in columns 40 through 44:

```

1  -- * - * - * --
2  ---- * -----
4  ---- * -----
8  ---- * -----
16 ---- * -----
32 ---- * -----
64 -- * - * - * --
   4  4  4  4  4
   0  1  2  3  4
    
```

The expression:

POINT 40 would return 0,
 POINT 41 would return 65,
 POINT 42 would return 127.

Demonstration Program

The program listed here fills the display within the character stored in A\$ and creates unusual patterns by reversing this character. The program utilizes several of the statements discussed in this chapter.

Program Listing:

```

10 A$ = "X"
20 WAIT 0
30 Y = 5 : X = 155/Y : C = 0
40 FOR I = 1 TO X
50 GCURSOR C
60 PRINT A$
70 C = C + Y
80 NEXT I
90 FOR I = 0 TO 155
100 GCURSOR I
110 A = 127 - POINT I
120 GPRINT A
130 NEXT I
140 GOTO 90

```

Keystrokes:

```

1 0 A SHIFT $ = SHIFT ^ X SHIFT " ENTER
2 0 W A I T 0 ENTER
3 0 Y = 5 SHIFT : X = 1 5 5 / Y
  SHIFT : C = 0 ENTER
4 0 F O R I = 1 T O X ENTER
5 0 G C U R S O R C ENTER
6 0 P R I N T A SHIFT $ ENTER
7 0 C = C + Y ENTER
8 0 N E X T I ENTER
9 0 F O R I = 0 T O 1 5 5 ENTER
1 0 0 G C U R S O R I ENTER
1 1 0 A = 1 2 7 - P O I N T I ENTER
1 2 0 G P R I N T A ENTER
1 3 0 N E X T I ENTER
1 4 0 G O T O 9 0 ENTER

```

NOTE: If the last line is included, the program will be in an infinite loop when run. (This may be stopped with the BREAK key). The character displayed may be changed by inserting a new character in the assignment in line 10.

G. DEBUGGING

No matter how careful you are, eventually you will create a program which does not do quite what you expect it to. In order to isolate the problem, SHARP's designers have provided a special method of executing programs known as the "Trace" mode. In the Trace mode, the PC-1500 will display the line-number of each program line and will halt after the execution of that line. This allows you to follow (or trace) the sequence of instructions as they are actually performed. When the program pauses after the execution of a line, you may inspect or alter the values of variables.

The form of the instruction for initiating the Trace mode is simply: TRON. The TRON instruction may be issued as a command (in RUN mode) or it may be embedded, as a statement, within a program. Used as a command, TRON informs SHARP that tracing is required during the execution of all subsequent programs. The programs to be traced are then started in a normal manner, with a GOTO or RUN command.

If TRON is used as a statement, it will initiate the Trace mode only when the line containing it is executed. If, for some reason, that line is never reached, the Trace mode will remain inactive.

Once initiated, the Trace mode of operation remains in effect until canceled by a TROFF instruction. The TROFF instruction may also be issued as either a command or a statement. The Trace mode can also be canceled by the key sequence:

[SHEFT] [CL]

As an example of using the Trace mode, enter the following program to compute the length of the hypotenuse of a triangle given the length of the sides:

Program Listing:

```
10 INPUT A, B
20 A = A * A : B = B * B
30 H = √(A + B)
40 PRINT "HYPOTENUSE = "; H
```

In RUN mode, issue the TRON command, followed by the RUN command. Notice that the INPUT command operates in the usual manner by displaying a question mark for each input value required. As soon as you have entered two values, the line number of the INPUT statement appears:

```

                                     RUN
10 :
```

By pressing the **[↑]** (Up Arrow) key and holding it, you may review the entire line:

```

                                     RUN
10 : INPUT A, B
```

To continue the program, press the **[↓]** (Down Arrow) key once. This causes the next line to be executed and its line number to be displayed. Again, you may review the line with the (Up Arrow) key. You may also check the contents of any variable by typing its name and pressing **[ENTER]** :

(where A is a program variable)

```

                                RUN                                1
                                                                •
                                                                4
    
```

It is necessary to press the (Down Arrow) key once for each line to be executed until the program ends. If you do not wish to continue normal line-by-line execution, press the ENTER key to suspend execution of the program. If you change your mind again, suspended programs may be continued with the CONT command.

A sample session, using our hypotenuse program, follows:

Keystrokes	Display
	>
<input type="text" value="T"/> <input type="text" value="R"/> <input type="text" value="O"/> <input type="text" value="N"/>	TRON_
<input type="text" value="ENTER"/>	>
<input type="text" value="R"/> <input type="text" value="U"/> <input type="text" value="N"/>	RUN_
<input type="text" value="ENTER"/>	?
<input type="text" value="3"/>	3_
<input type="text" value="ENTER"/>	?
<input type="text" value="4"/>	4_
<input type="text" value="ENTER"/>	10:
<input type="text" value="↑"/>	10: INPUT A, B_
<input type="text" value="↓"/>	20:
<input type="text" value="↑"/>	20: A=A*A: B=B*B_
<input type="text" value="A"/>	A_
<input type="text" value="ENTER"/>	9
<input type="text" value="B"/>	B_
<input type="text" value="ENTER"/>	16
<input type="text" value="↓"/>	30:
<input type="text" value="H"/>	H_
<input type="text" value="ENTER"/>	5
<input type="text" value="↓"/>	HYPOTENUSE= 5
<input type="text" value="↑"/>	40: PRINT "HYPOTENUSE=" ; H_
<input type="text" value="↓"/>	40:
<input type="text" value="↓"/>	>

H. HEXADECIMAL Numbers and Boolean Functions

H.1. Hexadecimal Numbers

The PC-1500 provides the capability to use a hexadecimal (base 16) number within any expression in which a decimal number may be used. Hexadecimal numbers are distinguished from decimal numbers by preceding them with an & (ampersand). The following are examples of valid hexadecimal numbers:

&16 &F &7ECA &08 &99A &-5B

Hexadecimal numbers may be used in calculations:

10 + &A

RUN

20

Or within programs:

Program:

```
35 GPRINT &F, 54, &3E
40 DATA 67, &7F, &2B, 12, 305
```

H.2. AND Function

The AND function provides a boolean AND of the internal representation of two values. The values must be in the range -32768 through 32767. Numbers which exceed this range will cause an ERROR 19.

<u>Example:</u>	<u>Result:</u>
10 AND &F	10
1 AND 0	0
-1 AND 1	1
55 AND 64	0
16 AND 63	16

H.3. OR Function

The OR function performs a boolean OR on the internal representations of two values. The values must be in the range -32768 through 32767 . Numbers which exceed this range will cause an ERROR 19.

<u>Example:</u>	<u>Result:</u>
10 OR &F	15
1 OR 0	1
-1 OR 1	-1
55 OR 64	119
16 OR 63	63

H.4. NOT Function

The NOT function returns the boolean NOT, or complement, of the internal representation of a single value. The value must be in the range -32768 through 32767 . If the value exceeds this range an ERROR 19 will occur.

<u>Examples:</u>	<u>Result:</u>
NOT 0	-1
NOT &F	-16
NOT 55	-56
NOT 1	-2
NOT -2	1

I. HALTING PROGRAM EXECUTION

STOP, CONT

The STOP statement causes the computer to suspend the execution of a program. When the program stops, the values of all variables are retained and the programmer may inspect and change these. The program may then be continued, at the point where it was halted, with the CONT command.

When the STOP statement is encountered by the computer a message similar to the following is displayed:

```

                                RUN
BREAK IN 60
  
```

where 60 is the number of the line which contains the STOP statement.

If you wish to review this line, depress and hold the (Up Arrow) key.

When the BREAK message appears you may also review and change the values of variables. For example:

```

HQ 
56.23
AS 
"DEDUCTIONS"
  
```

Whenever you are ready to resume execution, simply return to the prompt (>) and type CONT ENTER.

J. MODE CONTROL

LOCK, UNLOCK

The LOCK instruction may be used to control the mode (RUN, PROGRAM, or RESERVE) in which the computer operates. Included within a program it prevents the user from accidentally changing the mode and injuring the program. The LOCK instruction disables the MODE key, "locking" the computer into whatever mode it is currently in.

To re-enable the MODE key, the UNLOCK instruction is used. UNLOCK restores the normal functioning, allowing changes in mode.

Either instruction may be used as a command or a statement. The forms are simply:

LOCK

UNLOCK

VI. EXPANDING THE PC-1500

A. THE PRINTER/CASSETTE INTERFACE (CE-150)

The Printer/Cassette Interface is an option for the SHARP PC-1500 pocket computer. This unit can be connected to one or two cassette tape recorders. The tape recorders can be used to store programs and data on standard audio cassettes. Programs can be "loaded" back into the PC-1500 for use at a later date, saving you the trouble of typing them again.

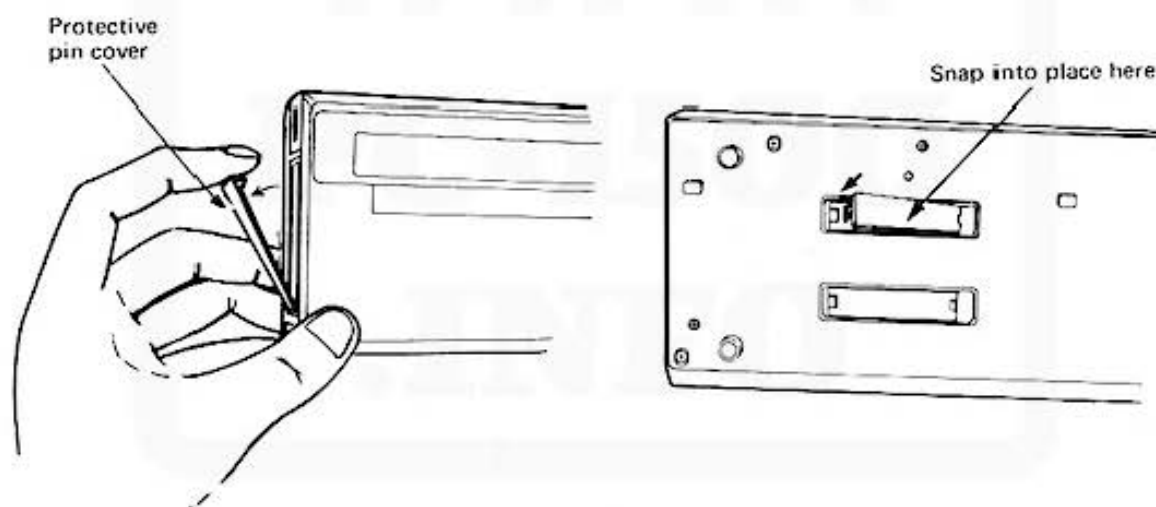
1. Connecting the Computer to the Interface

Connect the printer/cassette interface (CE-150) and the computer (PC-1500) in the following manner.

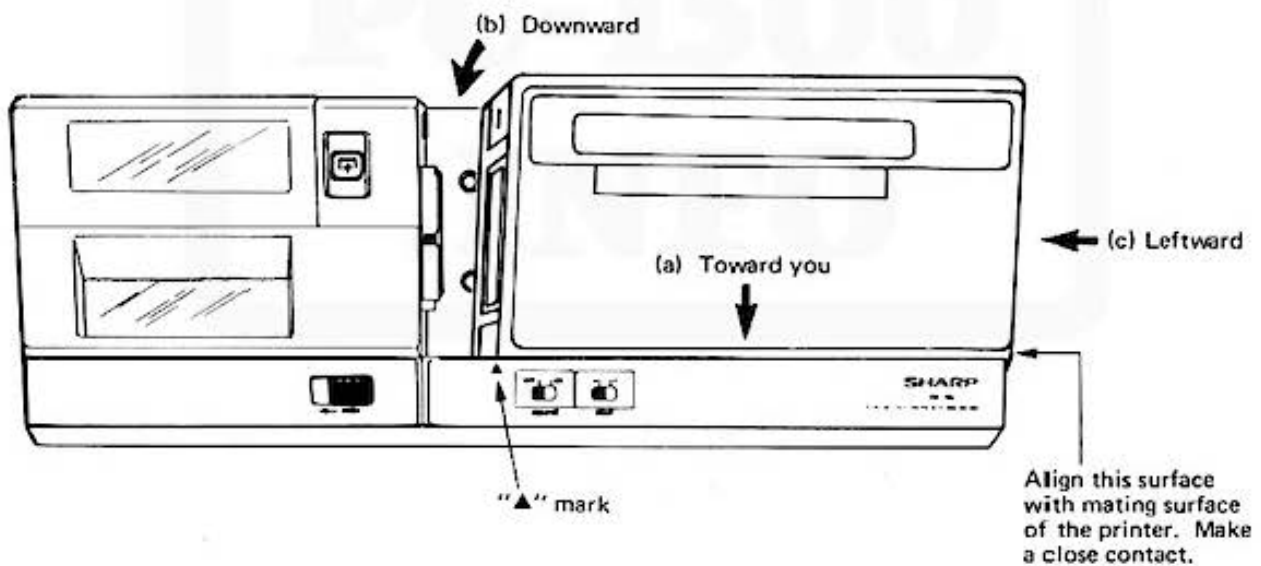
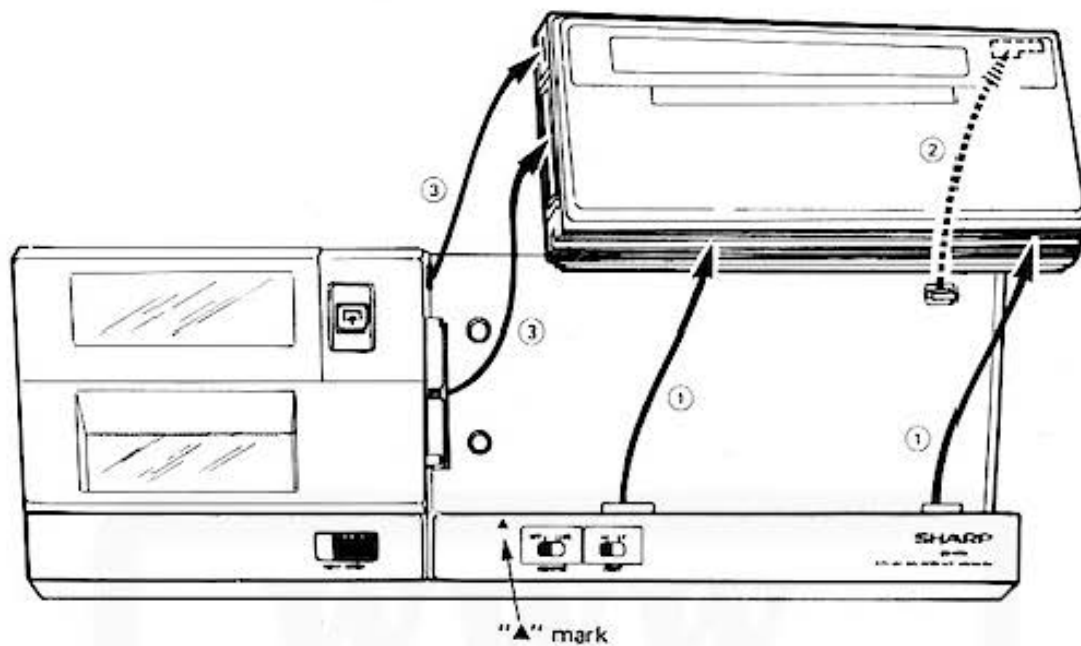
- (1) Turn the computer power OFF.

Important Note! It is essential that computer power be OFF. If power is ON, the computer may "hang up" (all keys inoperative). If this occurs, press the ALL RESET switch on the bottom of the computer while pressing the **ON** key.

- (2) Remove the protective pin cover from the left side of the computer and snap it into place on the bottom of the printer (see figure).



- (3) Place the lower edge of the computer into the "cradle" so that the printer guides match-up with the computer guide-slots.
- (4) Lay the computer down flat.
- (5) Gently slide the computer to the left so that the printer pins are inserted into the computer (see figure).



Do not force the computer and Printer together. If match-up does not easily take place, carefully shift the computer left and right to correctly position mating surfaces.

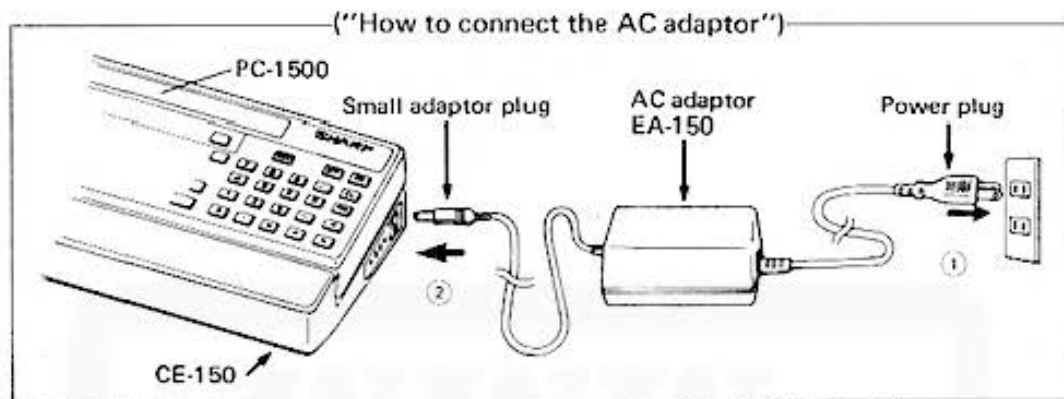
2. Power

The CE-150 unit utilizes a rechargeable Ni-CAD battery power source. Therefore it is necessary to recharge the batteries after unpacking, and when the message below is displayed. (In this case, the printer is locked. To unlock the printer, press the **OFF** and **ON** keys of the computer in that order after charging the battery)

(1) **ERROR 80** or **ERROR 78**

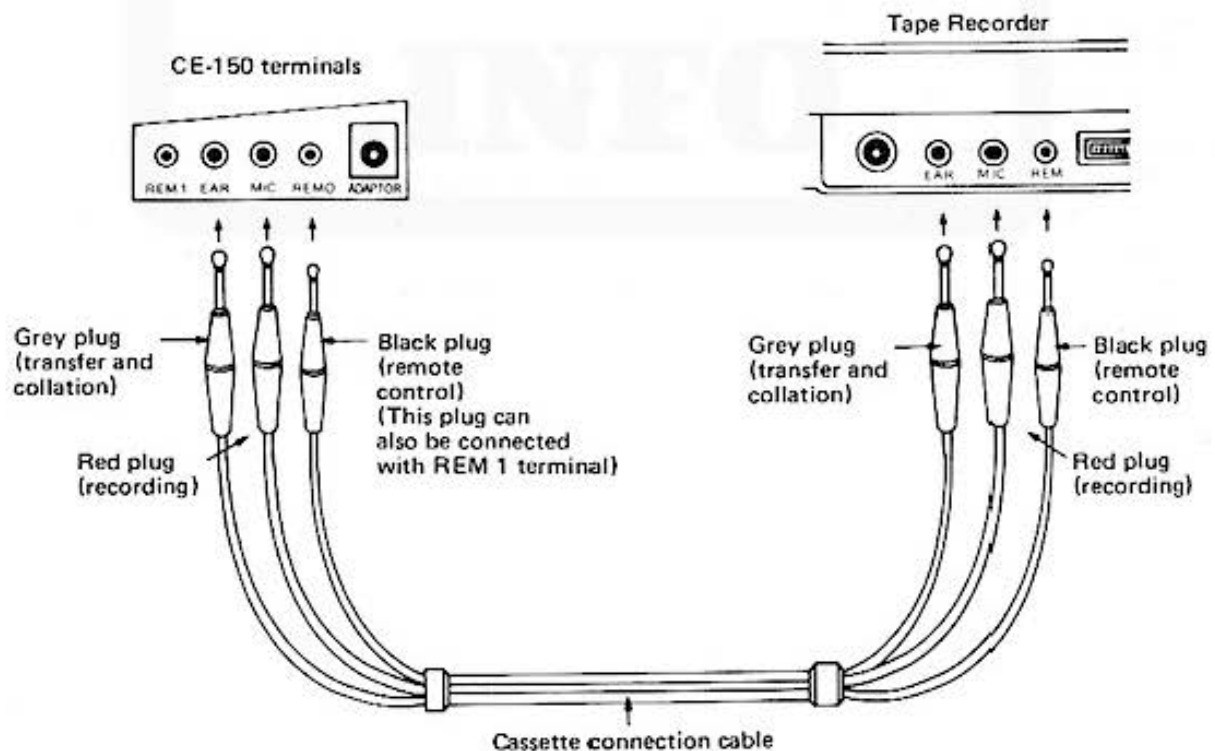
Note: When the printer is in the pen replacement state, the display of ERROR 78 may appear.

(2) **:CHECK 6** or **NEW 0? :CHECK 6**



3. Connecting a Tape Recorder to the Interface

First connect the CE-150 unit and computer, and connect a tape recorder with the CE-150 unit as shown in the following diagram.



The following is a description of the minimum tape recorder specifications necessary for interfacing with the CE-150:

Item	Requirements
1. Recorder Type	Any tape recorder, cassette, micro-cassette, or open reel recorders may be used in accordance with the requirements outlined below.
2. Input Jack	The recorder should have a mini-jack input labeled "MIC". Never use the "AUX" jack.
3. Input Impedance	The input jack should be a low impedance input (200 ~ 1,000 OHM.)
4. Minimum Input Level	Below 3 mV or -50 dB.
5. Output jack	Should be a minijack labeled "EXT. (EXTernal speaker)", "MONITOR", "EAR (EARphone)" or equivalent.
6. Output impedance	Should be below 10 OHM.
7. Output level	Should be above 1V (practical maximum output above 100 mW)
8. Distortion	Should be within 15% within a range of 2 KHz through 4 KHz.
9. Wow and Flutter	0.3% maximum (W.R.M.S)
10. Other	Recorder motor should not fluctuate speed.

- * In case the miniplug provided with the CE-150 is not compatible with the input/output jacks of your tape recorder special line conversion plugs are available on the market.

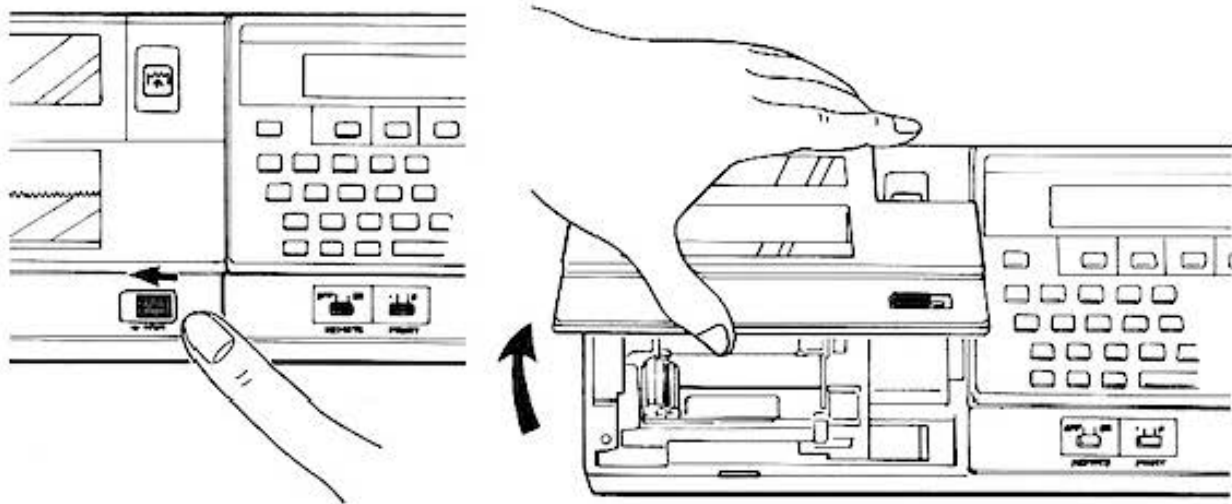
NOTE:

- Some tape recorders may reject connection due to different specifications. Or those tape recorders having distortion, increased noise, and power deterioration after long years of use may not show satisfactory results owing to change in their electrical characteristics.
- Precautionary Instructions for Tape Recorder Use
 - (1) For any transfer or collation, use the tape recorder that was used for recording. If the tape recorder for transfer or collation is different from that used for recording, no transfer or collation may be possible.
 - (2) The head of a tape recorder, if dirty, increases distortion or decreases the recording level. Therefore, keep the head clean.
 - (3) Use a tape that is free of extremely low frequency response, scratches and creases.

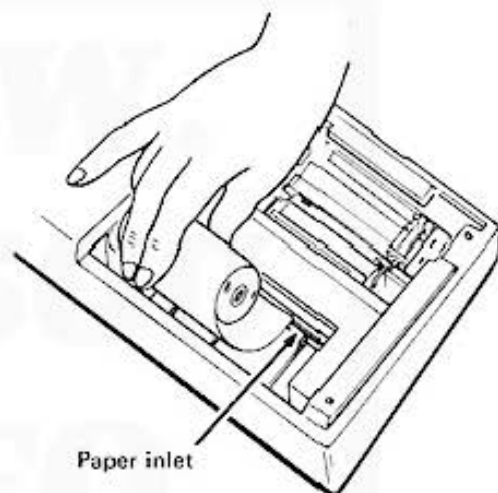
4. Loading the paper

For details refer to the instruction manual for CE-150.

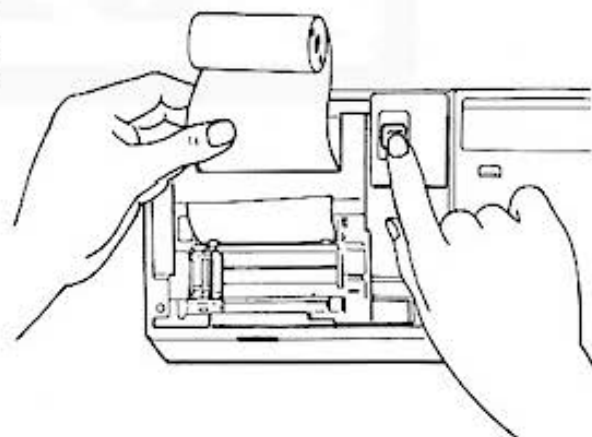
(1) To remove the printer cover, shift the printer cover lock lever in the direction of the arrow.



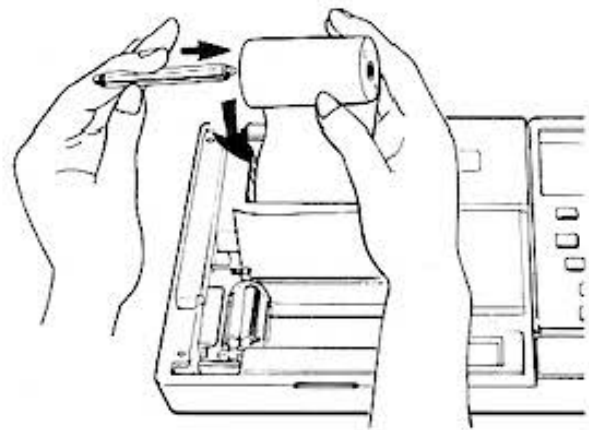
(2) Cut the tip of roll paper straight, and insert the paper correctly into the paper inlet. (Any curve or crease at the paper tip may prevent paper insertion.)



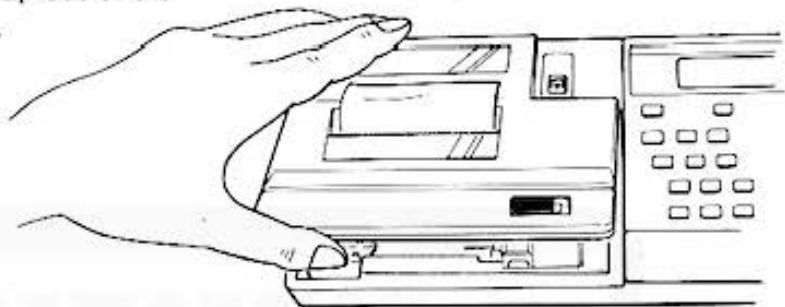
(3) Press the computer **ON** key to turn it on, and press the **F** key to feed paper. At this time, feed the paper so that the paper tip is 3 to 5 cm above the printer.



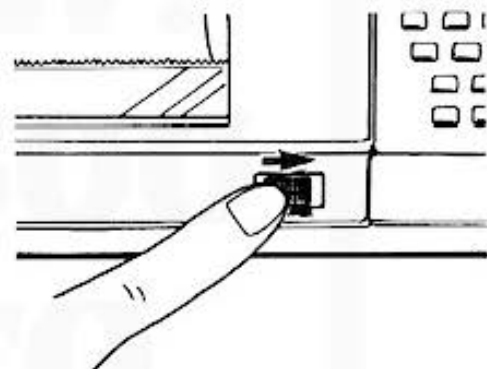
- (4) Insert the shaft into the paper roll and place the paper in the paper case.



- (5) Place the printer cover back in position. At this time, thread the paper roll tip out of the printer through the paper cutter.

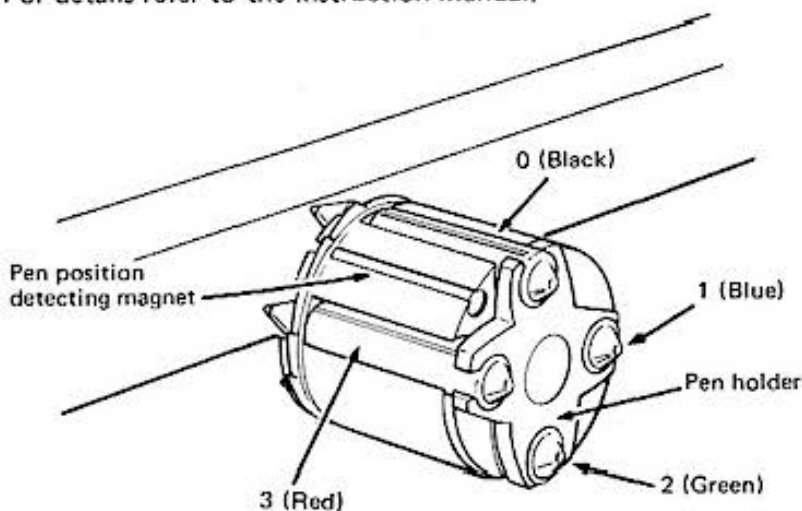


- (6) Lock the printer cover.



5. Replacing the Pens

Four kinds of pens can be installed in this unit.
Pen installation positions are as illustrated below:
For details refer to the instruction manual.



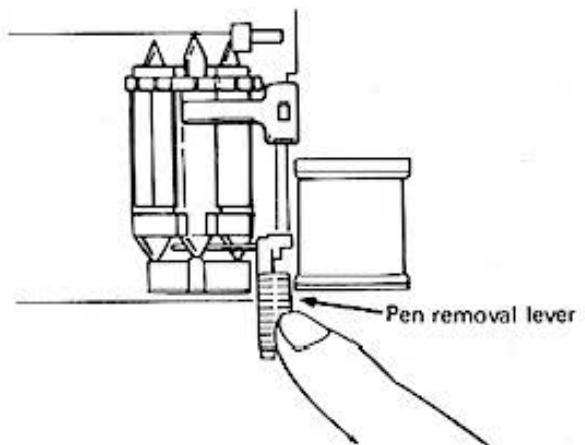
The pen slots are numbered 0, 1, 2 and 3, clockwise from the position-detecting magnet. These numbers correspond to the positions selected by the Color command.

For pen installation or replacement, follow the procedure below:

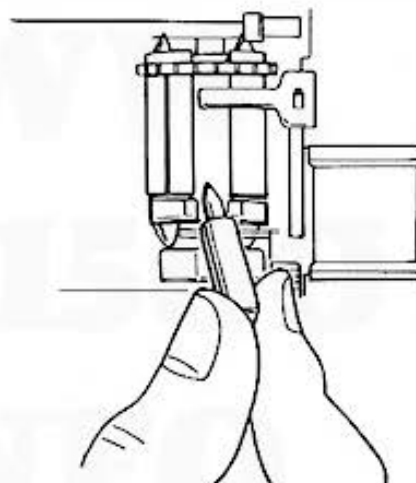
- (1) With the computer **[O]** key pressed, press the printer **[P]**. This allows the printer to be in its pen replacement state, when the pen holder shifts to the left and rotates. With the pen on top changed, the pen holder moves to the right. (Release the key when the pen holder starts moving.)

- (2) To remove the pen, press the pen removal lever. This causes the pen on top to come off.

(Note): When removing the pen, hold it lightly to prevent it from popping up into the printer.



- (3) Install a new pen.



- (4) To install or remove the next pen, press the **[P]** key. The pen holder returns to the left, rotates so that the next pen comes up, and shifts to the right again. Remove the pen and replace it with a new one as in steps (2) and (3).

- (5) After pen replacement or installation, press the printer **[P]** with the computer **[CL]** pushed down. This causes the printer to be released from its pen replacement state, and the pen to return to the left.

(Note): To use this unit, install the four pens on the pen holder. Operation with a lack of even one pen may cause color changes to malfunction.

Handling the pens:

The pens are installed on the printer when it is used, and removed from the printer after use. Cap the pens and place them in their refill for storage.

Leaving the pens installed on the printer for a long time or otherwise exposed may cause the ink to dry.

B. USING A CASSETTE RECORDER

THE COMMANDS FOR THE PRINTER AND TAPE RECORDER TO BE DESCRIBED HEREAFTER ARE ONLY AVAILABLE ON THE OPTIONAL PRINTER CE-150 (WITH A BUILT-IN CASSETTE INTERFACE). SINCE THE COMPUTER IS NOT EQUIPPED WITH THESE COMMANDS, PROGRAMMING WITH THEM IS POSSIBLE ONLY WHEN CONNECTED TO THE CE-150. THEREFORE, BE SURE TO CONNECT TO THE CE-150 FOR PROGRAMMING BY USING THESE COMMANDS.

1. Tape Recorder Operation

We recommend that you follow this procedure using a small, simple program. In the event of a problem, it will be easier to re-perform the operation. Enter the program now.

You must prepare the tape recorder for program and data transfer. The following steps are necessary to do this:

1. Turn the "remote" switch on the interface (CE-150) OFF.
2. Put a tape into the tape recorder (an important step).
3. Find a blank portion of tape. If the tape is brand new, advance it past the leader portion. If your recorder has a number counter, jot down the number. This is extremely helpful in relocating the program you are saving. O.K, next . . .
4. If your recorder has an automatic volume control, set it on automatic. If it is a manual volume control, turn the volume level up halfway between middle and maximum (i.e. 3/4 level). If your recorder has a tone control, turn the tone control knob to a position halfway between intermediate and high (3/4 position).
5. Turn the "remote" switch on the interface back ON. If your recorder does not have a "remote" feature, (this means that there is no place to connect one of the wires of the wiring harness,) use the "pause" key to temporarily halt recording. If your recorder does not have a "pause" switch, you are making things difficult. We strongly recommend that you get a recorder that has one. This will make your new programming life much easier. Better yet get one that has both, a remote feature, and a pause switch.
6. Depress both the RECORD and the PLAY keys simultaneously. If you are using a machine without a "pause" switch you will want to do this immediately before saving the program.

All ready? Ok, read on for the good stuff

2. The CSAVE command

Allright, countdown checklist

- | | | | | |
|---|------------|-------|----|-------|
| 1. Tape recorder ready and waiting? | yes | _____ | no | _____ |
| 2. Tape in? | yes | _____ | no | _____ |
| 3. Computer on? | yes | _____ | no | _____ |
| 4. Program on the computer? | yes | _____ | no | _____ |
| 5. Wires all hooked up right? | yes | _____ | no | _____ |
| 6. Did you jot down the counter number? | yes | _____ | no | _____ |
| 7. Are you dressed? | irrelevant | | | _____ |

If you answered "NO" to any of these questions reread the previous section to clear up the problem.

Ok, one more thing and you are ready to go:

With the same command which saves your program you must give the program a "filename". This is for reference purposes. Your filename can not be longer than 16 characters. To save the program with a filename type:

```
CSAVE [SHIFT] " PROG.-1 [SHIFT] "
```

Your program will be saved with the name "PROG.-1" You can assign any name you desire, whatever is easiest for you to keep track of. Also, note that there is a 16 character length limit for your filename. If the name is longer than 16 characters, the excess is ignored. A good practice is to maintain a program log, which includes the program name, starting and stopping location on tape (use the counter numbers), and a brief description of what the program does.

Press the **[ENTER]** key. At this time you should hear a shrill buzzing sound, and the tape should be turning. Also the "BUSY" indicator should light up. This tells you that the computer is "busy" transferring your program from memory to the tape. If this does not happen, start again from the beginning of the section. It took me only 10 tries to get it right, so don't be discouraged. If I can do it, you can do it.

Once the computer arrives at the end of the program, the "BUSY" indicator light will go off, the recorder will stop, and the "prompt" will re-appear on the display. Congratulations, your first program has been saved for future use. In order to insure that this has in fact been accomplished we can read it back into memory from the tape as explained in the next section.

3. The CLOAD command

Now that your first program is saved on tape, you will no doubt want to see if it is really there. To do this is relatively simple; use the CLOAD? command, of course.

"What does it do?" you ask. Well, after CSAVEing your program simply type CLOAD? The computer compares the CSAVEd program with the one in its memory. If all went well, it will calmly display your filename and end its check. If all did not go well, an error message will be displayed, usually ERROR 43. This tells you that the program on tape is somehow different from the program in SHARP's memory. Erase that portion of tape and start again. Check all your connections, and try turning the volume and tone up a little.

The following are instructions for loading a program back in:

1. Turn the remote switch on the interface OFF.
2. Rewind the tape to the place at which you started, again using the number counter. (See how handy the number counter is!).
3. Stop rewinding.
4. Turn the remote switch back ON.
5. Press the PLAY button.
6. Type:
`CLOAD` **[SHIFT]** "PROG.-1" **[SHIFT]** "
and press the **[ENTER]** key.
(Remember "PROG.-1" is the filename we have given to your program. If you saved the program under another name you must use that name instead of "PROG.-1")
7. The "BUSY" indicator will now light up, and the program will be brought back into the computer's memory for use.
8. Type RUN, and the program you previously CSAVEd will re-appear. The cassette retains a copy of the program, so you can CLOAD the same program over and over again! While loading, when an error message ERROR 43 or ERROR 44 is displayed, start again from the above step (1).

4. PRINT # and INPUT # Commands

PRINT # :

Now, that we have illustrated the use of CSAVE and CLOAD, we would like to introduce two similar commands. The PRINT # command saves the value of a variable or set of variables on tape. This is different from CSAVE which saves a program. The purpose for saving data is to enable you to use the same data in another program. For example, in the following program, the variable T\$ is in use:

```
10: PRINT "WHAT IS YOUR NAME?"  
20: INPUT T$  
30: PRINT T$
```

If you want to save the value of T\$ for use in another program, you must issue the PRINT # command to save it on tape. You can do this in two ways:

1. Manually
2. Through a program

Note: This operation requires the use of the tape recorder, so prepare the recorder to receive data. If you are not sure how, go back to the previous section.

1. The Manual Method

The manual method offers several options:

OPTION 1:

After running a program, switch to the RUN mode and type:

```
[P] [R] [I] [N] [T] [SHIFT] [#] A, B, C (then press [ENTER] )
```

The tape recorder will now spring into action and will save the value of all variables on tape.

OPTION 2:

If you want to identify only certain variables to be saved type the following:

P R I N T **SHIFT** **#** "filename" ; A, B, C

In this instance you have just specified variables A, B, and C as the ones to be saved on tape under the given filename.

OPTION 3:

You can also specify all values of related variables by typing:

P R I N T **SHIFT** **#** "filename" ; B (*)

The * symbol will save all varieties of "B", including B (1) and B itself.

2. Through a Program

To do this, simply assign a line number to the PRINT# command in your program. You may use any of the formats described above. When SHARP encounters this line number it will automatically activate the tape recorder and begin to transfer the data to tape. Once again, PLEASE experiment. If things are not working, go back to the previous sections. Something very simple may have been overlooked.

INPUT# :

This command allows the same formats as PRINT#. The difference is that INPUT# transfers data from tape to the computer. (PRINT# transfers data from the computer to tape). You can use the INPUT# command manually or as part of a program. Remember to prepare the tape recorder before you begin to use this command.

NOTE: If you are specifying more variables (as part of your INPUT# command line) than exist on tape the remaining variables will be assigned the value 0. If you are specifying fewer variables (as part of your INPUT# command line) than exist on tape the remaining variable will be ignored.

5. Editing Programs (MERGE)

The MERGE command allows you to store two programs in the computer memory at the same time. For example, let's assume the computer memory contains the following program:

```
10: PRINT "DEPRECIATION ALLOWANCE"  
20: PRINT "ENTER METHOD?"  
30: INPUT A
```

At this point you suddenly remember that you may have a similar program portion on tape under the filename "DEP1". You will, of course, want to see if this program has sections useful in the program you are currently constructing. The first step is to find the tape with DEP1 on it. Cue the tape to the place at which "DEP1" starts.

Now type:

M E R G E **SHIFT** ***** **D E P 1** **SHIFT** **"** then press **ENTER**

The computer will now load "DEP1" into memory in addition to the program above. After "DEP1" is loaded, you might find something, in memory, like this:

```
10: PRINT "DEPRECIATION ALLOWANCE"  
20: PRINT "ENTER YOUR METHOD"  
30: INPUT A  
10: PRINT "INTEREST CHARGES"  
20: PRINT "AMOUNT BORROWED?"  
30: INPUT B  
:  
(etc)
```

Notice that, unlike the CLOAD command, the new program did NOT replace the existing one. Another thing to notice is that you now have duplicate line numbers.

At this point you realize that the part of "DEP1" which begins with line number 40 is useful to the "Depreciation" program you are writing. Simply delete the first three lines of the new program (starting with 10: PRINT "INTEREST CHARGES") and then edit the rest of the new program to suit your needs. PRESTO, you have saved yourself quite a bit of typing, by using a common program portion.

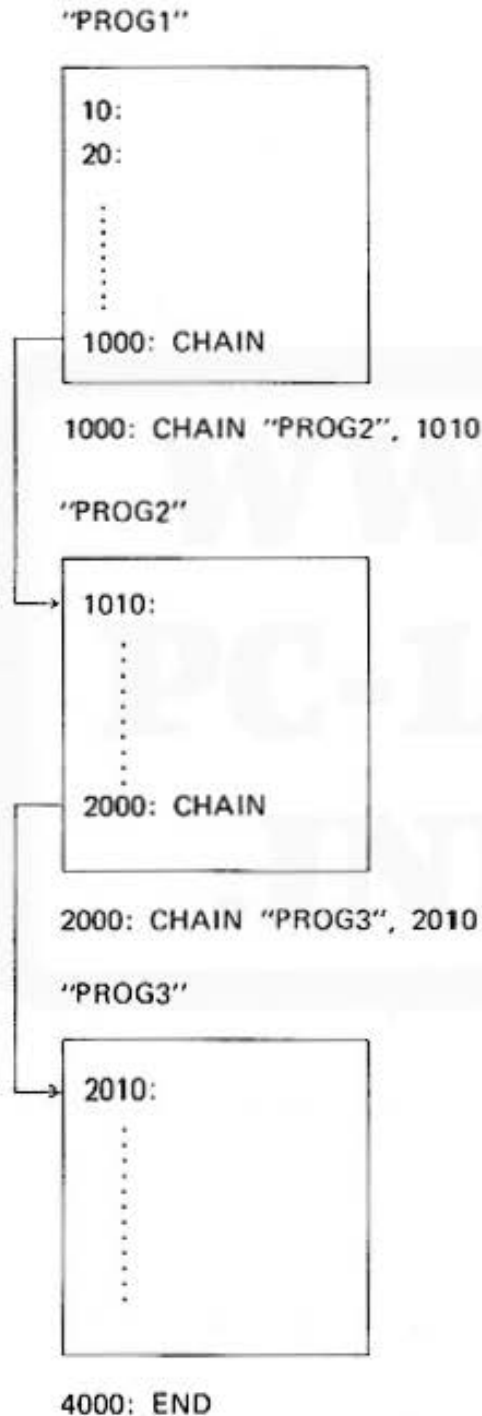
Again feel free to experiment. Try merging other program sections.

WWW.
PC-1500
.INFO

6. The CHAIN Statement

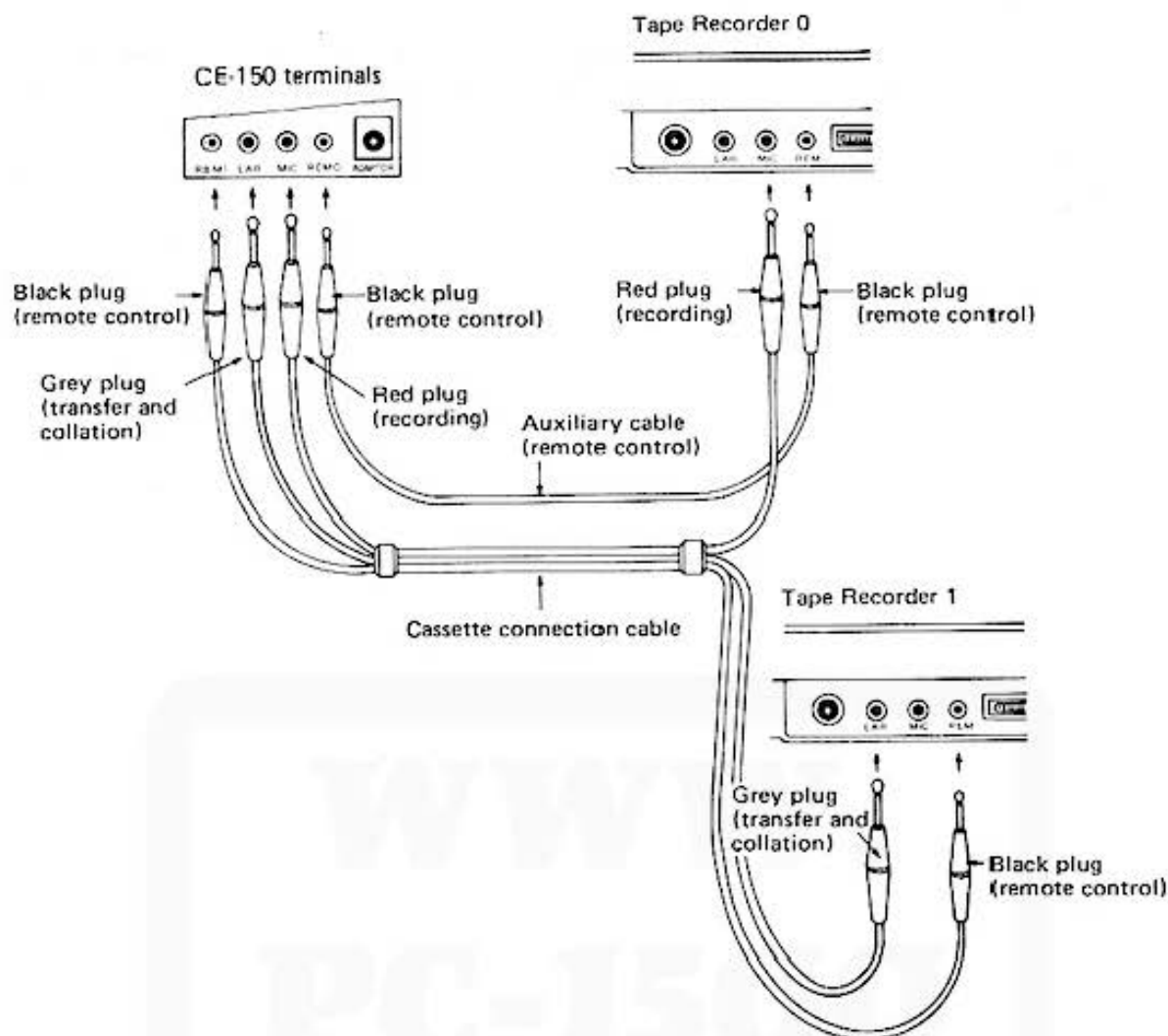
CHAIN is a program instruction; it can only be used within a program. It cannot be used manually like CSAVE, CLOAD, and MERGE. The CHAIN statement allows you to run a program that is too large to fit into memory all at once. Such large programs must be divided up into sections with a CHAIN statement at the end of each section. These sections can be saved on tape with CSAVE.

For example, let's assume you have three program sections named PROG1, PROG2, PROG3. Each of these sections ends with a CHAIN statement.



During execution, when the computer encounters the CHAIN statement, the next section is called into memory and executed. In this manner, all of the sections are eventually run.

7. Using Two Tape Recorders



When using two tape recorders, one of them is used for recording and the other for playback. As illustrated, the Printer/Cassette Interface and the two tape recorders are connected by using the connection cords and the auxiliary cord for remote control.

- The CE-150 is equipped with two remote control terminals REM 0 and REM 1, either of which can be used. In program operation (not manually), however, the program designates the tape recorder connected to the REM 0 terminal or the REM 1 terminal. Therefore, these remote control terminals should be connected according to the program.

In this section, how to operate the second tape recorder connected to the REM 1 terminal of the interface is explained

1. Saving

Procedures:

- (1) Type RMT OFF and press **[ENTER]** to reset the second remote control function. (Control of Tape recorder 1 in the illustration above)
- (2) Put a tape into the tape recorder.
- (3) Type RMT ON and press **[ENTER]** to set the second remote control function.
- (4) Set the volume and tone controls in the same manner as that explained previously in the single tape recorder.
- (5) Depress both the RECORD and the PLAY buttons simultaneously.

- (6) Execute RECORD instruction.

Program: CSAVE-1 "file name" **ENTER**

Data: PRINT # -1, "file name"; variable, variable,

(Example) Designate PRO or RUN mode.

CSAVE-1" PR-1" **ENTER**

After the saving the "prompt" will re-appear on the display and tape stops. Rewind the tape for collation.

2. Collating the Computer and Tape Contents

Procedures:

- (1) Type RMT OFF **ENTER** to clear remote control functions.
- (2) Rewind the tape to the place at which you started, again using the number counter.
- (3) Enter RMT ON and the **ENTER** key to set remote control functions.
- (4) Set the volume and tone controls in the same manner as that explained previously in the single tape recorder.
- (5) Press Playback button.
- (6) Execute COLLATION instruction.

CLOAD?-1" file name" **ENTER**

(Example) Designate PRO or RUN mode. CLOAD?-1 "PR-1" **ENTER**

Execution ends when both contents match, resulting in prompt displays.

3. Transfer from Tape

Procedures:

- (1) Enter RMT OFF and the **ENTER** key to clear remote control functions.
- (2) Put a recorded tape into the tape recorder.
- (3) Type RMT ON and press the **ENTER** to set remote control functions.
- (4) Set the volume and tone controls in the same manner as that explained previously in the single tape recorder.
- (5) Press the Playback button.
- (6) Execute TRANSFER instruction.

Program: CLOAD-1 "file name" **ENTER**

Data: INPUT # -1, "file name"; variable, variable, **ENTER**

(Example) Designate PRO or RUN mode.

CLOAD-1 "PR-1" **ENTER**

After the transfer, prompt displays result.

C. USING THE PRINTER

NOTE: All explanations and examples in this section assume that you have already:

- 1) Properly connected the PC-1500 Computer to the CE-150 Interface.
- 2) Provided power to the CE-150 with a SHARP EA-150 electrical adaptor.
- 3) Loaded the pens into the printer.
- 4) Loaded the paper into the printer.

If you have not done these things, please return to Section A of this Chapter for instructions.

C.1. CE-150 PRINTER SPECIFICATIONS

Characters/line:	4, 5, 6, 7, 9, 12, 18, or 36 depending on character size chosen.
Character Size:	1.2 x 0.8 mm to 10.8 x 7.2 mm depending on character size chosen.
Printing Speed:	Maximum: 11 Characters per second when printing smallest characters.
Rotation:	Characters may be printed in either of two directions on either of two axes.
Colors:	4 – Red, Blue, Green, Black.
Graphing System:	X–Y axis plotting.
Paper Feed:	Manual or Programmable.

C.2. The TEST Command

The first thing you will want to do is to test the functioning of the CE-150 printer. With the Computer and Interface ON, type:

TEST (and press **ENTER**)

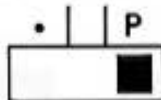
The printer will now draw 4 boxes, each a different color. The color of the boxes, from left to right, is what you chose as Colors 0 through 3 when you inserted the pens.

C.3. Printing Calculations

Using the CE-150 Interface it is possible to make a printed copy of a series of manual calculations performed on the PC-1500 Computer as in Figure 1:

```
Figure 1:  12000*.065          780
            780+25.56          805.56
            SIN 30              0.5
            TX=.065             0.065
            P=20000*TX          1300
            P/12                108.333333
```

To do this, simply set the Print Switch on the Interface to the "P" position:



To prevent the automatic printing of manual calculations, the Print Switch must be placed in the "•" position. With the switch in this position, you may still print out selected results by prefacing the computation with an LPRINT command (see LPRINT). Other commands which cause printing or drawing, such as LLIST, LINE, and others, are also functional in this mode.

When printing, the color used will be the color which was previously specified. If you have just turned the machine on, this will be whatever color pen you have selected to correspond to Color 0. To change the color, you must issue the Color command (see appropriate section).

The character size used to print manual calculations is dependent on the previous specification. If the previous character size specified was size 1 or size 2, then this size remains in effect. If the previous size, was larger than 2, then size 2 is used.

Automatic printing causes the printer mode to be set to TEXT. If you were in GRAPH mode and wish to return to this mode, you must issue the GRAPH command. (Printer modes are explained in the next section).

C.4 Printer Modes

As the printer is operating it may be in one of two modes; TEXT or GRAPH. These modes correspond roughly to human typing versus human drawing. Since most commands work only in one mode or the other, it is important to select the proper mode before issuing instructions.

The TEXT mode is used for printing numbers and characters. The width of the printer's paper is divided into columns, the number of which is related to the specified character size. Vertical and horizontal tabbing commands are provided to format text information.

In the GRAPH mode, a variety of diverse figures, charts, and tables may be created. Commands to draw both solid and dashed lines, using either a direct or relative coordinate system are available. All drawings utilize a normal X-Y coordinate scheme.

To specify TEXT mode, the statement:

TEXT

is sufficient. Certain commands (discussed later) cause an automatic switch to the TEXT mode.

Specifying the GRAPH mode is equally simple. The statement:

GRAPH

will initiate this mode, setting the pen to the far left side of the paper.

C.5. Listing Programs

The LLIST command causes the current program, or portions of the program to be printed. Because selective printing of program sections is possible, the LLIST command is extremely helpful during the program development process.

The form of the LLIST command is similar to the form of the LIST command. Because LLIST is more versatile, there are subtle differences. The LLIST forms are as follows:

LLIST

- Prints all program lines currently in the program memory.

LLIST expression

- Prints only the program line whose line number is given by expression.

LLIST , expression

- Prints all program lines up to, and including, the line whose number is given by the expression.

LLIST expression,

- Prints program lines beginning with the line whose number is given by expression.

LLIST expression 1, expression 2

- Prints program lines beginning with the line whose number is given by expression 1 and ending with the line whose number is given by expression 2. Thus, if the command is:

```
LLIST 100, 150
```

then all lines between 100 and 150 (if any) will be listed.

LLIST "label"

- Prints the program line containing the given label.

LLIST "label",

- Prints program lines beginning with the line containing the given label and continuing to the end.

NOTE: Specification of a label which does not exist will be signaled by an ERROR 11 message.

When printing a program, the color used will be the color which was previously specified. If you have just turned the machine on, this will be whatever color pen you have selected to correspond to Color 0. To change the color, you must issue the Color command (see appropriate section).

The character size used to list a program is dependent on the previous specification. If the previous character size specified was size 1 or size 2, then this size remains in effect. If the previous size was larger than 2, then size 2 is used.

The LLIST command causes the printer mode to be set to TEXT. If you were in GRAPH mode and wish to return to this mode, you must issue the GRAPH command.

While listing a program the PC-1500 computer attempts to justify the program lines for readability. This is done by leaving spaces in the line numbers. Line numbers which are 1 to 3 digits wide will be right justified within a 3 character field. Line numbers which are 4 or 5 digits wide will be printed in a 5 character field:

```
10: REM WIDTH 3
20: REM  "
300: REM  "
2001: REM WIDTH 5
2010: REM  "
```

C.6. PROGRAMMABLE PRINTER CONTROL**C.6.1. CSIZE**

The CSIZE command specifies the size of the characters for all subsequent printing. There are nine sizes available, ranging from 36 characters per printed line to 4 characters per printed line. The form of the CSIZE command is:

CSIZE expression
(either mode)

The expression must evaluate to a number in the range 1 through 9. The width and height of the characters for each size is given in the following table:

Table 1:

CSIZE	1	2	3	4	5	6	7	8	9
Characters per printed line.	36	18	12	9	7	6	5	4	4
Height of each character (mm)	1.2	2.4	3.6	4.8	6.0	7.2	8.4	9.6	10.8
Width of each character (mm)	0.8	1.6	2.4	3.2	4.0	4.8	5.6	6.4	7.2

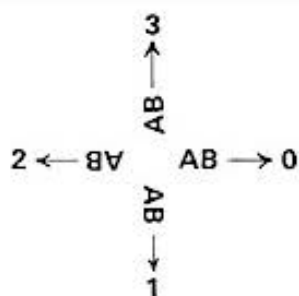
C.6.2. ROTATE

The ROTATE command is used, in GRAPH mode only, to specify the direction in which printing occurs. Four directions are possible; Up, Down, Left to Right, and Right to Left (with the letter upside down). The directions are illustrated in Figure 1. The form of the ROTATE command is:

ROTATE expression
(GRAPH mode only)

The expression must evaluate to a number in the range 0 to 3. ROTATE 0 designates the normal manner of printing characters from Left to Right.

Figure 1:



C.6.3. COLOR

The COLOR command allows the specification of the pen to be used for all subsequent printing and drawing. If each pen position contains a different color pen, then the COLOR command may be used to change pen colors. The form of the COLOR command is:

COLOR expression
(either mode)

The expression must evaluate to an integer in the range 0 to 3. Each integer corresponds to a different pen. The color represented by the integer will vary depending on the order in which the pens were loaded in the carrier. The correspondence may be determined by the TEST command (described above).

Numbers which are not integers but which are in the range 0 through 3 will be truncated to integers. All other numbers will cause an ERROR 19.

When the PC-1500 Computer is turned OFF and then ON, again the pen which corresponds to zero is selected.

In the TEXT mode, execution of the COLOR command will cause the pen position to be reset to the left side of the paper. In the GRAPH mode, the pen will return to its previous position.

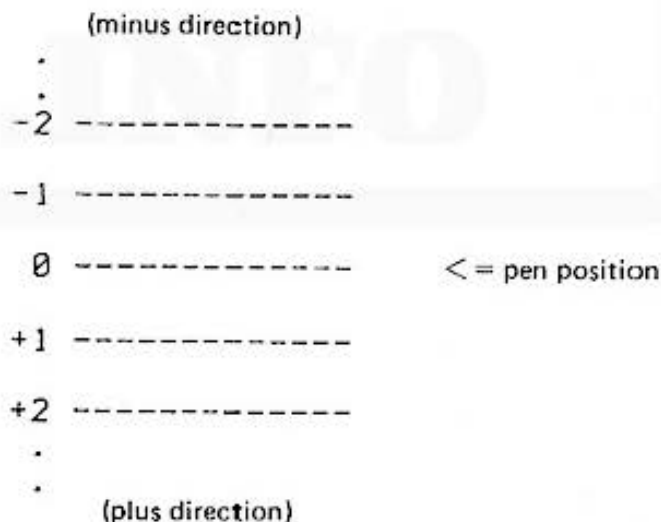
C.6.4. LF (Line Feed)

The LF command causes the paper in the printer to be moved forward or backwards. The form of the command is:

LF expression
(TEXT mode only)

If the expression evaluates to a positive number the paper is advanced the number of lines specified by the expression. An expression which evaluates to a negative number will cause the paper to be pulled back the number of lines specified by the expression. This is illustrated by Figure 2:

Figure 2:



The actual distance the paper is moved is related to the character size in effect when the LF instruction is specified.

When the paper is traveling in the reverse direction (i.e. it is being pulled back) an internal counter prevents it from moving backwards more than 10.24 centimeters (about 4 inches).

NOTE: Do not attempt to insert paper while the paper feed mechanism is operating. This can damage the printer.

C.6.5. LPRINT

The LPRINT command is the main command for displaying text information on the printer. It is similar in nature to the PRINT command of the PC-1500, and they share several of the same forms. However, because of the additional features available on the printer, the actions of the LPRINT instruction are more complex. We will therefore concentrate on the subtleties involved in using the LPRINT statement.

The following discussion of the LPRINT command assumes TEXT mode only. Although the given forms may work in GRAPH mode, their operation will be different.

The printing of a single item remains generally the same:

LPRINT item

where item is an expression, character string, number, or name of a variable whose contents will be printed. As usual, characters are left justified and numbers are right justified.

Like the cursor on the display, if the pen is not positioned at the left of the paper, printing will begin from the pen's position. The position of the pen may be changed by the LCURSOR statement or by the TAB clause (see below).

A problem occurs when one tries to print an item which is too big to fit on one print line because of the current character size. If the item is a number an ERROR 76 will result. If the item is a character string, the string will be continued on the next line.

Concern for the size of a printed item is also important for the two-item LPRINT statement, whose form is:

LPRINT item 1, item 2

Using CSIZE 1 guarantees that two numeric items will be printed on the same line. In this case, the items will be justified in the usual manner within the two halves of the printed line. For LPRINTs involving strings the picture is not so clear. If the items both fit, they are justified as usual and printed on the same line. If they do not fit, the results are usually split across two lines. For larger character sizes the two items are printed on two successive lines.

The semicolon may also be used in the LPRINT statement. It serves both to indicate minimum spacing of items and, at the end of a statement, to group successively printed items on the same line. In either case, if the total length of the items exceeds the capacity of the print line, the items are printed on successive lines (as many as necessary). The LPRINT with the semicolon has the form:

LPRINT item 1 ; item 2 ; ... (etc)

or the form:

LPRINT item-list;

Several of the aforementioned forms are used in the following demonstration program:

```
10 A$ = "ABCDEFGH"
20 B = 123456
30 FOR I = 1 TO 3
40 CSIZE I
50 LPRINT AS
60 LPRINT AS, B
70 LPRINT AS; B
80 LF 5
90 NEXT I
```


Which produces the output:

```

ABCDEF G      123456
ABCDEF G
ABCDEF G 123456

```

```

ABCDEF G
ABCDEF G
                123456
ABCDEF G 123456

```

```

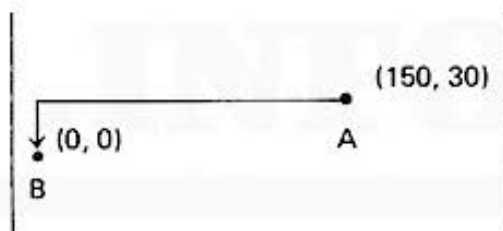
ABCDEF G
ABCDEF G
                123456
ABCDEF G 1234
56

```

The **LPRINT** command may be used without an item specification, in either **TEXT** or **GRAPH** mode:

LPRINT

Used in this manner, it will cause a Carriage Return and a single Line Feed. It **does not**, however, reset the counters of the **GRAPH** mode. Thus, in the following example, although the **LPRINT** statement has been used to move the pen from Point A to Point B, the printer believes itself to be at coordinates (150, 30) and will execute all subsequent commands as if it were.



The **LPRINT** statement also incorporates a **USING** clause which operates in the same manner as it does in the **PRINT** statement. The **USING** clause may only occur in an **LPRINT** statement which is executed in the **GRAPH** mode.

C.6.6. **LCURSOR**

The **LCURSOR** statement allows positioning of the pen in a manner analogous to the **CURSOR** statement of the display. The form of the **LCURSOR** statement is:

```

LCURSOR position
(TEXT mode only)

```

The character position to which the pen may be moved is, of course, dependent on the character size in effect. In general, the pen may be positioned at one space less than the maximum for the character size. For a list of the print line widths at each character size, refer to the **CSIZE** command in this section.

C.6.7. TAB

The TAB statement is identical to the LCURSOR statement, except that it may be used within an LPRINT statement. This type of LPRINT statement has the form:

```
LPRINT TAB position; item-list
```

The same comments which apply to the position expression of LCURSOR apply to TAB. If the item-list is empty the net result of the instruction listed above will be a Line Feed.

C.6.8. SORGN (Set ORiGiN)

The SORGN command is used to establish the origin of the X-Y coordinate system for subsequent graphing commands. The SORGN statement makes the current pen position the origin. Thus, this instruction is usually used directly after an instruction which moves the pen to a given spot on the paper. The form of the SORGN command is simply:

```
SORGN  
(GRAPH mode only)
```

NOTE. The CE-150 printer allows a pen position to be specified which is outside the range of drawable positions. In this case the pen moves as far as it can and then "cuts off". If the pen is moved into this imaginary realm and then the SORGN command is issued, subsequent printing or drawing statements will have no effect. It would thus appear as if the program was in error or that the Interface was damaged.

The following program sets the origin to 100 units up and 100 units to the right of the pen's current position. It then draws a 10 unit box with one of the boxes' corners at the new origin:

```
10 GRAPH  
20 LINE (0,0) - (100,100), 9  
30 SORGN  
40 LINE (0,0) - (10,10), 0,0,B  
50 TEXT  
60 END
```

C.6.9. GLCURSOR

The GLCURSOR statement moves the pen to any X-Y coordinate without drawing a line. The form of the GLCURSOR statement is:

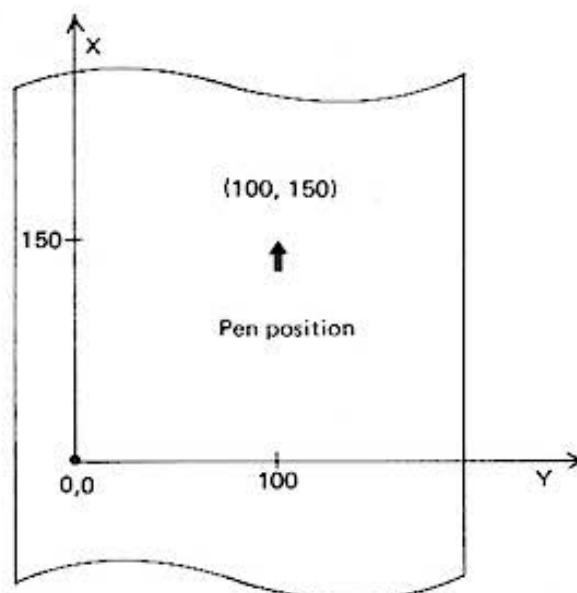
```
GLCURSOR (expression1, expression2)
```

Both expressions must evaluate to a number in the range -2047 to +2047. Expression1 represents the X distance to the destination point and expression2 represents the Y distance to the destination point.

NOTE If the destination point is outside the range of points to which the pen may actually move, the pen will stop at one side of the paper. Internally, the counters which control the pen are still counting toward the goal.

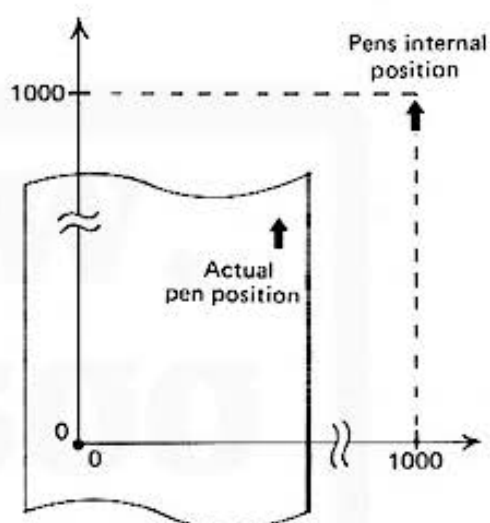
Examples of the use of the GLCURSOR statement follow:

In the example at the right, the pen is moved to position (100, 150).



In this example, the pen is incorrectly positioned in the "imaginary" region at position (1000, 1000).

The pen actually moves toward the right margin and scrolls the paper back. As it reaches the right side it continues upward until it reaches the built-in backing limit of 10 cm., at which time it stops.



C.6.10. LINE





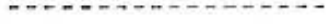





The LINE command is the primary command of the GRAPH mode. It specifies the movement of the pen from one point to another. If the pen is down as it moves, a line is drawn. The LINE command also allows dashed lines to be drawn with eight different dash lengths. The first from of the LINE command is:

LINE (X1, Y1) – (X2, Y2), line-type, color

The starting point of the line is determined by the values of the expressions X1 and Y1. The destination point of the line is determined by the values of the expression X2 and Y2. Both the X and the Y values must be in the range –2048 to +2047. A value specification which exceeds this range will produce an error.

The line-type, and color parameters are optional. If they are omitted, the values used are the values which were in effect before the command. Color is, of course, one of the colors in the range 0 to 3. The line-type must be an expression which evaluates to a number in the range 0 through 9. Table 2 identifies the meaning of each option:

Table 2:

Line-type Value	Resulting Line Size	
0	Solid Line	0 
1	0.4 mm dash	1 
2	0.6 mm dash	2 
3	0.8 mm dash	3 
4	1.0 mm dash	4 
5	1.2 mm dash	5 
6	1.4 mm dash	6 
7	1.6 mm dash	7 
8	1.8 mm dash	8 
9	Pen Up (no line)	9 

Used in another manner the LINE command interprets the point specifications as the endpoints of a diagonal. It will then proceed to draw the box represented by the diagonal. The form for this LINE statement is:

```
LINE (X1, X2) - (Y1, Y2), line-type, color, B
```

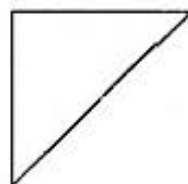
The upper case B indicates that this is a box drawing command. The other parameters are the same as for the previous form of the command.

The final form of the LINE command allows multiple point specifications to be made. Each point, after the first, represents a destination endpoint of the next line segment to be drawn. The current position is assumed as the starting endpoint for the line segment. This form of the LINE command is:

```
LINE (X1, Y1) - (X2, Y2) - ... (X6, Y6), line-type, color
```

The three dots in the form above are used to indicate that a series of point specifications (up to six in a row) may be given. Note that the B parameter may NOT be used in this form of the command. An example program to draw a triangle using four line segments is given below. Line 15 serves merely to establish the origin, while the triangle is drawn by line 20:

```
10: GRAPH
15: LINE (0, 0) - (10
    0, 0), 9: SORGN
20: LINE (0, 0) - (50
    , 50) - (-50, 50) -
    (-50, -50) - (0, 0
    ), 0, 0
30: TEXT
```

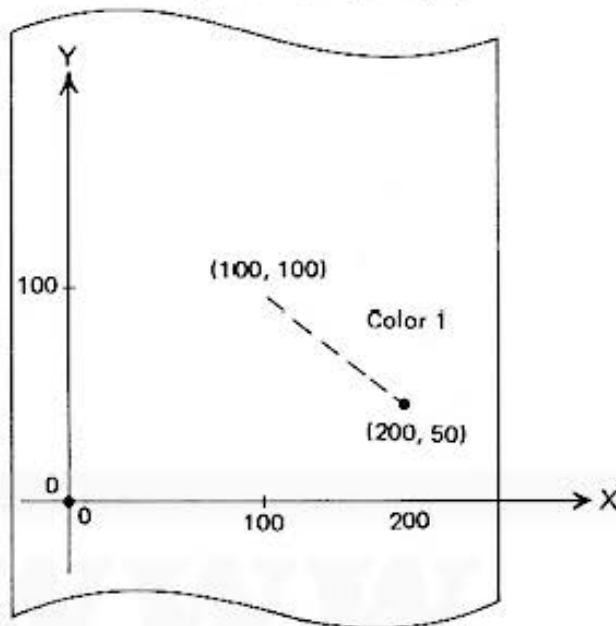


C.6.11. RLINE

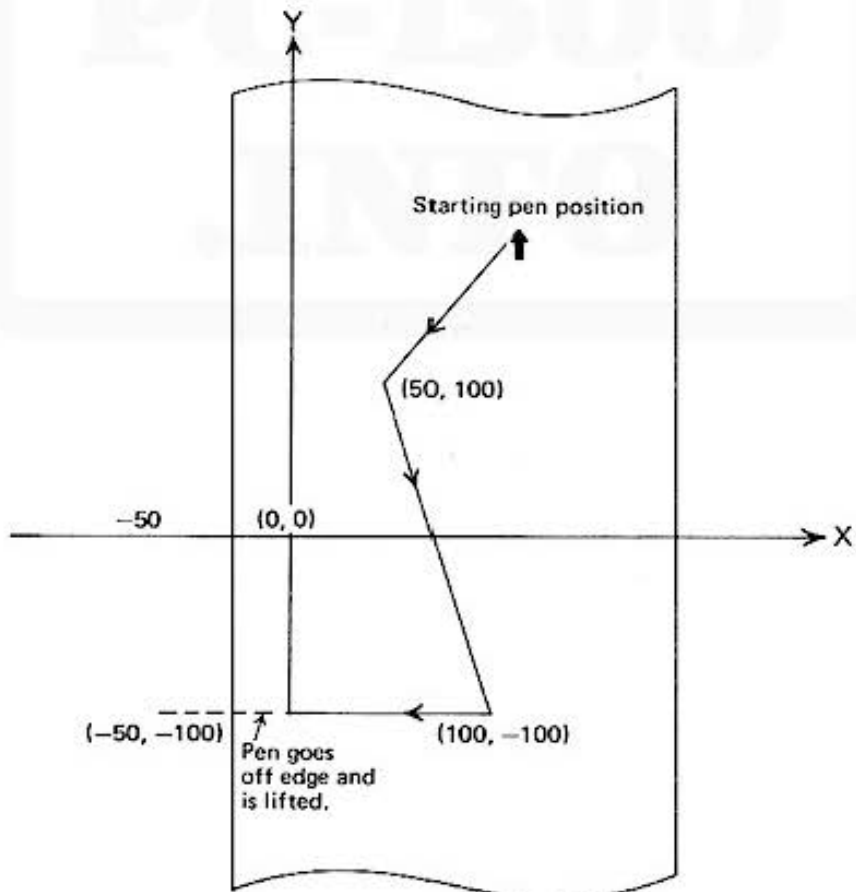
The RLINE command is the same as the LINE command except that all of the point specifications represent a position relative to the current pen position, rather than to the origin. The forms of the RLINE commands are the same as those of the LINE commands with the substitution of the word RLINE for LINE.

Examples follow:

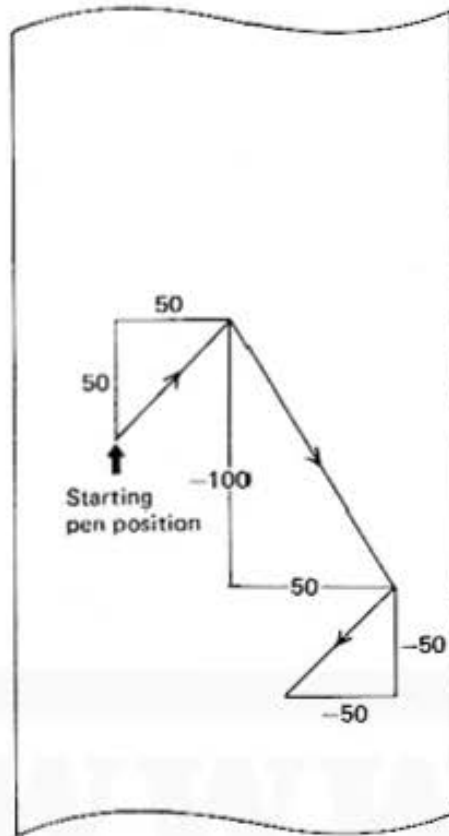
LINE (100, 100) - (200, 50), 2, 1



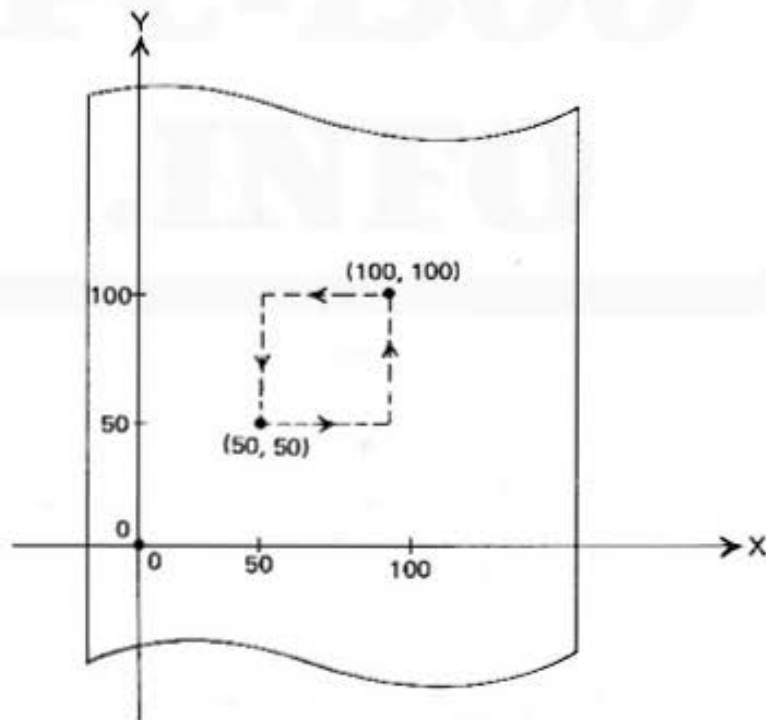
LINE - (50, 100) - (100, -100) - (-50, -100)



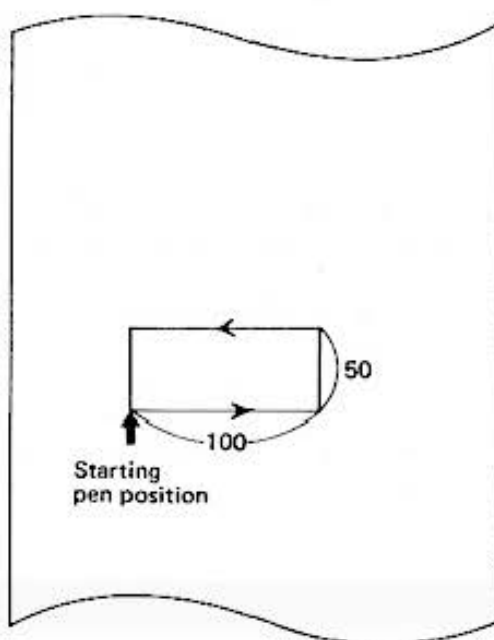
RLINE - (50, 50) - (50, -100) - (-50, -50)



LINE (50, 50) - (100, 100), 2, , B



RLINE – (100, 50), , , B



VII. RESERVE MODE

A. Defining and Selecting Reserve Keys

An important, labor-saving feature of the SHARP PC-1500 is the six keys whose function may be redefined. These Reserve keys allow the computer user to specify frequently typed phrases or keywords which are then recalled with a touch of a button. The Reserve keys are the six keys, in the top row of the keyboard, which are labeled !, ", #, \$, % and &. Each of these keys may be assigned up to three phrases or keywords for a total of 18 stored phrases.

The assignment of phrases to keys is performed in the third of SHARP's modes; the RESERVE mode. To enter the RESERVE mode press:

SHIFT **MODE**

The mode indicator at the top of the display should now read RESERVE. To escape from the RESERVE mode, simply press the MODE key once.

Because each Reserve key may be used to recall any of three stored phrases, there must be a method of selecting which phrase is recalled when a Reserve key is pressed. This method is a key in the lower left corner marked with the symbol (\dagger). Called the Reserve Select key, this button selects which stored phrases currently correspond to the Reserve keys. It is important to note that the Reserve Select key changes the correspondence for all of the Reserve keys at once. That is to say that the Reserve Select key selects a group of phrases, each of which corresponds to a single Reserve key. Which group is currently selected is indicated by the small Roman numerals (I, II, and III) at the top of the display.

To assign a phrase to a Reserve key, enter the RESERVE mode (press SHIFT MODE) and use the Reserve Select key to select the group (I, II, or III) in which the phrase will be stored. Then press the appropriate Reserve key (!, ", #, \$, %, or &). Depending on which Reserve key you pressed, the display will appear something like the following:

```
RESERVE  I  •
F 6 :
```

(The number following the F, six in our example, represents which key was pressed.) When this prompt appears you may key in the phrase to be stored. As an example type the following:

```
R U N 1 0 0 ENTER
```

This phrase is now associated with the Reserve key you selected.

Let's test this. Return to the RUN mode by pressing the MODE key. Press the Reserve key you used in our example. The display will now show the command just stored:

```
RUN 100_  RUN  I  •
```

If you press the ENTER key the computer will attempt to run a program at line 100. The advantage of the Reserve key is that you don't have to type an entire command each time you desire to issue it.

A special notation allowed in the Reserve mode could have saved us some trouble in the previous example. This notation is the use of the @ (At sign) to represent the ENTER command. If we had assigned the phrase "RUN 100@" to our Reserve key, execution of the program would have begun as soon as we returned to the RUN mode and pressed the Reserve key. To demonstrate this, enter the following statements as line 222:

```
222 BEEP 5,50 : END
```

Now enter the RESERVE mode and define one of the Reserve keys using these keystrokes:

```
G O T 0 2 2 2 0 ENTER
```

(Notice that the ENTER keystroke is still required to define the Reserve key itself).

Return to the RUN mode and press the Reserve key just defined. You will observe that the ENTER keystroke after the Reserve keystroke is now superfluous.

Actually our noisy example could have been accomplished by assigning the phrase:

```
BEEP 5,50@
```

directly to a Reserve key. Try it.

B. Identifying Reserve Keys

As you increase your use of the Reserve keys, you will want to remember which key has been assigned which function. The PC-1500 allows you to store three strings of characters (one for each group of Reserve keys) which identify the functions of the keys. These strings are

analogous to comments in the PROgram mode.

Identifying strings (called "templates") are created in the RESERVE mode. Switch to this mode and select the appropriate group of Reserve keys using the Reserve Select key. Then instead of pressing a Reserve key, as you normally would, type a template and press ENTER. The template is then stored in association with the group.

As an example, pretend that we have assigned Reserve keys one through six (in group I) the names of the Trigonometric Functions (Sine, Cosine, Tangent, ArcCosine, ArcSine, and ArcTangent). In order to remember which key corresponds to which function we will specify a template. To do this switch to the RESERVE mode (press SHIFT MODE), and use the Reserve Select key to select group I (a Roman numeral I will appear on the display). Now type:

"SIN COS TAN ACS ASN ATN "

Keystrokes:

SHIFT * S I N SPACE C O S SPACE
 T A N SPACE A C S SPACE A S N
 SPACE A T N SPACE
 SHIFT * ENTER

The template has now been defined and stored.

Return to the RUN mode by pressing the **MODE** key. To recall the meaning of the Reserve keys, simply press the **RCL** (recall) button and there they are! Press the **RCL** key once more and the prompt returns.

Templates may be created for each group of reserve keys, and may be up to 26 characters in length. Note that the template is strictly a reminder to you; the words or letters you store in a template have no meaning to the computer.

C. Deleting reserve programs

As you know, **N E W** **ENTER** keys clear all reserve memories.

Please note that the above key operation must be done in the RESERVE mode.

VIII. BEGINNING PROGRAM EXECUTION

A. The DEF key

The DEF key (short for define) provides several time-saving shortcuts . . .

A.1. Running DEFInable Programs

The DEF key is the third method of beginning a program. As we have seen in the section on the RUN command, a program may be "labeled" with a letter. The DEF key may be used to quickly initiate a labeled program. This is done by pressing the DEF key followed by the alphabetic key which corresponds to the label of the program. The following keys may be used in this manner:

A, S, D, F, G, H, J, K, L, Z, X, C, V,
B, N, M, SPACE and =

As an example, enter the following statements to create three labeled programs:

Program Listing:

```
10 "Z" : GOSUB 500
20 PRINT "Z KEY"
30 END
140 "A" : GOSUB 500
150 PRINT "A KEY"
160 END
270 " " : GOSUB 500
280 PRINT "SPACE KEY"
290 END
500 CLS : PAUSE "YOU PRESSED THE ";
510 RETURN
```

In RUN mode, try beginning each program with the DEFine key. Notice that specifying a letter for which no corresponding labeled program exists will produce an ERROR 11.

A.2. Pre-assigned Keywords

A few of the most frequently used keywords have been permanently assigned to a single alphabetic key in the second row of the keyboard. These keywords may be retrieved, in any mode, by pressing the DEFine key followed by one of the alphabetic keys. For example, to retrieve the keyword USING type:

DEF E

The keywords available and their corresponding alphabetic keys are:

<u>Alphabetic key</u>	<u>Keyword</u>	
Q	INPUT	
W	PRINT	
E	USING	
R	GOTO	
T	GOSUB	
Y	RETURN	
U	CSAVE	} Can be used, when the printer/ cassette interface is connected with the computer.
I	CLOAD	
O	MERGE	
P	LIST	

Template

INPUT	PRINT	USING	GOTO	GOSUB	RETURN	CSAVE	CLOAD	MERGE	LIST
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Two templates are supplied with your computer. Use them to identify the functional operation assigned to the definition keys.

A.3. The AREAD statement

Labeled programs, which are initiated using the DEF key, may be given a single value each time the program is run without using an INPUT statement. The reading of the value is performed by the AREAD statement, which must follow the program label on the same line. The AREAD statement has the form:

AREAD variable-name

where variable-name is a legal numeric or character variable name.

To pass a value to a program which incorporates an AREAD statement, the user types the value, presses the DEF key, and then presses the alphabetic key corresponding to the program label.

As an example, enter the following two programs:

```

Program Listing:
10 "X" : AREAD TM
20 TIME = TM
30 PRINT "TIME SET TO "; TM
40 END
100 "Z" : AREAD DS
110 PRINT "TODAY IS "; DS
120 END
    
```

Return to the RUN mode and begin the program labeled X by typing the month, day, time, and DEF X:

1 2 3 1 0 2 . 4 0 0 0 DEF X

This program will set the system clock to whatever time you specify before the DEF keystroke (see TIME function).

To begin the program labeled Z type a day of the week and DEF Z:

F R I D A Y DEF Z

B. Automatic Program Initiation

Not only may programs be started quickly and easily using the DEF key, but they may be started totally automatically when you turn the PC-1500 on.

To cause this to happen, you use the ARUN statement. This statement must be the very first statement in the program memory or it will be ignored. In addition, several other conditions are necessary for the ARUN statement to work. These are that the PC-1500 was turned OFF while in the RUN mode, and that no errors are detected as the PC-1500 is turned ON.

The following program uses the ARUN statement to greet the computer operator:

Program Listing:

```
10 ARUN
30 CLS
50 BEEP 5, 50
70 PRINT "WELL, HI THERE!"
90 END
```

C. Comparison of Initiation Methods

Although the various methods of beginning a program superficially achieve the same result, their internal operation differs. In order to exploit these differences to your advantage it is necessary to discuss the storage of data within the computer. Also included is a section comparing the various internal preparations which are made by the PC-1500 before running a program.

C.1. Fixed Memory Area

Although all variables of the same type are utilized in the same manner, they are not treated the same internally. The PC-1500 includes a "fixed memory" area with enough storage space for 26 numeric variables and 26 character string variables (string size of 16 characters). Consequently the variables A through Z, and the variables AS through ZS are permanently allocated within this area.

All other variables, including those with two character names, are allocated within the main memory area of the computer. This main memory area is also shared by the instructions of the program, although the variables are allocated beginning at the opposite end of memory from the instructions. Because the program instructions and data share the same area it is possible for them to use up all available storage. In this case an ERROR in the range 177 through 181 will occur.

It is important to realize that the two memory areas are not treated the same upon program initiation. This is explained in the chart in the next section. Basically, variables in the fixed memory area are never cleared except by an explicit CLEAR statement. Those in main memory are cleared whenever a program is begun with the RUN command.

One other idiosyncrasy about the fixed memory area is that the data in this area may be redefined as an array whose name is @ (At Sign), for numeric variables, and @\$ for string variables. Thus, the designation @(1) is the same storage location as the variable A and @(26) is the same storage location as the variable Z. The designation @\$ (5) refers to the same location as ES, and the designation @\$ (20) refers to the same location as TS. For obvious reasons subscripts above 26 are not allowed. Notice that the arrays @ and @\$ need not be dimensioned before use.

C.2. COMPARISON CHART OF PROGRAM**INITIATION METHODS**

	<u>RUN</u>	<u>GOTO</u>	<u>DEF</u>
Display is cleared.	Y	Y	N
Cursor returns to first column.	Y	N	N
WAIT interval is set to infinite.	N	N	N
Trace mode is altered.	N	N	N
Fixed Memory Area is cleared.	N	N	N
Main Memory Area is cleared.	Y	N	N
FOR-NEXT, GOSUB internal stack is cleared.	Y	Y	Y
ON ERROR GOTO is cancelled.	Y	N	N
DATA pointer for READ operation is RESTORED.	Y	N	N
USING format is cancelled.	Y	N	N





APPENDICES

A. A BRIEF TABLE OF ABBREVIATIONS

Printer Commands

COLOR	COL. COLO.	LPRINT	LP. LPR. LPRI. LPRIN.
CSIZE	CSI. CSIZ.		
GLCURSOR	GL. GLC. GLCU. GLCUR. GLCURS. GLCURSO.	RLINE	RL. RLI. RLIN.
GRAPH	GRAP.	ROTATE	RO. ROT. ROTA. ROTAT.
LCURSOR	LCU. LCUR. LCURS. LCURSO.	SORGN	SO. SOR. SORG.
LF	--	TAB	--
LINE	LIN.	TEST	TE. TES.
LLIST	LL. LLI. LLIS.	TEXT	TEX.

Cassette Commands

CHAIN	CHA. CHAI.	MERGE	MER. MERG.
CLOAD	CLO. CLOA.	PRINT #	P. # PR. # PRI. # PRIN. #
CLOAD?	CLO.? CLOA.?		
CSAVE	CS. CSA. CSAV.	RMT OFF RMT ON	RM. OF. RM. O.
INPUT #	I. # IN. # INP. # INPU. #		

Statements

AREAD	A. AR. ARE. AREA		
ARUN	ARU.	GOSUB	GOS. GOSU.
BEEP	B. BE. BEE.	GOTO	G. GO. GOT.
CLEAR	CL. CLE. CLEA.	GPRINT	GP. GPR. GPRI. GPRIN.
CLS	--	GRAD	GR. GRA.
CURSOR	CU. CUR CURS. CURSO.	IF	--
DATA	DA. DAT,	INPUT	I. IN. INP. INPU.
DEGREE	DE. DEG. DEGR. DEGRE.	LET	LE.
		LOCK	LOC.
DIM	D. DI.	NEXT	N. NE. NEX.
END	E. EN.	ON	O.
ERROR	ER. ERR. ERRO.		
FOR	F. FO.		
GCURSOR	GCU. GCUR. GCURS. GCURSO.		

PAUSE	PA. PAU. PAUS.	STEP	STE.
PRINT	P. PR. PRI PRIN.	STOP	S ST. STO.
RADIAN	RAD. RADI. RADIA.	THEN	T. TH. THE.
RANDOM	RA. RAN. RAND. RANDO.	TRON	TR. TRO.
READ	REA.	TROFF	TROF.
		UNLOCK	UN. UNL. UNLO. UNLOC.
REM	--		
		USING	U. US. USI. USIN.
RESTORE	RES. REST. RESTO. RESTOR.		
		WAIT	W. WA. WAI.
RETURN	RE. RET. RETU. RETUR.		

Commands

CONT	C CO. CON.	NEW	--
LIST	L. LI. LIS.	RUN	R. RU.

Functions

ABS	AB.	MEM	M. ME.
ACS	AC.	MIDS	MI. MID.
AND	AN.	NOT	NO.
ASC	--	OR	--
ASN	AS.		
ATN	AT.	PI	--
CHRS	CH. CHR.	POINT	POI. POIN.
COS	--	RIGHTS	RI. RIG. RIGH. RIGHT.
DEG	--	RND	RN.
DMS	DM.	SGN	SG.
EXP	EX.	SIN	SI.
INKEYS	INK. INKE. INKEY.	SQR	SO.
INT	--	STATUS	STA. STAT. STATU.
LEFTS	LEF. LEFT.	STR\$	STR.
LEN	--	TAN	TA.
LOG	LO.	TIME	TI. TIM.
LN	--	VAL	V. VA.

B. BATTERY REPLACEMENT FOR THE PC-1500

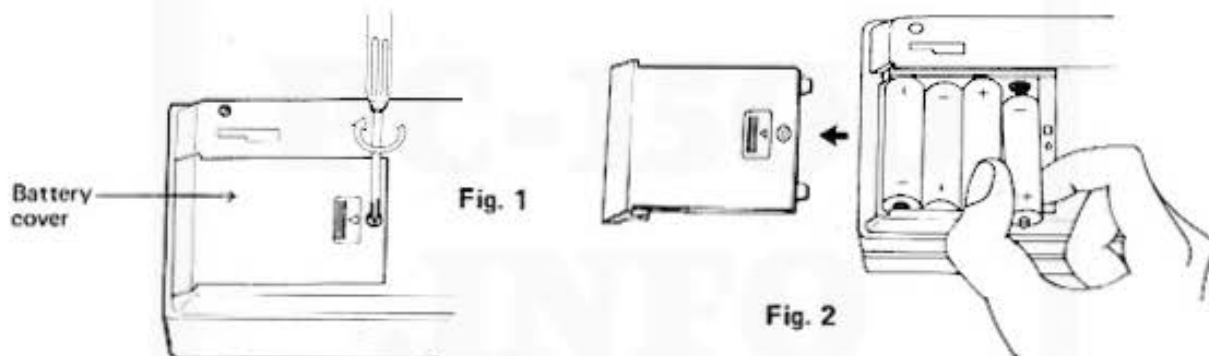
When replacing the batteries, these cautionary instructions will eliminate many problems:

- Always replace all four (4) batteries at the same time.
- Do not mix new batteries with used batteries.
- Use only: Dry battery (type AA, R6 or SUM-3, 1.5V) x 4

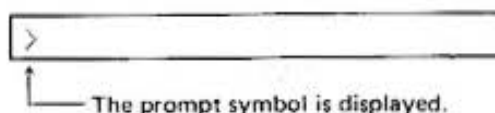
BATTERY REPLACEMENT PROCEDURE

Before the computer can be used, or whenever the battery indicator disappears, replacement of the batteries is necessary. Please follow this replacement procedure:

1. Turn off the computer by pressing the **[OFF]** key.
2. Remove the screw from the battery cover with a coin or a small screw driver (see Fig. 1).
3. Replace the 4 batteries. (Fig. 2)

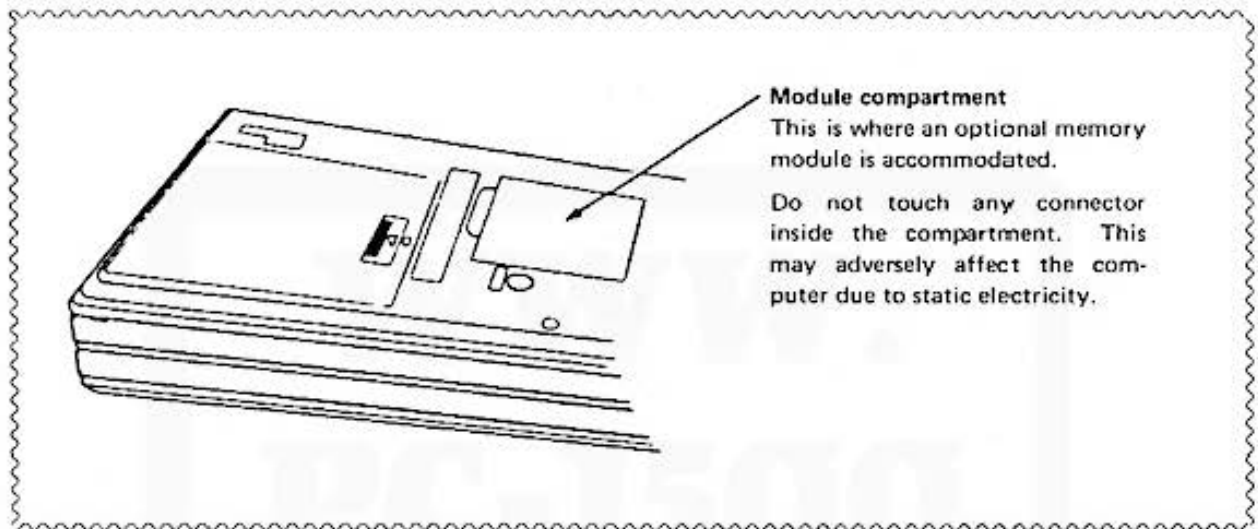


4. Push the battery cover in slightly while replacing the screw.
5. To proceed press the **[ON]** and **[CL]** keys, type **NEW B** and press **[ENTER]** key.
6. Check the following display.



If the display is blank or displays any other symbol than ">", remove the batteries and install them again, and check the display.

- Note:**
- If the computer will not be operated for an extended period of time, remove the batteries to avoid possible damage caused by battery leakage.
 - Keeping a dead battery may result in damage to the computer due to solvent leakage of the battery. Remove a dead battery promptly.
 - The rechargeable battery can not be used as a battery for PC-1500.
 - The AC adaptor, EA-150 for CE-150 is also used as an AC adaptor for the PC-1500, when it is separated from the CE-150.
- (Do not connect EA-150 to the computer, PC-1500, when the PC-1500 is connected to the printer/cassette interface, CE-150.)



C. ASCII CHARACTER CODE CHART FOR THE PC-1500

		Upper Bit Positions → b7, b6, b5							
		000	001	010	011	100	101	110	111
Low Bit Positions b4, b3, b2, b1 ↓	Hexa decimal	0	1	2	3	4	5	6	7
	0000	0			SPACE	0	@	P	
0001	1			!	1	A	Q	a	q
0010	2			"	2	B	R	b	r
0011	3			#	3	C	S	c	s
0100	4			\$	4	D	T	d	t
0101	5			%	5	E	U	e	u
0110	6			&	6	F	V	f	v
0111	7			[7	G	W	g	w
1000	8			(8	H	X	h	x
1001	9)	9	I	Y	i	y
1010	A			*	:	J	Z	j	z
1011	B			+	:	K	√	k	{
1100	C			,	<	L	¥	l	!
1101	D			-	=	M	π	m	}
1110	E			.	>	N	^	n	~
1111	F			/	?	O	—	o	■

E: PC-1500 ERROR CODE LIST

<u>Error Code</u>	<u>Explanation</u>
1.	<p>Syntax error: This error results from typing errors such as:</p> <p>missing information: 10: GOTO</p> <p>or invalid commands. 10: 5A = 1 or 10: NEW</p> <p>(because NEW may not be a statement.)</p> <p>< Display > ERROR 1 IN 10</p>
2.	<p>This error occurs when there is no FOR command corresponding to a NEXT command, or when there is no GOSUB command corresponding to a RETURN command.</p> <p>Ex. 10: FOR A = 1 TO 10 : : 100: NEXT B</p> <p>< Display > ERROR 2 IN 100</p>
4.	<p>This error occurs when there is no DATA corresponding to a READ command.</p> <p>Ex. 10: READ X, Y 20: DATA 10 30: END</p> <p>< Display > ERROR 4 IN 10</p>
5.	<p>This error occurs when an array variable is declared with the name of an existing variable.</p> <p>Ex. 10: DIM A (10, 10) 20: DIM A (5)</p> <p>< Display > ERROR 5 IN 20</p>
6.	<p>This error occurs when an array variable has been used without a DIM (dimension) statement.</p> <p>Ex. 10: CLEAR 20: A (3) = 1</p> <p>< Display > ERROR 6 IN 20</p>

PC-1500 ERROR CODE LIST

- | <u>Error Code</u> | <u>Explanation</u> | | | | | | |
|----------------------|---|----------------------|----------------|-------------|---|----------------------|-----------------|
| 7. | This error occurs when the variable name is inappropriate.
Ex. 10: A\$ = 10 or
10: FOR A\$ = 1 TO 10
< Display > ERROR 7 IN 10 | | | | | | |
| 8. | This error occurs when the dimension exceeds 3 in the declaration of array variable.
Ex. 10: DIM A (3, 4, 5, 6)
< Display > ERROR 8 IN 10 | | | | | | |
| 9. | This error occurs when the subscript number of an array variable exceeds the size of the array stated in the DIM command.
Ex. 10: DIM A (3)
20: A (4) = 1
< Display > ERROR 9 IN 20 | | | | | | |
| 10. | This error occurs when there is not enough memory available to create more variables.
Ex. <table border="1" style="margin-left: 40px;"> <thead> <tr> <th><u>key operation</u></th> <th><u>display</u></th> </tr> </thead> <tbody> <tr> <td>MEM [ENTER]</td> <td style="text-align: right;">7</td> </tr> <tr> <td>AB = 10 [ENTER]</td> <td style="text-align: right;">ERROR 10</td> </tr> </tbody> </table> | <u>key operation</u> | <u>display</u> | MEM [ENTER] | 7 | AB = 10 [ENTER] | ERROR 10 |
| <u>key operation</u> | <u>display</u> | | | | | | |
| MEM [ENTER] | 7 | | | | | | |
| AB = 10 [ENTER] | ERROR 10 | | | | | | |
| 11. | This error occurs when the specified line is not in the program.
Ex. 10: PRINT "X="; X: GOTO 5
< Display > ERROR 11 IN 10 | | | | | | |
| 12. | This error occurs when the USING command specifies incorrect format specifications.
Ex. 100: PRNT USING "### A#"; 10
< Display > ERROR 12 IN 100 | | | | | | |
| 13. | This error occurs when a program exceeds program-memory capacity or when the Reserve key specification exceeds the Reserve-area capacity.
Ex. <table border="1" style="margin-left: 40px;"> <thead> <tr> <th><u>key operation</u></th> <th><u>display</u></th> </tr> </thead> <tbody> <tr> <td>MEM [ENTER]</td> <td style="text-align: right;">7</td> </tr> <tr> <td>15 A = A + 1 [ENTER]</td> <td style="text-align: right;">ERROR 13</td> </tr> </tbody> </table> | <u>key operation</u> | <u>display</u> | MEM [ENTER] | 7 | 15 A = A + 1 [ENTER] | ERROR 13 |
| <u>key operation</u> | <u>display</u> | | | | | | |
| MEM [ENTER] | 7 | | | | | | |
| 15 A = A + 1 [ENTER] | ERROR 13 | | | | | | |

PC-1500 ERROR CODE LIST

<u>Error Code</u>	<u>Explanation</u>
14.	(1) FOR statements have been nested too deeply and the stack capacity has been exceeded. (2) While parsing an expression, buffer space has been exceeded.
15.	(1) GOSUB statements are nested too deeply and the stack area has been exceeded. (2) While parsing an expression, the string buffer size has been exceeded by the character strings handled.
16.	(1) The value specified is over 1E100 or less than - 1E100. Ex. 123E 99 (2) The value set by hexadecimals exceeds 65535. Ex. &1FFAB
17.	Data type (numerals, character strings) is inappropriate for calculation expression. Ex. 1 + "A" <input type="button" value="ENTER"/>
18.	Number of arguments inappropriate for expression. Ex. LEFTS ("ABC") <input type="button" value="ENTER"/> SIN (30, 60) <input type="button" value="ENTER"/>
19.	Specified numeric value is outside the permitted range. Ex. 10: DIM A (256) < Display > ERROR 19 IN 10
20.	When fixed memory array variables are specified, there is no '(' following '@' or '@\$'. Ex. 100: @\$ = "A" < Display > ERROR 20 IN 100
21.	A variable is required in the expression. Ex. 10: FOR 1 = 0 TO 10 < Display > ERROR 21 IN 10
22.	This error occurs when the program is loaded, and there is no memory space available for loading.
23.	This error occurs the time is set incorrectly. Ex. TIME = 131005.10 <input type="button" value="ENTER"/>

PC-1500 ERROR CODE LIST

<u>Error Code</u>	<u>Explanation</u>
26.	This error occurs when the command cannot be executed in the current mode. Ex. < RUN MODE > NEW ENTER
27.	This error occurs when an optional printer is not connected and there is no program which corresponds to the given label. Ex. <u>key operation</u> <u>display</u> DEF I ENTER ERROR 27
28.	This error occurs when a command or function code has been inserted inside " ", or when you try to substitute INPUT commands or AREAD commands for character variables. Ex. 10 INPUT AS <u>key operation</u> <u>display</u> DEF W ENTER ERROR 28
30.	This error occurs when a line number exceeds 65539. (65280 ~ 65539 : ERROR 1) Ex. 102235 A = 10 ENTER
32.	This error occurs when the graphic cursor is between columns 152 ~ 155 during execution of input commands (input code cannot be displayed) Ex. 100: GCURSOR 152 110: INPUT X < Display > ERROR 32 IN 110
177 ~ 181	During program creation the program has overwritten the data area. Overlapping of these two areas occurred.
0, 224 ~ 241	During execution of INPUT commands or AREAD commands incorrect input data is given. Ex. 10: INPUT A <u>key operation</u> <u>display</u> 123 PRINT ENTER ERROR 240

PC-1500 ERROR CODE LIST

<u>Error Code</u>	<u>Explanation</u>
36.	Data or characters cannot be displayed in accordance with the format specified by USING commands. Ex. 10: USING "#####.##" 20: PRINT 12345 The integer section together with its sign has exceeded 4 digits spaces.
37.	This error occurs in numeric calculations, when the calculation results have exceeded 9.999999999 E99.
38.	This error occurs when division has occurred using 0 as the denominator. Ex. 10: PRINT 5/0
39.	This error occurs when an illogical calculation has been made: <ul style="list-style-type: none"> * negative number logarithmic calculation Ex. LN (-10) * ASN, ACS in the case of X = 1 Ex. ASN (1.5) ACS (100) * Square Root of negative numbers Ex. SQR (-10)

Cassette Related Errors

40. Inappropriate specification for the expression.
41. SAVE and LOAD have been specified for the ROM area.
42. The Cassette file data is too large and cannot be LOADED.
43. While verifying data using the CLOAD? command the format of data to be loaded does not match the file format.
44. A CHECK SUM Error has occurred.

Printer Related Commands

70. (1) The pen has exceeded the coordinate range of:
-2048 ≤ X, Y ≤ 2047
(2) The pen will exceed the range upon execution of further commands.
71. (1) The paper has backed up more than 10.24 cm in the TEXT mode.
(2) The paper will back up more than 10.24 cm. upon execution of further commands (in TEXT MODE).

PC-1500 ERROR CODE LIST

<u>Error Code</u>	<u>Explanation</u>
72.	The value given is inappropriate for the value of TAB.
73.	A command has been used in the wrong printer mode (GRAPH/TEXT).
74.	The number of commas (,) in LINE or RLINE command is too large. Note: Entry of over seven commas results in an error. Also, if the first comma is omitted, more than six would cause an error.
76.	For LPRINT, when the printing of calculation results cannot be done on one line (in TEXT mode).
78.	(1) Pen(s) are in the process of being changed. (2) The LOW BATTERY state has not been corrected. (see Note 1) This error occurs when, for either of these two reasons, commands that move the pen (such as LPRINT and LINE) are not able to be executed.
79.	The color signal hasn't gone on. (see NOTE 2)
80.	Low Battery. (see NOTE 3)

NOTES:

- (1) If ERROR 78 is due to LOW BATTERY state of the printer, turn OFF the CE-150. After recharging the printer, turn the CE-150 ON. You may now continue.
- (2) The color signal is for COLOR and goes on only when the pen comes to the left side. When the pen is in this position, it is possible to know the number of the present pen color position.
- (3) After recharging, immediately push the PC-1500 ON key again and start operation.

F: SUGGESTIONS FOR FURTHER READING

BASIC Programming

Problem Solving and Structured Programming in BASIC by Elliot Koffman and Frank Friedman. (Addison-Wesley Publishing Co., Reading, Massachusetts), 1979.

basic BASIC by Donald M. Monro. (Winthrop Publishers, Inc. Cambridge, Massachusetts), 1978.

BASIC with Business Applications by Richard W. Lott. (John Wiley & Sons, New York, New York), 1977.

Practical BASIC Programs edited by Lon Poole. (OSBORNE/McGraw-Hill, Berkeley, California.), 1980.

General Reference

Introduction to Computers and Data Processing by Gary B. Shelley and Thomas J. Cashman. (Anaheim Publishing Co., Fullerton, California), 1980.

O: ORDER OF EXPRESSION EVALUATION

Calculations are performed in accordance with the following hierarchy; expressions in parentheses having the highest priority and logical operations having the lowest. If two or more operations of the same priority are found in the same expression or sub-expression, they are evaluated from left to right.

- 1) Expressions in Parentheses
- 2) Retrieval of values from variables, TIME, PI, MEM, INKEYS
- 3) Functions (SIN, COS, LOG, EXP, etc.)
- 4) Exponentiation (Example: $2A^3 = 2 * (A^3)$)
- 5) Arithmetic Sign (+, -)
- 6) Multiplication, Division (*, /)
- 7) Addition, Subtraction (+, -)
- 8) Comparison Operators (<, <=, =, >=, >, <>)
- 9) Logical Operators (AND, OR, NOT)

NOTES:

When both arithmetic sign and exponents are used in the same expression, the exponent is evaluated before the sign.

Example: -5^4 is evaluated to -625 instead of 625

Calculations within parentheses will be evaluated first. In expressions with several "layers" of parentheses, calculations start with the innermost pair and proceed to the outermost pair of parentheses.

Sample Evaluation

$$\begin{aligned}
 &7^2 + 3 * \sqrt{144} / \sqrt{81} + \text{SIN} (120 + 150) * -3 \\
 &7^2 + 3 * \frac{\sqrt{144}}{\sqrt{81}} + \text{SIN} (270) * -3 \\
 &7^2 + 3 * \frac{12}{9} + -1 * -3 \\
 &49 + 3 * \frac{12}{9} + -1 * -3 \\
 &49 + \frac{36}{9} + 3 \\
 &49 + 4 + 3 \\
 &\frac{53}{\quad} + 3 \\
 &\quad\quad\quad 56
 \end{aligned}$$

(CALCULATION RANGE)

Functions	Dynamic range
y^x (y^x)	$-1 \times 10^{100} < x \log y < 100$ $(y = 0, x \leq 0: \text{ERROR 39})$ $(y = 0, x > 0: 0)$ $(y < 0, x \neq \text{integer}: \text{ERROR 39})$
SIN x COS x TAN x	DEG: $ x < 1 \times 10^{10}$ RAD: $ x < \frac{\pi}{180} \times 10^{10}$ GRAD: $ x < \frac{10}{9} \times 10^{10}$ In TAN x , however, the following cases are excluded. DEG: $ x = 90(2n-1)$ RAD: $ x = \frac{\pi}{2}(2n-1)$ GRAD: $ x = 100(2n-1)$ (n : integer)
$\text{SIN}^{-1}x$ $\text{COS}^{-1}x$	$-1 \leq x \leq 1$
$\text{TAN}^{-1}x$	$ x < 1 \times 10^{100}$
LN x LOG x	$1 \times 10^{-99} \leq x < 1 \times 10^{100}$
EXP x	$-1 \times 10^{100} < x \leq 230.2585092$
\sqrt{x}	$0 \leq x < 1 \times 10^{100}$

Functions other than shown above can be calculated only when x stays within the following range.

$1 \times 10^{-99} \leq |x| < 1 \times 10^{100}$ and 0

(Ex.) \emptyset^{\emptyset} [ENTER] → ERROR 39
 \emptyset^5 [ENTER] → \emptyset
 $(-4)^{\emptyset.5}$ [ENTER] → ERROR 39
 $-4^{\emptyset.5}$ [ENTER] → -2

- As a rule, the error of functional calculations is less than ± 1 at the lowest digit of a displayed numerical value (at the lowest digit of mantissa in the case of scientific notation system) within the above calculation range.

X. COMMAND COMPARISON: PC-1211 vs. PC-1500

X.1 PC-1211 Instructions Available on the PC-1500

1. Functions

ABS
ACS
ASN
ATN
COS
DEG
DMS
EXP
INT
LOG
LN
 π (PI)
SGN
SIN
 $\sqrt{\quad}$ (Square Root)
TAN
 \wedge (exponentiation)

2. Statements

AREAD
USING
CLEAR
DEGREE
END
FOR-TO-STEP
GOSUB
GOTO
GRAD
IF
INPUT
LET
MEM

3. Commands

CONT
LIST
NEW
RUN

4. Cassette Commands

CHAIN
CLOAD
CLOAD?
CSAVE
INPUT #
PRINT #

NEXT
PAUSE
PRINT
RADIAN
REM
RETURN
STOP
THEN
USING

X-2 COMMANDS UNIQUE TO THE PC-1500

1. Functions

AND
ASC
CHRS
INKEYS
LEFTS
LEN
MIDS
NOT
OR
POINT
RIGHTS
RND
STATUS
STRS
TIME
VAL

2. Statements

ARUN
BEEP (not PC-1211)
CLS
CURSOR
GCURSOR
GPRINT
DATA
DIM
LOCK
ON ERROR
ON GOSUB
ON GOTO
POINT
RANDOM
READ
RESTORE
TRON
TROFF
UNLOCK
WAIT

3. Commands

(same as PC-1211)

4. Cassette Instructions

CHAIN
CLOAD
CLOAD?
CSAVE
INPUT#
MERGE
PRINT#
RMT OFF
RMT ON

5. Printer Instructions

COLOR
CSIZE
GCURSOR
GLCURSOR
GPRINT
GRAPH
LCURSOR
LF
LINE
LLIST
LPRINT
RLINE
ROTATE
SORGN
TAB
TEST
TEXT

Z. COMMAND REFERENCE TABLE

1. Functions

Function	Abbreviations	Remarks
ABS	AB.	Absolute value
ACS	AC.	\cos^{-1}
AND	AN.	exp. AND exp. [logical And]
ASC	ASC	Converts characters to ASCII code ASC "character" character variable
ASN	AS.	\sin^{-1}
ATN	AT.	\tan^{-1}
CHRS	CH. CHR.	Converts ASCII code to characters CHRS ASCII decimal code numeric value
COS	COS	
DEG		Converts decimal degrees to degrees, minutes, seconds
DMS	DM.	Converts degrees, minutes, seconds to decimal degrees
EXP	EX.	e^x
INKEYS	INK. INKE. INKEY.	Character variable = INKEYS If a key is pushed during execution of INKEYS command, the ASCII character will be read into the character variable.
INT		Truncates value to integer INT (10/3) ENTER < Display > 3
LEFTS	LEF. LEFT.	LEFT\$ (character variable, numeric expression) Takes the specified number of characters from the left side of the specified character string.
LEN		LEN "character" character variable Seeks the character count of the specified character string
LOG	LO.	$\log_{10} X$
LN		$\log_e X$
MEM	M. ME.	Displays the remaining number of steps available in memory. Same as STATUS 0.

Function	Abbreviations	Remarks
MIDS	MI. MID.	MIDS (character variable, numeric exp1, numeric exp2); Takes character (s) from the middle of the specified character string.
NOT	NO.	NOT exp. [logical Negation]
OR		exp. OR exp. [logical Or]
π (PI)		Specifies ratio of circumference (= 3.141592654)
POINT	POI. POIN.	Returns a number which represents the pattern of activated dots within the given column. POINT numeric expression
RIGHT\$	RI. RIG. RIGH. RIGHT.	RIGHT\$ (character variable, numeric exp) Takes the specified number of characters from the right side of the specified character string.
RND	RN.	RND expression Command to generate random numbers
SGN	SG.	Signum function
SIN	SI.	sine
$\sqrt{\quad}$ (SQR)	SQ.	square root
STATUS	STA. STAT. STATU.	STATUS 0 or 1 (0) Number of program steps available. (1) Number of program steps already used.
STR\$	STR.	STR\$ numeric expression Converts numerals to character string.
TAN	TA.	tan
TIME	TI. TIM.	(1) TIME = month, day, hour, minute, second Time function sets month, day, hour, minute, second (2) TIME [calls up date and time]
VAL (value)	V. VA.	VAL { "character" character variable } Converts character strings to numerals
^		Powers

2. Statements

Command	Abbreviation	Remarks
AREAD (auto read)	A. AR. ARE. AREA.	AREAD variable When executing programs by defined keys, AREAD enters display content into specified variables.
ARUN (auto run)	ARU.	ARUN Command to automatically start program execution when the PC1500 is switched on.
BEEP	B. BE. BEE.	BEEP exp1, exp2, exp3 Sound command. Turns sound generating functions ON/OFF, specifies loudness and length of sounds.
CLEAR	CL. CLE. CLEA.	Command to clear all data (variables)
CLS (clears)		Display clear command. Erases display.
CURSOR	CU. CUR. CURS. CURSO.	(1) CURSOR exp. ($0 \leq \text{exp} \leq 25$) specification of starting position of display. (2) CURSOR cancels previous specification.
DATA	DA. DAT.	DATA exp, exp, ... Data to be read in using the READ command
DEGREE	DE. DEG. DEGR. DEGRE.	Angular mode specification. Degree [$^{\circ}$] is designated.
DIM (dimension)	D. DI.	(1) DIM variable name (exp) (2) DIM variable name (exp) * exp3 (3) DIM variable name (exp1, exp2) variable names: A, B, C\$, D\$, etc ... (): specifies size and dimension of the array exp3: digit space specification
END	E. EN.	Program ending command
FOR TO STEP	F. FO. STE.	(1) For numerical variable = exp1 to exp2 Start of FOR-NEXT loop. Used in correspondence with NEXT command. (2) FOR numerical variable = exp1 to exp2 STEP exp3 exp1: initial expression exp2: final value exp3: interval to increase with each loop

Command	Abbreviation	Remarks
GCursor (graphic cursor)	GC. GCU. GCUR. GCURS. GCURSO.	Specifies display position by dot units.
		GCursor expression (0 ≤ expression ≤ 155) or (& 0 ≤ expression ≤ & 9B)
GOSUB	GOS. GOSU.	GOSUB { expression "character" character variable } Subroutine jump commands. Moves execution to specified line or label. Statements at this point are executed as a subroutine. Used in correspondence with RETURN command.
GOTO	G. GO. GOT.	Jump commands. GOTO { expression "character" character variable } Moves execution to specified line or label.
GPRINT (graphic print)	GP. GPR. GPRI. GPRIN.	Displays on printer content given on display.
		(1) GPRINT "OO OO OO ..." (inside " " are hexadecimal numbers)
		(2) GPRINT O; O; ... (3) GPRINT & O; & O; ...
GRAD	GR. GRA.	angular mode designation. grad ([⁹]) is designated.
IF		(1) IF conditional expression execution command. (2) IF arithmetic expression execution command. Evaluates given conditions and either moves execution to the next line or executes command.
INPUT	I. IN. INP. INPU.	(1) INPUT variable, variable, ... (2) INPUT "character", variable, "character", variable (3) INPUT "character"; variable, "character"; variable
LET	LE.	(1) LET numeric variable = expression (2) LET character variable = "characters" (3) LET character variable = character variable. LET follows IF commands. It can be omitted in other cases.
LOCK	LOC.	Locks the MODE the computer is in.
NEXT	N. NE. NEX.	NEXT numeric variable Shows the very end of the FOR-NEXT loop. The numeric variable must be the same as the numeric variable following the FOR command.

Command	Abbreviation	Remarks
ON ERROR	O. ER. ERR. ERRO.	ON ERROR GOTO expression Error trapping command.
ON GOSUB	O. GOS. GOSU.	ON expression GOSUB exp 1, exp2, exp3 Subroutine jump command. Specifies the place to jump to (exp 1, exp2, exp3) by the value of expression.
ON GOTO	O. G. GO. GOT.	On expression GOTO exp 1, exp2, exp3 Jump command. Specifies the place to jump to (exp 1, exp2, exp3) by the value of the expression following ON.
PAUSE	PA. PAU. PAUS.	Same form as PRINT command. Displays the specified content for about 0.85 seconds, then executes program.
POINT	POI. POIN.	POINT expression (0 ≤ expression ≤ 155) (& 0 ≤ expression ≤ & 9 B) Reads out dot pattern of information displayed at specified point. ex. A = POINT 56
PRINT	P. PR. PRI. PRIN.	(1) PRINT { expression "character" character variable } ; (2) PRINT { expression "character" character variable } , { expression "character" character variable } (3) PRINT { expression "character" character variable } ; { expression "character" character variable } ; ... ; { expression "character" character variable } ;
RADIAN	RAD. RADI. RADIA.	Angular mode designation. Radian ([rad]) is designated.
RANDOM	RA. RAN. RAND. RANDO.	Place the seed of random numbers prior to the use of RND commands.
READ	REA.	READ variable, variable, . . . Data read in command. Enters data from DATA statements into specified variables.
REM (remark)		REM . . . document notes . . . Specify remarks not to be executed.

Command	Abbreviation	Remarks
RESTORE	RES. REST. RESTO. RESTOR.	(1) RESTORE expression Changes the order of data read in by READ commands (2) RESTORE start at beginning of first DATA statement
RETURN	RE. RET. RETU. RETUR.	Return to continue execution after GOSUB statement that invoked this subroutine.
STOP	S. ST. STO.	Command to halt program execution.
THEN	T. TH. THE.	Defines execution command for IF statement. Jump commands are only possible to be defined as execution commands for IF command. THEN { expression "character" character variable }
TRON (trace on)	TR. TRO.	Specifies the mode to perform debugging.
TROFF (trace off)	TROF.	Cancel the mode for performing debugging.
UNLOCK	UN. UNL. UNLO. UNLOC.	Cancel LOCK mode.
USING	U. US. USI. USIN.	(1) USING "###.###^" (2) USING "&&& &&&" (3) PRINT USING "Format"; (4) USING (5) PRINT USING; Specifies display position by the value of the expression.
WAIT	W. WA. WAI.	WAIT expression ($0 \leq \text{expression} \leq 65535$) Specifies time duration of display when using PRINT commands. WAIT with no argument cancels the previous specification (duration = infinity)

3. Commands

Command	Abbreviation	Remarks
CONT (continue)	C. CO. CON.	Restarts execution of programs that have been temporarily halted. Effective in the RUN mode.
LIST	L. LI. LIS.	Command that performs program listing. Effective in the PRO mode.
NEW		(1) NEW (2) NEW 0 In the PRO mode, clears the program of all variables.
RUN	R. RU.	(1) RUN (2) RUN expression (3) RUN "character" character variable Effective in RUN mode.

4. Cassette Commands

Command	Abbreviation	Remarks
CHAIN	CHA. CHAI.	Transmission commands. When used in the middle of a program, reads in programs from the tape (transmits) and executes the program. (1) CHAIN "filename" (CHAIN-1 "filename") (2) CHAIN "filename", expression (CHAIN-1 "filename", expression) (3) abbreviated forms of the "filename" for (1) and (2).
CLOAD (cassette load)	CLO. CLOA.	Transmission command. This command transmits program or reserve content from the tape to the computer memory. (1) CLOAD (CLOAD-1) (2) CLOAD "filename" (CLOAD-1 "filename")

Command	Abbreviation	Remarks
CLOAD? (cassette load?)	CLO.? CLOA.?	Comparison command. This command compares the program in the memory or the reserve content with the content recorded on tape. (1) CLOAD? (CLOAD?-1) (2) CLOAD? "filename" (CLOAD?-1 "filename")
CSAVE (cassette save)	CS. CSA. CSAV.	This command records on tape the content of program and reserve memory. (1) CSAVE (CSAVE-1) (2) CSAVE "filename" (CSAVE-1 "filename")
INPUT #	I. # IN. # INP. # INPU. #	Data transmission command. This command transmits data recorded on tape into the specified variables. Takes same form as PRINT # command.
MERGE	MER. MERG.	Transmission command. This command transmits programs from tape to the computer. Takes same form as CLOAD command. In this command, previously recorded programs will be retained as they are and programs newly read in will be added.
PRINT #	P. # PR. # PRI. # PRIN. #	Data recording command. This command records onto tape the data stored in the PC-1500. (1) PRINT # variable name, variable name, . . . (PRINT # -1, variable name, variable name) (2) PRINT # "filename"; variable name, . . . (PRINT #-1, "filename"; variable name, . . .)
RMT OFF (remote off)	RM. OF. RMT OF.	This command cancels the remote function of REM 1 terminal. (For second tape recorder)
RMT ON (remote on)	RM. O. RMT O.	This command resets the remote function of REM 1 terminal. (For second tape recorder)

5. Printer Commands

Command	Abbreviation	Remarks
COLOR	COL. COLO.	Specifies color of characters. COLOR expression (0 <= expression <= 3)
CSIZE (character size)	CSI. CSIZ.	Specifies size of the characters to be printer. CSIZE expression (1 <= expression <= 9)

Command	Abbreviation	Remarks
GLCURSOR (graphic line cursor)	GL. GLC. GLCU. GLCUR. GLCURS. GLCURSO.	Command that moves the pen position from the starting point to an X, Y coordinate. Valid for GRAPH mode only. GLCURSOR (exp 1, exp2)
GRAPH (graphic)	GRAP.	This mode is used to draw graphs and illustrations.
LCURSOR (line cursor)	LCU. LCUR. LCURS. LCURSO.	Moves pen to desired position on printer.
LF (line feed)		Performs paper feed as far as number of line feeds shown by the expression. Valid for TEXT mode only. LF expression
LINE	LIN.	Line drawing commands. Valid for GRAPH mode only. (1) LINE (exp1, exp2) – (exp3, exp4) (2) LINE (exp1, exp2) – (exp3, exp4), exp5, exp6 (3) LINE (exp1, exp2) – (exp3, exp4), exp5, exp6, B exp5: specifies line type exp6: specifies color B: specifies a box drawing (4) LINE (exp1, exp2) – (exp3, exp4) – (exp11, exp12)
LLIST	LL. LLI. LLIS.	List program.
LPRINT	LP. LPR. LPRI. LPRIN.	Prints specified content. Valid for text mode only. Takes same form as PRINT commands.
RLINE (relative line)	RL. RLI. RLIN.	Command that draws lines with the pen position as the starting point. Valid for GRAPH mode only. Takes same form as LINE commands.

Command	Abbreviation	Remarks
ROTATE (rotate)	RO. ROT. ROTA. ROTAT.	Specifies direction of characters to be printed (print direction). Valid for GRAPH mode only. ROTATE expression ($0 \leq \text{expression} \leq 3$)
SORGN (set origin)	SO. SOR. SORG.	This command specifies the present pen position as the new starting point (point of origin). Valid for GRAPH mode only.
TAB		Specifies pen position. Valid for TEXT mode only. (1) TAB Expression (2) LPRINT TAB Expression; . . .
TEST	TE. TES.	Color check. When executed, test draws a 5 mm by 5mm square in each color.
TEXT	TEX.	TEXT mode specification. This mode prints characters and numerals.



END

Command	Abbreviation	Remarks
GLCURSOR (graphic line cursor)	GL. GLC. GLCU. GLCUR. GLCURS. GLCURSO.	Command that moves the pen position from the starting point to an X, Y coordinate. Valid for GRAPH mode only. GLCURSOR (exp 1, exp 2)
GRAPH (graphic)	GRAP.	This mode is used to draw graphs and illustrations.
LCURSOR (line cursor)	LCU. LCUR. LCURS. LCURSO.	Moves pen to desired position on printer.
LF (line feed)		Performs paper feed as far as number of line feeds shown by the expression. Valid for TEXT mode only. LF expression
LINE	LIN.	Line drawing commands. Valid for GRAPH mode only. (1) LINE (exp1, exp2) – (exp3, exp4) (2) LINE (exp1, exp2) – (exp3, exp4), exp5, exp6 (3) LINE (exp1, exp2) – (exp3, exp4), exp5, exp6, B exp5: specifies line type exp6: specifies color B: specifies a box drawing (4) LINE (exp1, exp2) – (exp3, exp4) – (exp11, exp12)
LLIST	LL. LLI. LLIS.	List program.
LPRINT	LP. LPR. LPRI. LPRIN.	Prints specified content. Valid for text mode only. Takes same form as PRINT commands.
RLINE (relative line)	RL. RLI. RLIN.	Command that draws lines with the pen position as the starting point. Valid for GRAPH mode only. Takes same form as LINE commands.



All and more about Sharp PC-1500 at <http://www.PC-1500.info>

SHARP CORPORATION

OSAKA, JAPAN

Do not sale this PDF !!!

Printed in Japan
2A9.7T(TINSE3483CCZZ)②