TANDY®

Pocket Computer PC-8

OPERATION MANUAL

CAT. NO. 26-3655

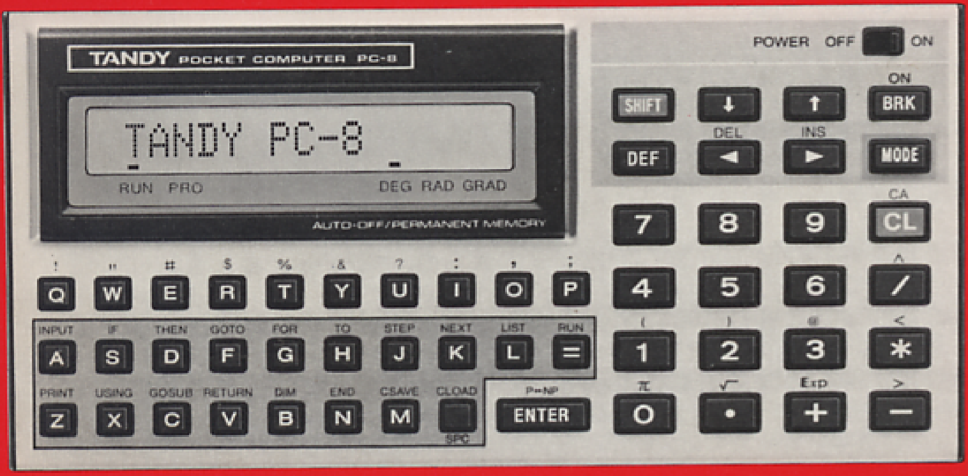


TABLE OF CONTENTS

Page

APPENDICES

INTRODUCTORY NOTE

Welcome to the world of **Tandy** computers!

Few industries in the world today can match the rapid growth and technological advances being made in the field of personal computing. Computers which just a short time ago would have filled a room, required a Ph. D. to program, and cost thousands of dollars, now fit in the palm of your hand, are easily programmed, and cost so little that they are within the reach of nearly everyone.

Your new **Tandy PC-8** was designed to bring you all of the latest state-of-the-art features of this computing revolution. As one of the most sophisticated hand-held computers in the world today. It incorporates many advanced capabilities:

- MEMORY SAFE GUARD – the **PC-8** remembers stored programs and variables, even when you turn it off.
- Battery powered operation for true portability.
- AUTO POWER OFF function conserves the batteries by turning the power off if no activity takes place within a specified time limit.
- Programmable functions which allow the **PC-8** to be used as a "smart" calculator.
- An expanded version of BASIC which provides formatted output, two-dimensional arrays, variable length strings, program chaining and many other advanced features.
- An optional Printer/Cassette Interface (Cat. No. 26-3591) for long-term storage and hard-copy printout of programs and data.

Congratulations on entering an exciting and enjoyable new world. We are sure that you will find this purchase one of the wisest you have ever made. The **Tandy PC-8** is a powerful tool, designed to meet your specific mathematical, scientific, engineering, business and personal computing needs. With the **Tandy PC-8** you can begin NOW providing the solutions you'll need tomorrow!

CHAPTER 1
HOW TO USE THIS MANUAL

This manual is designed to introduce you to the capabilities and features of your **PC-8** and to serve as a valuable reference tool. Whether you are a "first time user" or an "old hand" with computers, first acquaint yourself with the **PC-8** by reading and working through Chapters 2 through 6.

- Chapter 2 describes the physical features of the **PC-8**.
- Chapter 3 demonstrates the use of the **PC-8** as a calculator.
- Chapter 4 defines some terms and concepts which are essential for BASIC programming, and tells you about the special considerations of these concepts on the **PC-8**.
- Chapter 5 introduces you to BASIC programming on the **PC-8**, showing you how to enter, correct, and run programs.
- Chapter 6 discusses some shortcuts that make using your new computer easier and more enjoyable.

Experienced BASIC programmers may then read through Chapter 8 to learn the specific features of BASIC as implemented on the **PC-8**. Since every dialect of BASIC is somewhat different, read through this material at least once before starting serious programming.

Chapter 8 is a reference section covering all the verbs, commands, and functions of BASIC arranged in convenient alphabetical groupings.

If you have never programmed in BASIC before, we suggest that you buy a separate book on beginning BASIC programming or attend a BASIC class, before trying to work through these chapters. This manual is not intended to teach you how to program.

The remainder of the manual consists of:

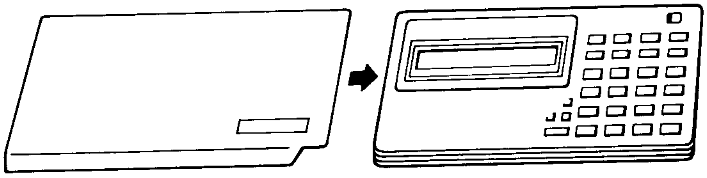
- Chapter 7 — Basic information on the optional Printer/Cassette Interface (Cat. No. 26-3591).
- Chapter 9 — A troubleshooting guide to help you solve some operating and programming problems.
- Chapter 10 — The care and maintenance of your new computer.

Detailed Appendices, at the end of the manual, provide you with useful charts, comparisons, and special discussions concerning the use and operation of the **PC-8.**

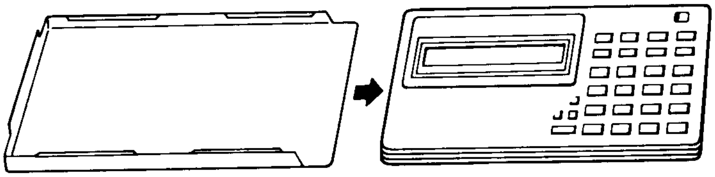
Using the Hard Cover

When the computer is not being used, mount the hard cover over the keyboard of the computer.

- When the computer is not in use.



- When the computer is in use.



CHAPTER 2
INTRODUCTION TO THE PC-8

Description of System

The **Tandy PC-8** system consists of:

- 53-character keyboard.
- 16-character display.
- Powerful BASIC in 17.4KB ROM.
- 4-bit CMOS processor.
- 2KB RAM
- Option: Printer/Cassette Interface.

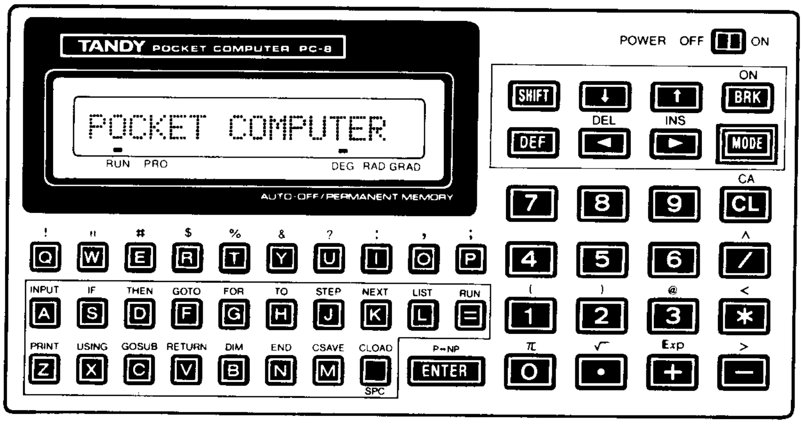


Figure 1.

Now study each section of the keyboard to familiarize yourself with the location and functions of specific parts of the **PC-8** keyboard. For now just locate the keys and read the description of each. Chapter 3 will teach you how to begin operation.

Description of Keys

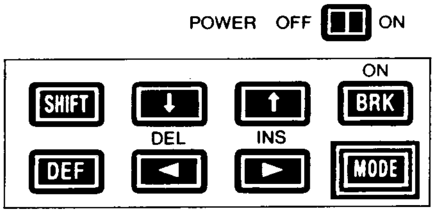


Figure 2.

POWER OFF ▥ ON Use to turn the **PC-8** ON and OFF.

SHIFT Press before pressing a key to enter the letter/command shown in brown on the **PC-8** keyboard. Not used to capitalize letters as all alphabet keys on the **PC-8** are in the upper case.

↓ Press to display the next program line.

↑ Press to display the previous program line.

ON BRK Temporarily interrupts a program being executed. Press after an auto power off to turn the computer back on.

DEL ◄ Allows you to move the cursor to the left without erasing characters. Press SHIFT before pressing this key to DELete the character the cursor is "on top of".

INS ► Allows you to move the cursor to the right without erasing characters. Press SHIFT before pressing this key to make a space directly before the character the cursor is "on top of". You can then INSert a new character into this space.

MODE Press to change the mode from RUN to PROgram or from PRO to RUN.

DEF A special key used to execute BASIC programs.

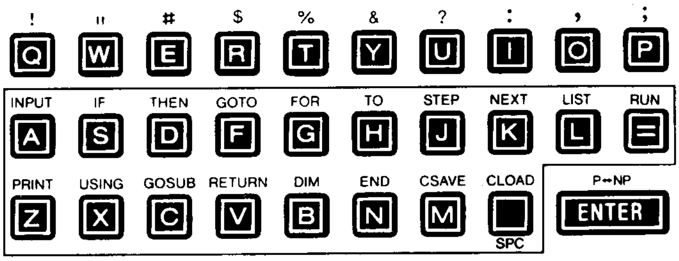

! " # $ % & ? : , ;
Q W E R T Y U I O P
INPUT IF THEN GOTO FOR TO STEP NEXT LIST RUN
A S D F G H J K L =
PRINT USING GOSUB RETURN DIM END CSAVE CLOAD
Z X C V B N M
SPC
P↔NP
ENTER

Figure 3.

A ~ Z Press to enter the characters as you would on a standard typewriter. On the **PC-8** display, the characters always appear in the upper case.

= On the **PC-8**, this key is **not** used to indicate the end of a calculation; in BASIC programming, this symbol has a special function.

SPC Press to insert a blank space.

P←→NP
ENTER When you press this key, whatever you previously typed is "entered" into the computer's memory. You must press ENTER before the **PC-8** will act on input from the keyboard. Press SHIFT before pressing this key to switch on and off the printing on the printer.

! " #
$ % &
? : ,
;

These symbols are found above the top row of alphabet keys. Press SHIFT and then the alphabet key under the character to display these symbols.

INPUT
IF
≀
CLOAD

Preset command and statement keys. Press SHIFT and then the alphabet (including equals and space) key under the command or statement to enter the designated command or statement to the **PC-8**.

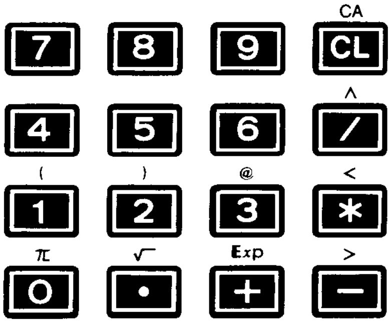


Figure 4.

Ø ~ 9 • Use to enter numerical values.

(1 Press SHIFT and then this key to enter open parenthesis.

) 2 Press SHIFT and then this key to enter close parenthesis.

CA CL Press to erase the characters you have just typed in and release errors. Press SHIFT before pressing this key to activate the CA (reset) function. CA clears the display and resets the computer.

^ / Press to include the division operator in calculations. Press SHIFT and then this key to display the "power" symbol, indicating that a number is to be raised to a specific power.

< * Press to include the multiplication operator in calculations. Press SHIFT and then this key to display the "less than" symbol.

> − Press to include the subtraction operator in calculations. Press SHIFT and then this key to display the "greater than" symbol.

Exp + Press to include the addition operator in calculations. Press SHIFT and then this key to display the exponentiation symbol used in scientific notation.

π 0 Press SHIFT and then this key to enter the π (ratio of circumference to diameter of the circle).

√ • Press SHIFT and then this key to enter the square root operator.

@ 3 Press SHIFT and then this key to enter the @ character.

Description of Display

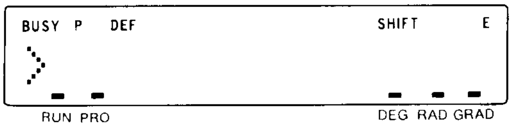


Figure 5.

The liquid crystal display of the **Tandy PC-8** shows up to 16 characters at one time. Although you may input up to 8Ø characters including ENTER in one line, only the 16 characters are displayed. To review the remaining characters in a line, move the cursor to either end; the display will 'scroll' – that is, as characters drop off the display, new characters appear on the opposite end.

The display consists of:

> The Prompt. This symbol appears when the computer is awaiting input. As you type, the prompt disappears and is replaced by the cursor.

_ The Cursor. This symbol (the underline) tells you the location of the next character to be typed in. As you begin typing, the cursor replaces the prompt. Move the cursor with ◄ or ► key to position it over certain characters when you want to INSert and DELete.

▬ **RUN** RUN Indicator. Tells you the **PC-8** is in the RUN mode.

▬ **PRO** PROgram Indicator. Tells you the **PC-8** is in the PROgramming mode.

BUSY Program Execution Indicator. Lights when the **PC-8** is executing a program (except when characters are displayed). The **PC-8** will not undergo auto power off while the BUSY indicator is on. BUSY disappears from the display when execution is completed.

P Printer Indicator. Appears when you select the print option when using the **PC-8** as a calculator.

DEF Definable Mode Indicator. Lights up when you press the DEF key.

▬ **DEG RAD GRAD** Angular Measurement Indicator. Displays the current unit of angles for the input of trigonometric functions. Depending on the mode in use the indicator DEG (degrees), RAD (radians), or GRAD (gradients) will appear.

SHIFT Shift Key Indicator. Lights up when the SHIFT key has been pressed. Remember, the SHIFT key must be released before pressing any other key.

E Error Indicator. Appears when an error is encountered. Reset with the CL key.

RESET Button

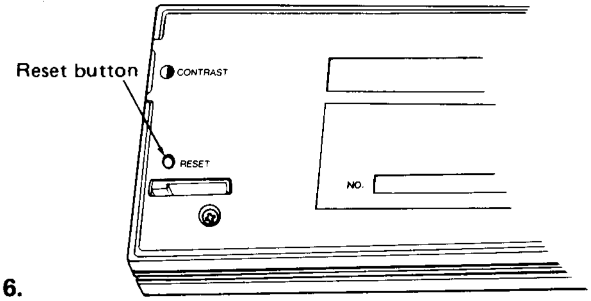
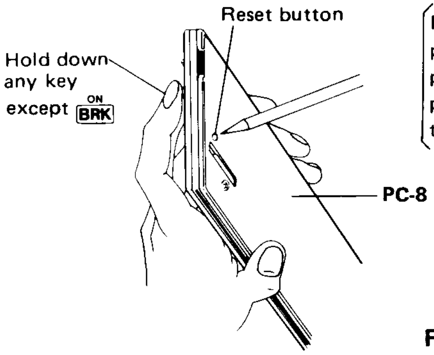


Figure 6.

Use the RESET button to reset the computer when CLear or CA is not sufficient to correct the problem.

To reset the **PC-8**, set the power switch to the ON position and press the RESET button on the back while holding down any key except $\overset{\text{ON}}{\boxed{\text{BRK}}}$. This preserves all programs and variables.

Note: Never press the RESET button while holding down $\overset{\text{ON}}{\boxed{\text{BRK}}}$. This operation may not preserve the reserved contents.



(Press the RESET button 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.)

Figure 7.

If you get no response from any key even after the above operation, press the RESET button and do the following:

1 Set the **PC-8** to the PROgram mode with the $\boxed{\text{MODE}}$ key.

2 Enter NEWØ and press $\boxed{\text{ENTER}}$.

This operation clears all the contents reserved in the **PC-8**. Re-enter the program.

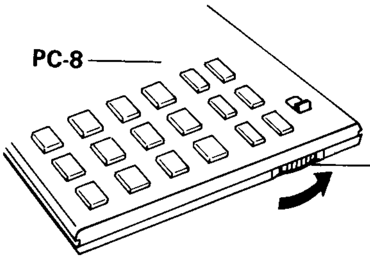


Figure 8.

Turn the control in the direction of the arrow to darken the display, and turn it in the opposite direction to lighten the display.
Adjust the control so that the display is easy to see.

Battery Replacement

The **PC-8** operates on two lithium batteries. When connected to the Printer/Cassette Interface, the **PC-8** power can also be supplied from the Printer/Cassette Interface if it has enough power voltage. This minimizes the power consumption of the lithium batteries.
When replacing the batteries, take the following precautions to eliminate many problems:

- Always replace both batteries at the same time.
- Do not mix a new battery with a used battery.
- Use only: Lithium battery (type CR-2032 Cat. No. 23-162)

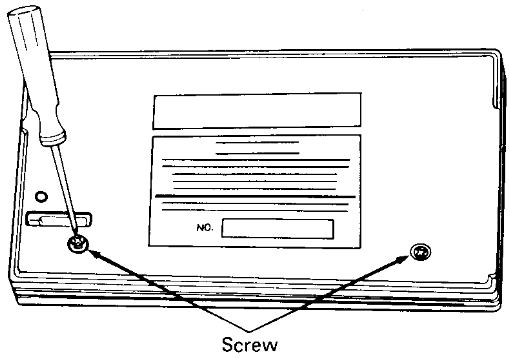
INSTALLING THE BATTERIES

The display may be dim and difficult to see, even after turning the contrast control fully counterclockwise. This indicates that the battery power is consumed. Replace the batteries promptly. Remember that the computer's memory is cleared when the batteries are removed. Use the optional Printer/Cassette Interface to save programs and data on tape.

(1) Turn off the computer by setting the POWER switch to OFF.

(2) Remove the screws from the back cover with a small screw driver. (Fig. 9)

Figure 9.



(3) Remove the battery cover by sliding it in the direction of the arrow shown in Figure 10.

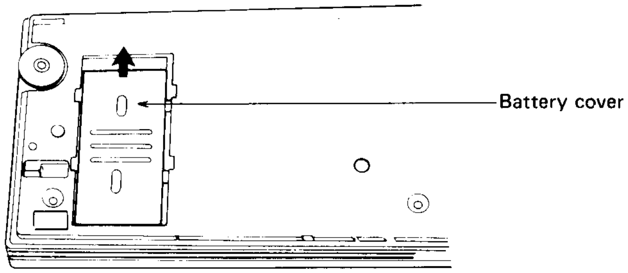


Figure 10.

(4) Replace the two batteries. (Fig. 11)'

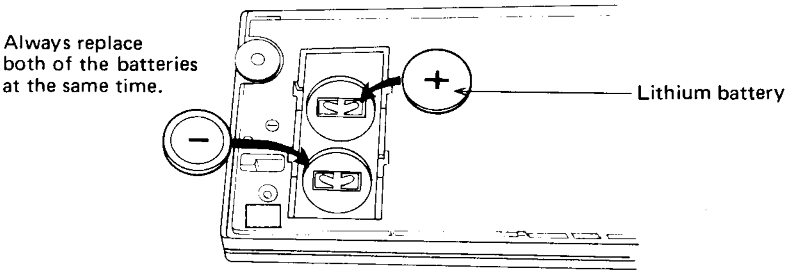


Figure 11.

(5) Replace the battery cover by sliding it in the reverse direction of the arrow shown in Figure 10.

(6) Hook the claws of the back cover into the slits of the computer body. (Fig. 12)

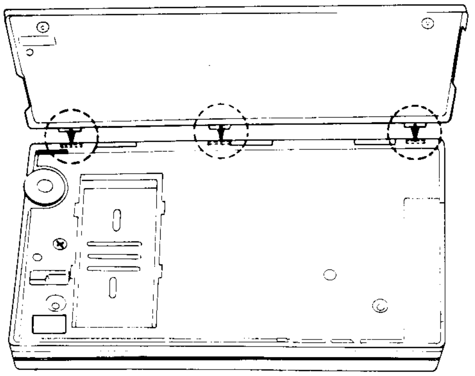


Figure 12.

(7) Push the back cover in slightly while replacing the screws.

(8) Turn on the **PC-8** by setting the power slide switch to the ON position and press the RESET button to clear the **PC-8**.
Change the mode to PRO.
Enter NEWØ and press [ENTER] .
The display should look like this:

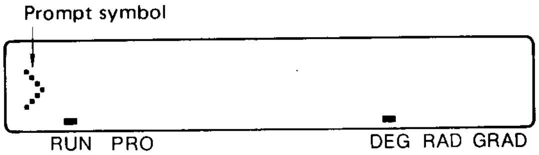


Figure 13.

If the display is blank or displays any other symbol than the prompt > remove the batteries and install them again, then check the display.

NOTE:

Keeping a dead battery in the computer may result in damage to the computer from chemical leakage of the battery. Remove a dead battery promptly.

CAUTION: Keep batteries out of the reach of children.

CHAPTER 3
USING THE PC-8 AS A CALCULATOR

Now that you are familiar with the layout and components of the **Tandy PC-8**, we will begin investigating the exciting capabilities of your new computer.

Because the **PC-8** allows you the full range of calculating functions, plus the increased power of BASIC programming abilities (useful in more complex calculations), it is commonly referred to as a "smart" calculator.

Before using the **PC-8**, be sure that the batteries are correctly installed.

Start Up

To turn ON the **PC-8**, slide the power switch to the ON position. Confirm that the mode indicator (▬) is positioned above the label RUN; if not, press the MODE key. For use as a calculator, the **PC-8** must be in the RUN mode. When the machine is ON, the PROMPT (>) will appear on the display.

Shut Down

To turn OFF the **PC-8**, slide the power switch to the OFF position.

When you turn OFF the machine, you clear (erase) the display. However, the **PC-8** does remember all programs and reserved contents. All of these contents are still in effect when the machine is turned back ON.
When the CLOAD command is executed, stop the execution by pressing the BRK key before sliding the power switch to the OFF position.

Auto Off

To save battery wear, the **PC-8** automatically shuts off when no keys have been pressed for about 11 minutes. (Note: The **PC-8** will not shut off while you are executing a program.)

To restart the computer after an auto off, press the ON BRK key. All settings will be exactly as they were when the AUTO OFF occurred.

Some Helpful Hints

Until you are used to your new machine, you are bound to make mistakes. Later we will discuss some simple ways to correct these mistakes. For now, if you get an Error Message, press CLear and retype the entry. If the computer "hangs up" – you cannot get it to respond at all – press the RESET button (See Chapter 2).
The PROMPT (>) tells you that the **PC-8** is awaiting input. As you enter data the PROMPT disappears and the cursor (–) moves to the right, indicating the next available location in the display.

The right ▶ and left ◀ arrows move the cursor within a line. When you move the cursor over a character, it changes to a flashing solid block (▒).

ENTER informs the **PC-8** that you finished entering data and signals the computer to perform the indicated operations. **YOU MUST PRESS ENTER AT THE END OF EACH LINE OF INPUT, OR YOUR CALCULATIONS WILL NOT BE PROCESSED BY THE PC-8.**

When performing numeric calculations, input appears on the left of the screen: the results appear on the right of the display.

When using the SHIFT key in conjunction with another key (to enter square root for example) press SHIFT , release the SHIFT , then press the other key. SHIFT is active for only one key at a time.

Do not use dollar signs or commas when entering calculations into the **PC-8**. These characters have special meaning in the BASIC programming language.

In this manual we use Ø to indicate zero, so that you can distinguish between the number (Ø) and the letter (O).

To help get you started entering data correctly, we will show you each keystroke in the example calculations. When SHIFT is used, we will indicate the desired character in the following keystroke. For example pressing SHIFT and 1 will produce the (character. These keystrokes are written as SHIFT (.

Be sure to enter CLear after each calculation (unless you are performing chain calculations). CLear erases the display and resets the error condition. It does not erase anything stored in the computer's memory.

Key Operation

Now let's operate the keys. Set the **PC-8** to the RUN mode and press the following keys while watching the display:

(Example)

```
PRINT USING GOSUB
 Z     X     C              →  ZXC_

 (     )     √     @
 1     2     •     3        →  ZXC12.3_

 CA
 CL                         →  >

INPUT  RUN        Exp
 A     =     4     +    5   →  A=4+5_
                                     ↑
                                   Cursor
```

When you press an alphabet or number key, the item written on the key will be entered. When you wish to enter the character or symbol written above a key, press SHIFT before operating the key.

(Example)

```
┌Clears the display.
↓
CA         PRINT
CL  SHIFT   Z                       →  PRINT_

      "           (           √
SHIFT W     SHIFT 1     SHIFT •     →  PRINT"(√_
```

The SHIFT key is used to enter the characters or symbols labeled in brown above the keys that have dual functions. If you repeatedly press the SHIFT key, the SHIFT symbol in the top right of the display will go on and off. The SHIFT symbol indicates that the SHIFT key is activated and the characters labeled in brown can be entered.

Simple Calculations

The **PC-8** performs calculations with ten-digit precision. Set the **PC-8** to the RUN mode. Now try these simple arithmetic examples. Remember to CLear between calculations.

Input	Display
[5] [Ø] [+] [5] [Ø] [ENTER]	1ØØ.
[1] [Ø] [Ø] [−] [5] [Ø] [ENTER]	5Ø.
[6] [Ø] [*] [1] [Ø] [ENTER]	6ØØ.
[3] [Ø] [Ø] [/] [5] [ENTER]	6Ø.
[1] [Ø] [SHIFT] [^] [2] [ENTER]	1ØØ.
[2] [*] [SHIFT] [π] [ENTER]	6.2831853Ø7
[SHIFT] [√] [6] [4] [ENTER]	8.

Recalling Entries

Even after the **PC-8** has displayed the results of your calculation, you can redisplay your last entry (unless you pressed CLear). To recall, use the left [◄] or right [►] arrow.

The left arrow [◄] recalls the expression with the cursor positioned after the **last** character.

The right arrow [►] recalls the expression with the cursor positioned on top of the **first** character.

Remember that the left and right arrows are also used to position the cursor along a line. The right and left arrows are very helpful in editing (or modifying) entries without having to retype the entire expression.

You will become familiar with the use of the right and left arrows in the following examples. Now, take the role of the manager and perform the calculations as we discuss them.

As the head of personnel in a large marketing division, you are responsible for planning the annual sales meeting. You expect 3ØØ people to attend the three day conference. For part of this time, the sales force will meet in small groups. You believe that groups of six would be a good size. How many groups would this be?

Input	Display
[3] [Ø] [Ø] [/] [6] [ENTER]	5Ø.

On second thought you decide that groups containing an odd number of participants might be more effective. Recall your last entry using the ◄ arrow.

Input	Display
◄	3ØØ/6_

To calculate the new number of groups you must replace the six with an odd number. Five seems to make more sense than seven. Because you recalled using the ◄ arrow, the cursor is positioned at the end of the display. Use the ◄ to move the cursor one space to the left.

Input	Display
◄	3ØØ/▒

Notice that after you move the cursor, it becomes a flashing block ▒ . Whenever you position the cursor on top of an existing character, it will be displayed as the flashing cursor.

Type in a 5 to replace the 6. One caution in replacing characters – once you type a new character over an existing character, the original is gone forever! You cannot recall an expression that has been typed over.

Input	Display
5	3ØØ/5_
ENTER	6Ø.

Sixty seems like a reasonable number of groups, so you decide that each small group will consist of five participants.

Recall is also useful to verify your last entry, especially when your results do not make sense. For instance, suppose you had performed this calculation:

Input	Display
3 Ø / 5 ENTER	6.

Even a tired, overworked manager like you realizes that 6 does not seem to be a reasonable result when you are dealing with hundreds of people! Recall your entry using the [▶].

Input	Display
[▶]	3Ø/5

Because you recalled using the [▶] the flashing cursor is now positioned over the first character in the display. To correct this entry you wish to insert another zero. Using the [▶], move the cursor until it is positioned over the zero. When making an INSert, you position the flashing cursor over the character **before** which you wish to make the insertion.

Input	Display
[▶]	3Ø/5

Use the INSert key to make space for the needed character.

Input	Display
[SHIFT] [INS]	3⊔Ø/5

Pressing INSert moves all the characters one space to the right, and inserts a bracketed open slot. The flashing cursor is now positioned over this open space, indicating the location of the next typed input. Type in your zero. Once the entry is corrected, display your new result.

Input	Display
[Ø]	3ØØ/5
[ENTER]	6Ø.

On the other hand, suppose that you had entered this calculation:

Input	Display
[3] [Ø] [Ø] [Ø] [/] [5] [ENTER]	6ØØ.

The result seems much too large. If you only have 3ØØ people attending the meeting, how could you have 6ØØ "small groups"? Recall your entry using the [▶] .

Input	Display
[▶]	3ØØØ/5

The flashing cursor is now positioned over the first character in the display. To correct this entry eliminate one of the zeros. Using the [▶] move the cursor to the first zero (or any zero). When deleting a character, you position the cursor on top of the character to be deleted.

Input	Display
[▶]	3ØØØ/5

Now use the DELete key to get rid of one of the zeros.

Input	Display
[SHIFT] [DEL]	3ØØ/5

Pressing DELete causes all the characters to shift one space to the left. It deletes the character it is "on top of" and the space the character occupies. The flashing cursor stays in the same position indicating the next location for input. Since you have no other changes to make, complete the calculation.

Input	Display
[ENTER]	6Ø.

(Note: Pressing the SPaCe key, when it is positioned over a character, replaces the character leaving a blank space. DELete eliminates the character and the space it occupied.)

Errors

Recalling your last entry is essential when you get an ERROR message. Let us imagine that, unintentionally, you typed this entry into the **PC-8**:

Input	Display
[3] [Ø] [Ø] [/] [/] [5] [ENTER]	ERROR 1

Naturally you are surprised when this message appears! ERROR 1 is simply the computer's way of saying, "I don't know what you want me to do here". To find out what the problem is, recall your entry using either the [◄] or [►] key.

Input	Display
[◄] (or [►])	3ØØ//5

Whether you use the [◄] or [►] key, the flashing cursor indicates the point at which the computer got confused. And no wonder, you have too many operators! To correct this error use the DELete key.

Input	Display
[SHIFT] [DEL] [ENTER]	6Ø.

If, upon recalling your entry after an ERROR 1, you find that you have **omitted** a character, use the INSert sequence to correct it.

When using the **PC-8** as a calculator, the majority of the errors you encounter will be ERROR 1 (an error in syntax). For a complete listing of error messages, see APPENDIX A.

Chain Calculations

The **PC-8** allows you to use the results of one calculation as part of the following calculation.

Part of your responsibility in planning this conference is to draw up a detailed budget for approval. You know that your total budget is $15Ø.ØØ for each attendant. Figure your total budget:

Input	Display
[3] [Ø] [Ø] [*] [1] [5] [Ø] [ENTER]	45ØØØ.

Of this amount you plan to use 15% for the final night's awards presentation. When performing chain calculations it is not necessary to retype your previous results, but DO NOT CLear between entries. What is the awards budget?

Input	Display
* . 1 5	45ØØØ. *. 15_

Notice that as you type in the second calculation (* . 15), the computer automatically displays the result of your first calculation at the left of the screen and includes it in the new calculation. In chain calculations, the entry must begin with an operator. As always, you end the entry with ENTER :

NOTE: The % key cannot be used in the calculation. The % key should be used as a character only.

Example: 45ØØØ * 15 SHIFT % ENTER → ERROR 1

Input	Display
ENTER	675Ø.

Continue allocating your budget. The hotel will cater your dinner for $4ØØØ:

Input	Display
− 4 Ø Ø Ø	675Ø. −4ØØØ_
ENTER	275Ø.

Decorations will be $1225:

Input	Display
− 1 2 2 5 ENTER	1525.

Finally, you must allocate $22ØØ for the speaker and entertainment:

Input	Display
− 2 2 Ø Ø ENTER	−675.

Obviously, you will have to change either your plans or your allocation of resources!

Negative Numbers

Since you want the awards dinner to be really special, you decide to stay with the planned agenda and spend the additional money. However, you wonder what percentage of the total budget will be used up by this item. First, change the sign of the remaining sum:

Input	Display
[*] [-] [1]	-675.*-1_
[ENTER]	675.

Now you add this result to your original presentation budget:

Input	Display
[+] [6] [7] [5] [Ø] [ENTER]	7425.

Dividing by 45ØØØ gives you the percentage of the total budget this new figure represents:

Input	Display
[/] [4] [5] [Ø] [Ø] [Ø] [ENTER]	Ø.165

Fine, you decide to allocate 16.5% to the awards presentation.

Compound Calculations and Parentheses

In performing the above calculations, you could have combined several of these operations into one step. For instance, you might have typed both these operations on one line:

675+675Ø/45ØØØ

Compound calculations, however, must be entered very carefully:

675+675Ø/45ØØØ might be interpreted as

$$\frac{675 + 6750}{45000} \qquad \text{or} \qquad 675 + \frac{6750}{45000}$$

When performing compound calculations, the **PC-8** has specific rules of expression evaluation and operator priority (see APPENDIX D). Be sure you get the calculation you want by using parentheses to clarify your expressions:

(675+675Ø)/45ØØØ or 675+(675Ø/45ØØØ)

To illustrate the difference that the placement of parentheses can make, try these two examples:

Input	Display
SHIFT (6 7 5 + 6 7 5 Ø SHIFT) / 4 5 Ø Ø Ø ENTER	Ø.165
6 7 5 + SHIFT (6 7 5 Ø / 4 5 Ø Ø Ø SHIFT) ENTER	675. 15

Using Variables in Calculations

The **PC-8** can store up to 26 simple numeric **variables** under the alphabetic characters A to Z. If you are unfamiliar with the concept of variables, refer to Chapter 4. You assign a value to variables with an Assignment Statement:

A = 5
B = –2

You can also assign the value of one variable (right) to another variable (left):

C = A + 3
D = E

A variable may be used in place of a number in any calculation.

Now that you have planned your awards dinner, you need to complete arrangements for your conference. You wish to allocate the rest of your budget by percentages also. First you must find out how much money is still available. Assign a variable (R) to be the amount left from the total:

Input	Display
R = 4 5 Ø Ø Ø - 7 4 2 5	R=45ØØØ-7425_
ENTER	37575.

As you press ENTER the **PC-8** performs the calculation and displays the new value of R. You can display the current value of any variable by entering the character of the variable:

Input	Display
R ENTER	37575.

You can then perform calculations using your variable. The value of (R) will not change until you assign it a new value.

You wish to allocate 6Ø% of the remaining money to room rental:

Input	Display
R * . 6 Ø	R*. 6Ø_
ENTER	22545.

Similarly, you want to allocate 25% of your remaining budget to conduct management training seminars:

Input	Display
R * . 2 5 ENTER	9393. 75

Variables will retain their assigned values even if the machine is turned OFF – either manually or automatically. Variables are lost only when:

- ∗ You assign a new value to the same variable.
- ∗ You type in NEW ENTER or NEW Ø ENTER .
- ∗ You type in CLEAR ENTER (not the CLear key).
- ∗ The batteries are changed.

There are certain limitations on the assignment of variables, and certain programming procedures which cause them to be changed. See Chapter 4 for a discussion of assignment. See Chapter 5 for a discussion of the use of variables in programming.

Chained Calculations

In addition to combining several operators in one calculation, the **PC-8** also allows you to perform several calculations one after the other – without having to press [ENTER] before moving on. You must separate the equations with commas. Only the result of the **final** calculation is displayed. (Remember too, that the maximum line length accepted by the computer is 8Ø characters including [ENTER] .)

You wonder how much money would have been available for rooms if you had kept to your original allocation of 15% for the awards dinner:

Input	Display
[R] [=] [•] [8] [5] [*] [4] [5] [Ø] [Ø] [Ø] [SHIFT] [,] [R] [*] [•] [6] [Ø]	.85*45ØØØ,R*.6Ø–

Although the computer performs all the calculations in the chain, it displays only the final result:

Input	Display
[ENTER]	2295Ø.

To find the value of R used in this calculation, enter R:

Input	Display
[R] [ENTER]	3825Ø.

Now It's Your Turn

This concludes our discussion of using the **PC-8** as a calculator. Undoubtedly, as you become more familiar with your machine's capabilities and special features, you will find many new and useful applications for this "smart" calculator.

But calculating is only one of the many potential uses of the **PC-8**. In the next chapter we will examine the concepts and terms of the BASIC language, as it is used by the **PC-8**. Then you can begin to create your own, unique, problemsolving programs.

CHAPTER 4
CONCEPTS AND TERMS OF BASIC

In this chapter we will examine some concepts and terms of the BASIC language. Because the many features of BASIC can be used also when used as a calculator, some of these concepts are also useful for advanced calculator functions.

Numeric Constants

In Chapter 3 you entered simple numbers for use in calculations, without worrying about the different ways that numbers can be represented, or the range of numbers that the **PC-8** can process. Some of you, however, may need or desire to know more about how the **PC-8** uses numbers.

The **PC-8** recognizes three different ways to represent numbers:

∗ Decimals.
∗ Exponential or scientific notation.
∗ Hexadecimal numbers.

Decimal numbers are familiar to most of you. Scientific notation and hexadecimal numbers may require some explanation.

Scientific Notation

People who need to deal with very large and very small numbers often use a special format called exponential or **scientific notation.** In scientific notation a number is broken down into two parts.

The first part consists of a regular decimal number between 1 and 1Ø. The second part represents how large or small the number is in powers of 1Ø.

As you know, the first number to the left of the decimal point in a regular decimal number shows the unit of 1's, the second shows the unit of 1Ø's, the third the unit of 1ØØ's, and the fourth the unit of 1ØØØ's. These are simply increasing powers of 1Ø:

$1Ø^Ø = 1$, $1Ø^1 = 1Ø$, $1Ø^2 = 1ØØ$, $1Ø^3 = 1ØØØ$, etc.

Scientific notation breaks down a decimal number into two parts: the first part shows what the numbers are, the second part shows how far a number is to the left, or right, of the decimal point. For example:

1234 becomes 1.234 times 1Ø^{3} (3 places to the right)
654321 becomes 6.54321 times 1Ø^{5} (5 places to the right)
.ØØØ125 becomes 1.25 times 1Ø$^{-4}$ (4 places to the left)

Scientific notation is useful for many shortcuts. You can see that it would take a lot of writing to show 1.Ø times 1Ø^{87} – a 1 and 87 zeros! But, in scientific notation this number looks like this:

1.Ø x 1Ø^{87} or 1.Ø E 87

The **PC-8** uses scientific notation whenever numbers become too large to display using decimal notation. This computer uses a special exponentiation symbol, the E to mean "times ten to the":

1234567890ØØØ is displayed as 1.23456789 E 12
.ØØØØØØØØØØØ1 is displayed as 1. E –12

If you are unfamiliar with this type of notation, take some time to put in a few very large and very small numbers to note how they are displayed. By the way, if you want to enter a number in scientific notation, use SHIFT EXP key where you want E displayed.

Limits

The largest number which the **PC-8** can handle is ten significant digits, with two digit exponents. In other words the largest number is:

```
9.999999999 E 99 = 9999999999ØØØØØØØØØØØØØØØØØØØØØØØØ
                   ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
                   ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
```

and the smallest number is:

```
9.999999999 E –99 = .ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
                     ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
                     ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ9
                     999999999
```

Under certain circumstances, when numbers will be used frequently, the **PC-8** uses a special compact form. In these cases there are special limits imposed on the size of numbers, usually either Ø to 65535 or –32768 to +32767. Those with some computer background will recognize both these numbers as the largest range which can be represented in 16 binary bits. The circumstances in which this form is used are noted in the Chapter 8.

Hexadecimal Numbers

The decimal system is only one of many different systems to represent numbers. Another which has become quite important when using computers is the hexadecimal system. The hexadecimal system is based on 16 instead of 1Ø. To write hexadecimal numbers you use the familiar Ø ~ 9 and 6 more "digits": A, B, C, D, E, and F. These correspond to 1Ø, 11, 12, 13, 14, and 15. When you want the **PC-8** to treat a number as hexadecimal put an ampersand '&' character in front of the numeral:

```
&A    = 1Ø
&1Ø   = 16
&1ØØ  = 256
&FFFF = 65535
```

Those with some computer background may notice that the last number (65535) is the same as the largest number in the special group of limits discussed in the last paragraph. Hexadecimal notation is never required in using the **PC-8**, but there are special applications where it is convenient.

String Constants

In addition to numbers, there are many ways that the **PC-8** uses letters and special symbols. These letters, numbers, and special symbols are called characters. These characters are available on the **PC-8**:

```
1 2 3 4 5 6 7 8 9 Ø
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
! " # $ % & ( ) * + , − . / : ; < = > ? @ √ π ^ E
```

In BASIC a collection of characters is called a **string**. In order for the **PC-8** to tell the difference between a string and other parts of a program, such as verbs or variable names, you must enclose the characters of the string in quotation marks (").

The following are examples of string constants:

```
"HELLO"
"GOODBYE"
"PC-8"
```

The following are **not** valid string constants:

"COMPUTER	No ending quote
"ISN"T"	Quote can't be used within a string

Variables

In addition to constants, whose values do not change during a program, BASIC has **variables,** whose values can change. Variables are names used to designate locations where information is stored. These variables are like the letters used in algebraic equations. Just as there are numeric and string constants, there are numeric and string variables.

Simple Numeric Variables

You have already used **simple numeric variables** when working with the **PC-8** as a calculator in Chapter 3. Simple numeric variables are used to store a single number and are designated by a single letter (A – Z):

A = 5
C = 12.345

Simple numeric variables may take the same range of values as numeric constants.

Simple String Variables

String variables are used to hold strings (a collection of characters). They are named by a single letter followed by a dollar sign:

A$ = "ABCD"
C$ = "HELLO!"

NOTE: Strings must be put between the quotation marks.

A string variable may be from Ø to 7 characters long. If you try to store more than 7 characters in a string variable, only the first 7 will be saved. When a string variable is empty, or its length is zero, it is called NUL or the NUL string.

Numeric Array Variables

For some purposes it is useful to deal with numbers as an organized group, such as a list of scores or a tax table. In BASIC these groups are called **arrays.** An array can be either **one-dimensional,** like a list, or **two-dimensional,** like a table. Array names are designated in the same manner as simple variable names, except that they are followed by parentheses. The elements of an array are referred to by a number inside the parentheses; when the array is two dimensional there must be two numbers separated by a comma:

A(5) The fifth element of a one-dimensional array A
B (3,2) The element in the third row and second column of a two dimensional B array.

Arrays are created using the DIM verb or command. To create an array you give its name and its size:

```
DIM X(5)
DIM Y(32)
```

Note that DIM X(5) actually creates an array with six entries:

```
X(Ø)   X(1)   X(2)   X(3)   X(4)   X(5)
```

Similarly DIM Y(2, 2) creates an extra Ø row and a extra Ø column:

```
Y(Ø,Ø)   Y(Ø,1)   Y(Ø,2)
Y(1,Ø)   Y(1,1)   Y(1,2)
Y(2,Ø)   Y(2,1)   Y(2,2)
```

This extra element, or row and column, is often used by programmers to hold partial products during computations. For example, you might total the elements of the X array by summing them into X(Ø).

The form and use of the DIM verb is covered in detail in Chapter 8.

Note: The A array **does not** have the extra Ø element and does not need to be DIMensioned (see section below on Preallocated Variables).

String Array Variables

String array variables have the same relationship to numeric array variables as simple string variables have to simple numeric variables, – their names are the same except for the addition of a dollar sign:

C$(5) The fifth string element in the array C$

With string arrays the length of each string will be 16 characters unless you specifically choose a different length in the DIM statement:

DIM X$(12) * 8 DIMensions a string array with 13 elements, each a string 8 characters long

Chapter 8 details the use of the DIM statement.

Preallocated Variables

Some of the variables which you will use most frequently have already been allocated space in the **PC-8**'s memory. Twenty-six locations are reserved for numeric variables A – Z, string variables A$ – Z$, numeric array A(26) **OR** string array A$(26). The locations are assigned as follows:

Loc.	Num. Var.	Str. Var.	Num. Arr. Var.	Str. Arr. Var.
1	A	A$	A(1)	A$(1)
2	B	B$	A(2)	A$(2)
3	C	C$	A(3)	A$(3)
4	D	D$	A(4)	A$(4)
⋮	⋮	⋮	⋮	⋮
23	W	W$	A(23)	A$(23)
24	X	X$	A(24)	A$(24)
25	Y	Y$	A(25)	A$(25)
26	Z	Z$	A(26)	A$(26)

NOTE: There are only twenty-six locations and you must be careful not to use the same location in two different ways.

If you use location 24 to store a numeric value in X and then try to print X$, you will get an ERROR 9. Similarly, if you store a number in A(24) and then store another number in X you will over-write the first number, but you will not get an error message.

The A( ) and A$( ) arrays are different from all other arrays – they don't have a zero element. It is possible to use DIM to make A( ) or A$ ( ) larger than 26, but if you do, the first 26 elements will use the reserved locations while the elements from 26 on will be stored in a different part of the memory. The only way that you will notice this, however, is that these 26 special locations are not cleared when you RUN a program. All other array variables are cleared with each new RUN. By using good programming practice and always initializing your variables to the desired value, you will avoid any possible confusion.

If DIM is used to allocate the A( ) or A$( ) arrays larger than 26 elements, there are certain special conditions in which an error can cause the part of the array from A(27) or A$(27) on to become inaccessible. If this occurs, it is necessary to redimension the array.

Expressions

An **expression** is some combination of variables, constants, and operators which can be evaluated to a single value. The calculations which you entered in Chapter 3 were examples of expressions. Expressions are an intrinsic part of BASIC programs. For example, an expression might be a formula that computes an answer to some equation, a test to determine the relationship between two quantities, or a means to format a set of strings.

Numeric Operators

The **PC-8** has five **numeric operators.** These are the arithmetic operators which you used when exploring the use of the **PC-8** as a calculator in Chapter 3:

+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Power

A **numeric expression** is constructed in the same way that you entered compound calculator operations. Numeric expressions can contain any meaningful combination of numeric constants, numeric variables, and these numeric operators:

```
(A * B)^2
A(2,3) + A(3,4) + 5.Ø – C
(A/B) * (C+D)
```

In certain circumstances the multiplication operator can be implied:

2A is the same as 2 * A
7C is the same as 7 * C
ABC is the same as A * B * C

As you can see from the last example, there is a possibility that implied multiplication could be confused with other BASIC words, so don't use this form unless the context is very clear. For example, when you enter ABS it does not mean A * B * S but function to obtain absolute value.

String Expressions

String expressions are similar to numeric expressions except that there is only one string operator -- concatenation (+). This is the same symbol used for plus. When used with a pair of strings, the + attaches the second string to the end of the first string and makes one longer string. You should take care in making more complex string concatenations and other string operations because the work space used by the **PC-8** for string calculations is limited to only 79 characters.

NOTE: String quantitites and numeric quantities cannot be combined in the same expression unless one uses one of the functions which convert a string value into a numeric value or vice versa:

"15" + 1Ø	is illegal
"15" + "1Ø"	is "151Ø", not "25"

Relational Expressions

A **relational expression** compares two expressions and determines whether the stated relationship is True or False. The relational operators are:

>	Greater Than
>=	Greater Than or Equal To
=	Equals
<>	Not Equal To
<=	Less Than or Equal To
<	Less Than

The following are valid relational expressions:

```
A<B
C(1,2)>=5
D(3)<>8
```

If A was equal to 1Ø, B equal to 12, C(1, 2) equal to 6, and D(3) equal to 9, all of these relational expressions would be True.

Character strings can also be compared in relational expressions. The two strings are compared character by character according to their ASCII value starting at the first character (see Appendix B for ASCII values). If one string is shorter than the other, a Ø or NUL will be used for any missing positions. All of the following relational expressions are True:

```
"ABCDEF" = "ABCDEF"
"ABCDEF" <> "ABCDE"
"ABCDEF" > "ABCDE"
```

Relational expressions evaluate to either True or False. The **PC-8** represents True by a 1; False is represented by a Ø. In any logical test an expression which evaluates to 1 or more will be regarded as True while one which evaluates to Ø or less will be considered False. Good programming practice, however, dictates the use of an explict relational expression instead of relying on this coincidence.

Logical Expressions

Logical expressions are relational expressions which use the operators AND, OR, and NOT. AND and OR are used to connect two relational expressions; the value of the combined expression is shown in the following tables:

A AND B

		Value of A	
		True	False
Value of B	True	True	False
	False	False	False

A OR B

		Value of A	
		True	False
Value of B	True	True	True
	False	True	False

(Note: Value of A and B must be Ø or 1)

- Decimal numbers can be expressed in the binary notation of 16 bits as follows:

DECIMAL NOTATION	BINARY NOTATION OF 16-BIT
32767	Ø111111111111111
⋮	
3	ØØØØØØØØØØØØØØ11
2	ØØØØØØØØØØØØØØ1Ø
1	ØØØØØØØØØØØØØØØ1
Ø	ØØØØØØØØØØØØØØØØ
−1	1111111111111111
−2	111111111111111Ø
−3	11111111111111Ø1
⋮	⋮
−32768	1ØØØØØØØØØØØØØØØØ

The negative (NOT) of a binary number ØØØØØØØØØØØØØØØ1 is taken as follows:

NOT ØØØØØØØØØØØØØØØ1
(Negative) ⟶ 111111111111111Ø

Thus, 1 is inverted to Ø, and Ø to 1 for each bit, which is called "to take negative (NOT)."

Then, the following will result when 1 and NOT 1 are added together:

```
    ØØØØØØØØØØØØØØØ1  (1)
+)  111111111111111Ø  (NOT 1)
------------------------------
    1111111111111111  (-1)
```

Thus, all bits become 1. According to the above number list, the bits become −1 in decimal notation, that is 1 + NOT 1 = −1.

The relationship between numerical value X and its negative (NOT X) is:

X + NOT X = −1

This results in an equation of NOT X = −X−1

i.e. NOT X = −(X + 1)

From the equation the following are found to result.

NOT Ø = −1
NOT −1 = Ø
NOT −2 = 1

More than two relational expressions can be combined with these operators. You should take care to use parentheses to make the intended comparison clear.

```
(A<9) AND (B>5)
(A>=1Ø) AND NOT (A>2Ø)
(C=5) OR (C=6) OR (C=7)
```

The **PC-8** implements logical operators as "bitwise" logical functions on 16 bit quantities. (See note on relational expressions and True and False). In normal operations this is not significant because the simple 1 and Ø (True and False) which result from a relational expression uses only a single bit. If you apply a logical operator to a value other than Ø or 1, it works on each bit independently. For example if A is 17, and B is 22, (A OR B) is 23:

```
17 in binary notation is  1ØØØ1
22 in binary notation is  1Ø11Ø
17 OR 22 is               1Ø111  (1 if 1 in either number, otherwise Ø)
1Ø111 is 23 in decimal.
```

If you are a proficient programmer, there are certain applications where this type of operation can be very useful. Beginning programmers should stick to clear, simple True or False relational expressions.

Parentheses and Operator Precedence

When evaluating complex expressions the **PC-8** follows a predefined set of priorities to determine the operation sequence. This can be quite significant:

5 + 2 * 3   could be

5 + 2 = 7 7 * 3 = 21	or	2 * 3 = 6 6 + 5 = 11

The exact rules of "operator precedence" are given in Appendix D.

To avoid having to remember all these rules and to make your program clearer, always use parentheses to determine the sequence of evaluation. The above example is clarified by writing either:

(5 + 2) * 3	or	5 + (2 * 3)

Calculator Mode

In general, any of the above expressions can be used in the calculator mode as well as when programming a BASIC statement. In the RUN mode an expression is computed and displayed immediately. For example:

Input	Display
(5 > 3) AND (2 < 6)	1.

The 1 means that the expression is True.

Functions

Functions are special components of the BASIC language which take one value and transform it into another value. Functions act like variables whose value is determined by the value of other variables or expressions. ABS is a function which produces the absolute value of its argument:

ABS (−5)	is	5
ABS (6)	is	6

LOG is a function which computes the log to the base 1Ø of its argument.

LOG (1ØØ) is 2
LOG (1ØØØ) is 3

A function can be used any place that a variable can be used. Many functions do not require the use of parentheses:

LOG 1ØØ is the same as LOG (1ØØ)

You must use parentheses for functions which have more than one argument. Using parentheses always makes programs clearer.

See Chapter 8 for a complete list of functions available on the **PC-8**.

CHAPTER 5
PROGRAMMING

In the previous chapter we examined some of the concepts and terms of the BASIC programming language. In this chapter you will use these elements to create programs on the **PC-8**. Let us reiterate, however, this is **not** a manual on how to program in BASIC. What this chapter will do is familiarize you with the use of BASIC on your **PC-8**.

Programs

A **program** consists of a set of instructions to the computer. Remember the **PC-8** is only a machine. It will perform the exact operations that you specify. You, the programmer, are responsible for issuing the correct instructions.

BASIC Statements

The **PC-8** interprets instructions according to a predetermined format. This format is called a **statement**. You always enter BASIC statements in the same pattern. Statements must start with a line number:

```
1Ø: INPUT A
2Ø: PRINT A * A
3Ø: END
```

Line Numbers

Each line of a program must have a unique line number – any integer between 1 and 999. Line numbers are the reference for the computer. They tell the **PC-8** the order to perform the program. You need not enter lines in sequential order (although if you are a beginning programmer, it is probably less confusing for you to do so). The computer always begins execution with the lowest line number and moves sequentially through the lines of a program in ascending order.

When programming it is wise to allow increments in your line numbering (1Ø, 2Ø, 3Ø, . . . 1Ø, 3Ø, 5Ø, etc). This enables you to insert additional lines if necessary. **CAUTION: Do not use the same line numbers in different programs.** If you use the same line number, the oldest line with that number is deleted when you enter the new line.

BASIC Verbs

All BASIC statements must contain **verbs.** Verbs tell the computer what action to perform. A verb is always contained within a program, and as such is not acted upon immediately.

```
1Ø: INPUT A
2Ø: PRINT A * A
3Ø: END
```

Some statements require or allow an **operand:**

```
1Ø: INPUT A
2Ø: PRINT A * A
3Ø: END
```

Operands provide information to the computer telling it what data the verb will act upon. Some verbs require operands, with other verbs they are optional. Certain verbs do not allow operands. (See Chapter 8 for a complete listing of BASIC verbs and their use on the **PC-8.**)

BASIC Commands

Commands are instructions to the computer which are entered outside of a program. Commands instruct the computer to perform some action with your program or to set modes which effect how your programs are executed.

Unlike verbs, commands have immediate effects – as soon as you complete entering the command (by pressing the ENTER key), the command will be executed. Commands **are not** preceded by a line number:

```
RUN
NEW
RADIAN
```

Some verbs may also be used as commands. (See Chapter 8 for a complete listing of BASIC commands and their use on the **PC-8**).

Modes

You may remember that when using the **PC-8** as a calculator, it is set in the RUN mode.

The RUN mode is also used to execute the programs you create.

The PROgram mode is used to enter and edit your programs.

Beginning to Program on the PC-8

After all your practice in using the **PC-8** as a calculator you are probably quite at home with the keyboard. From now on, when we show an entry, we will **not** show every keystroke. Remember to use SHIFT to access characters above the keys and END EVERY LINE BY PRESSING THE ENTER KEY.

Now you are ready to program. Set the computer to the PROgram mode and enter this command:

Input	Display
NEW	>

The NEW command clears the **PC-8's** memory of all existing programs and data. The prompt appears after you press ENTER , indicating that the computer is awaiting input.

Example 1 – Entering and Running a Program

Make sure the **PC-8** is in the **PRO** mode and enter the following program:

Input	Display
1Ø PRINT "HELLO"	1Ø: PRINT "HELLO"

Notice that when you push ENTER the **PC-8** displays your input, automatically inserting a colon (:) between the line number and the verb. Verify that the statement is in the correct format.

Now change the mode to RUN by pressing the MODE key:

Input	Display
RUN	HELLO

Since this is the only line of the program, the computer will stop executing at this point. Press ENTER to get out of the program and reenter RUN if you wish to execute the program again.

Example 2 – Editing a Program

Suppose you wanted to change the message that your program was displaying, that is you wanted to edit your program. With a single line program you could just retype the entry, but as you develop more complex programs editing becomes a very important component of your programming. Let's edit the program you have just written.

Are you still in the RUN mode?   If so return to the PROgram mode.

You need to recall your program to edit it. Use the Up Arrow (↑) to recall your program. If your program was completely executed, the ↑ will recall the last line of the program. If there was an error in the program, or if you used the BREAK (BRK) key to stop execution, the ↑ will recall the line in which the error or BREAK occurred. To make changes in your program use the ↑ to move up in your program (recall the previous line) and the ↓ to move down in your program (display the next line). If held down the ↑ and the ↓ will scroll vertically, that is, they will display each line moving up or down in your program.

You will remember that to move the cursor within a line you use the ▶ (right arrow) and ◀ (left arrow). Using the ▶ position the cursor over the first character you wish to change:

Input	Display
↑	1Ø: PRINT "HELLO"
◀◀◀◀◀	1Ø PRINT "HELLO"

Notice that the cursor is now in the flashing block form indicating that it is "on top of" an existing character. Type in:

Input	Display
GOOD"!	1Ø "GOOD"!_

Don't forget to press ENTER at the end of the line. Change to the RUN mode.

Input	Display
RUN	ERROR 1 IN 1Ø ^{E}

This is a new kind of error message. Not only is the error type identified (syntax error) but the **line number** in which the error occurs is also indicated.

Press CL and change back to the PROgram mode. You must be in the PROgram mode to make changes in a program. Using ↑ , recall the last line of your program.

Input	Display
↑	1Ø: PRINT "GOOD" !

The flashing cursor is positioned over the problem area. In Chapter 4 you learned that when entering string constants in BASIC all characters must be contained within quotation marks. Use the DELete key to eliminate the "!":

Input	Display
DEL	1Ø PRINT "GOOD"_

Now put the ! in the correct location. When editing programs, DELete and INSert are used in exactly the same way as they are in editing calculations (See Chapter 3). Using the ◄ position the cursor on top of the character which will be the first character following the insertion.

Input	Display
◄	1Ø PRINT "GOOD"

Press the INSert key. A □ will indicate the spot where the new data will be entered:

Input	Display
INS	1Ø PRINT "GOOD□"

Type in the !. The display looks like this:

Input	Display
!	1Ø PRINT "GOOD!"

Remember to press ENTER so the correction will be entered into the program.

NOTE: If you wish to DELete an entire line from your program just type in the line number and the original line will be eliminated.

Examples 3 – Using Variables in Programming

If you are unfamiliar with the use of numeric and string variables in BASIC, re-read these sections in Chapter 4.

Using variables in programming allows much more sophisticated use of the **PC-8**'s computing abilities.

Remember, you assign simple numeric variables using any letter from A to Z:

```
A = 5
```

To assign string variables you also use a letter, followed by a dollar sign. **Do not use the same letter in designating a numeric and a string variable.** You cannot designate A and A$ in the same program.

Remember that simple string variables cannot exceed 7 characters in length:

```
A$ = "TOTAL"
```

The values assigned to a variable can change during the execution of a program, taking on the values typed in or computed during the program. One way to assign a variable is to use the INPUT verb. In the following program the value of A$ will change in response to the data typed in answering the inquiry "WORD?". Enter this program:

```
1Ø INPUT "WORD?"; A$
2Ø B= LEN (A$)
3Ø PRINT "WORD⌴IS⌴"; B; "⌴LTRS"
4Ø END                     ↑———————— means space
```

Before you RUN the program, notice several new features. Line 30 of this program exceeds the 16 character maximum of the **PC-8** display. When a line is longer than 16 characters (up to the 79 character maximum), **PC-8** moves the characters to the left as the 16 character maximum is exceeded. This does not destroy the previous input. This move to the left is referred to as horizontal scrolling.

The second new element in this program is the use of the END statement to signal the completion of a program. END tells the computer that the program is completed. It is always good programming practice to use an END statement.

As your programs get more complex you may wish to review them before you begin execution. To look at your program, use the LIST command. LIST, which can only be used in the PROgram mode, displays programs beginning with the lowest line number.

Try listing this program:

Input	Display
LIST	1Ø: INPUT "WORD?"

Use the ↑ and ↓ arrows to move through your program until you have reviewed the entire program. To review a line which contains more than 16 characters move the cursor to the extreme right of the display and the additional characters will appear on the screen. After checking your program, run it:

Input	Display
RUN	WORD? _
HELP ENTER	WORD IS 4. LTRS
ENTER	>

This is the end of your program. Of course you may begin it again by entering RUN. However, this program would be a bit more entertaining if it presented more than one opportunity for input. We will now modify the program so it will keep running without entering RUN after each answer.

Return to the PRO mode and use the up or down arrows (or LIST) to reach line 4Ø.

You may type 4Ø to Delete the entire line or use the ► to position the cursor over the E in End. Change line 4Øso that it reads:

```
4Ø: GOTO 1Ø
```

Now RUN the modified program.

The GOTO statement causes the program to loop (keep repeating the same operation). Since you put no limit on the loop it will keep going forever (an "infinite" loop). To stop this program hit the BREAK (BRK) key.

When you have stopped a program using the BRK key, you can restart it using the CONT command. CONT stands for CONTinue. With the CONT command the program will restart on the line which was being executed when the BRK key was pressed.

Example 4 – More Complex Programming

The following program computes N Factorial (N!). The program begins with 1 and computes N! up to the limit which you enter. Enter this program.

```
100 F = 1: WAIT 128
110 INPUT "LIMIT? "; L
120 FOR N = 1 TO L
130 F = F * N
140 PRINT N, F
150 NEXT N
160 END
```

Several new features are contained in this program. The WAIT verb in line 100 controls the length of time that displays are held before the program continues. The numbers and their factorials are displayed as they are computed. The time they appear on the display is set by the WAIT statement to approximately 2 seconds, instead of waiting for you to press ENTER .

Also in line 100, notice that there are two statements on the same line separated by a colon (:). You may put as many statements as you wish on one line, separating each by a colon, up to the 80 character maximum including ENTER. Multiple statement lines can make a program hard to read and modify, however, so it is good programming practice to use them only where the statements are very simple or there is some special reason to want the statements on one line.

Also in this program we have used the FOR verb in line 120 and the NEXT verb in line 150 to create a loop. In Example 3 you created an "infinite" loop which kept repeating the statements inside the loop until you pressed the BRK key. With this FOR/NEXT loop the **PC-8** adds 1 to N each time execution reaches the NEXT verb. It then tests to see if N is larger than the limit L. If N is less than or equal to L, execution returns to the top of the loop and the statements are executed again. If N is greater than L, execution continues with line 160 and the program stops.

You may use any numeric variable in a FOR/NEXT loop. You also do not have to start counting at 1 and you can add any amount at each step. See Chapter 8 for details.

We have labelled this program with line numbers starting with 1ØØ. Labelling programs with different line numbers allows you to have several programs in memory at one time. To RUN this program instead of the one at line 1Ø enter:

```
RUN 1ØØ
```

In addition to executing different programs by giving their starting line number, you can give programs a letter name and start them with the DEF key (see Chapter 6).

You will notice that while the program is running, the BUSY indicator is lit at those times that there is nothing on the display. RUN the program a few more times and try setting N at several different values.

Storing Programs in the Memory

Programs remain in memory when the computer is turned off whether manually or automatically. Even if you use the BRK , CLear or CA keys, the programs will remain.

Programs are lost from memory when you perform the following actions:

* You enter NEW or NEW Ø before beginning programming.
* You create a new program using the SAME LINE NUMBERS as a program already in memory.
* You change the batteries.

This brief introduction to programming on the **PC-8** should serve to illustrate the exciting programming possibilities of your new computer.

The following tables show the number of bytes used to define each variable and the number used by each program statement.

Variable	Variable name	Data
Numeric array variable	6 bytes	8 bytes
String array variable	6 bytes	Specified number of bytes*

* For example, if DIM Z$ (2, 3) ∗ 10 is specified, 12 variables, each capable of storing 10 characters, are reserved. This requires 6 bytes (variable name) + 10 bytes (number of characters) x 12 = 126 bytes.

Element	Line number	Statement & function	Others
Number of bytes used	2 bytes	1 byte	1 byte

Remaining Number of Bytes and Number of Usable Variables

The number of remaining bytes in the program/data area can be determined by operating:

MEM ENTER

To determine the number of numeric variables which can be reserved by the DIM statement use the following formula:

(MEM – 6) / 8 ENTER

↑—When calculating the number of string variables, replace this "8" with the necessary number of characters.

CHAPTER 6
SHORTCUTS

The **PC-8** includes several features which make programming more convenient by reducing the number of keystrokes required to enter repetitive material.

One such feature is in the availability of abbreviations for verbs and commands (See Chapter 8)

This chapter discusses the additional feature which can eliminate unnecessary typing — the DEF key.

The DEF Key and Labelled Programs

Often you will want to store several different programs in the **PC-8** memory at one time. (Remember that each must have unique line numbers). Normally, to start a program with a RUN or GOTO command, you need to remember the beginning line number of each program (see Chapter 8). But, there is an easier way! You can label each program with a letter and execute the program using only two keystrokes. This is how to label a program and execute it using DEF:

NOTE: Put a label on the first line of each program to which you can refer. The label consists of a single character in quotes, followed by a colon:

```
1Ø: "A": PRINT "FIRST"
2Ø: END
8Ø: "B": PRINT "SECOND"
9Ø: END
```

Any one of the following characters can be used: A, S, D, F, G, H, J, K, L, =, Z, X, C, V, B, N, M, and SPC. Notice that these are the keys in the last two rows of the alphabetic portion of the keyboard.

NOTE: To execute the program, instead of typing RUN 8Ø or GOTO 1Ø, you need only press the DEF key and then the letter used as a label. In the above example, pressing DEF and then 'B' would cause 'SECOND' to appear on the display.

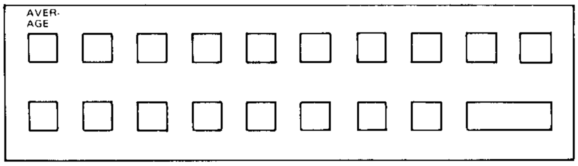
When DEF is used to execute a program, variables and mode settings are affected in the same way as when GOTO is used. See Chapter 8 for details.

Template

One template is provided with the **PC-8**. You can use this template to help you remember frequently used DEF key assignments. After you have labelled the programs, use a pencil to mark the template so you will know what is associated with each key. You can then execute programs using the two-keystroke operation.

For example, if you have one group of programs which you often use at the same time, label the programs with letters and mark the template so that you can easily begin execution of any of the programs with two keystrokes.

Example:



Two strips of double-sided tape are provided with your unit. Use these to keep the template in place.

CHAPTER 7
USING THE OPTIONS

The PC-3 Printer/Cassette Interface (Cat. No. 26-3591) allows you to add a printer and cassette interface to your **Tandy PC-8** Pocket Computer.
The PC-3 Printer/Cassette Interface features:

* 24-character-wide thermal printer with approximately 48-line-per-minute print speed.
* Convenient paper feed and tear bar.
* Simultaneous printing of calculations as desired.
* Easy control of display or printer output in BASIC.
* Cassette interface to connect to any standard cassette recorder.
* Manual and program control of recorder for storing programs and data.
* Filenames and passwords on tape for control and security.
* Built-in rechargeable Nickel-Cadmium batteries for portability.
* Recharger supplied.

Introduction to the Printer/Cassette Interface

Before you begin to use the PC-3 Printer/Cassette Interface you should first become familiar with its components. Examine the front of the machine:

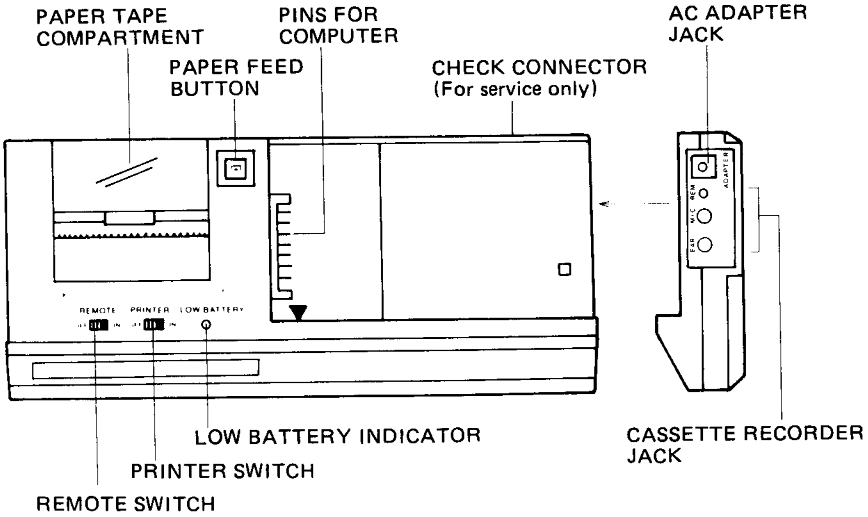


Figure 14. Printer/Cassette Interface (Front View)

- * REMOTE switch. This switch is used to operate the Cassette Recorder manually.
- * PRINTER ON/OFF. This switch is used to turn the printer on and off to conserve batteries when not in use.
- * LOW BATTERY indicator. This indicates when there is insufficient power to operate the **PC-3** Printer/Cassette Interface.
- * Paper feed button. Pressing this key will feed the paper in the printer.

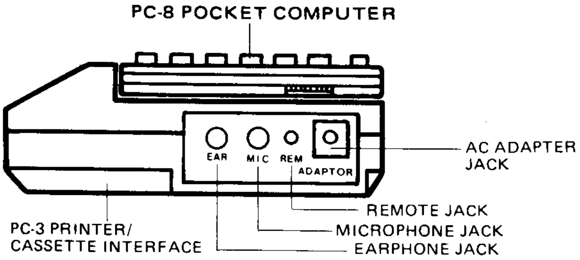


Figure 15. PC-3 Printer/Cassette Interface (Right Side View)

Power

The **PC-3** Printer/Cassette Interface is powered by a rechargeable Nickel Cadmium battery. It is necessary to recharge the battery when the low battery indicator comes ON.

To recharge the battery, turn the Computer and Printer/Cassette Interface power OFF, connect the AC adapter to the Printer/Cassette Interface, and plug the AC adapter into a wall outlet. (See the diagram.) It will take about 15 hours before the battery is fully charged.

Important Note! Using any AC adapter other than the one supplied may damage the Printer/Cassette Interface.

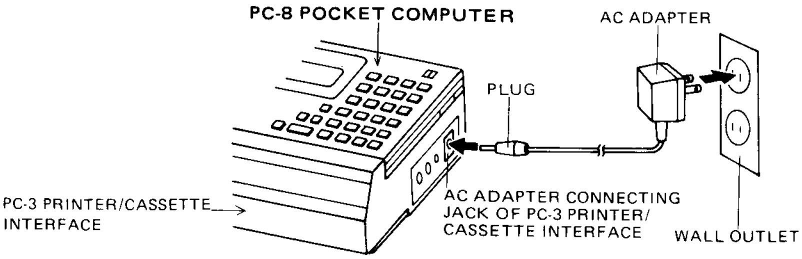

PC-8 POCKET COMPUTER
AC ADAPTER
PLUG
PC-3 PRINTER/CASSETTE INTERFACE
AC ADAPTER CONNECTING JACK OF PC-3 PRINTER/ CASSETTE INTERFACE
WALL OUTLET

Figure 16. How to Connect the AC Adapter

Always connect the recharger to the Printer/Cassette Interface first. Then plug the recharger into the wall socket.

When the batteries in the **PC-3** Printer/Cassette Interface become discharged, the low battery indicator on the front of the unit lights up and the unit will not function. At this point, you must recharge the batteries. When you first receive your Printer/Cassette Interface it is likely that the batteries insufficiently charged due to the time spent in storage. The unit will require charging before its first use.

NOTE: When the Computer is used with the Printer/Cassette Interface and the battery power of the Computer decreases, the power will be supplied to the Computer from the Printer/Cassette Interface.

Connecting the PC-8 Pocket Computer to the PC-3 Printer/Cassette Interface (Cat. No. 26-3591)

To connect the **PC-8** pocket Computer to the **PC-3** Printer/Cassette Interface, use the following procedure:

1. Turn OFF the power in both units.

   NOTE: It is important that the power be OFF on the Computer before connecting the units, or the Computer may "hang up". If this should occur, use the RESET button to clear the Computer.

2. Place the Computer on the Printer/Cassette Interface shown in Fig. 17.
3. Lay the Computer down flat.
4. Gently slide the Computer to the left so that the pins on the Printer/Cassette Interface are inserted into the plug on the Computer.
   DO NOT FORCE the Computer and Printer/Cassette Interface together. If the two parts do not connect easily, STOP and check to see that the parts are correctly aligned.

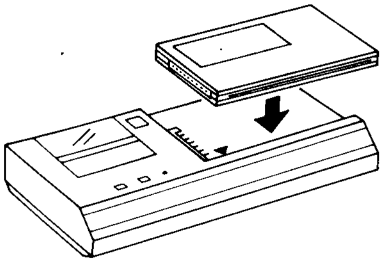


Figure 17.

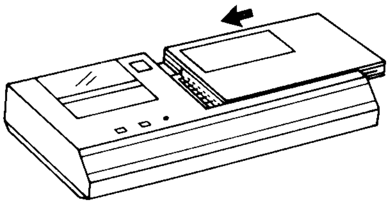


Figure 18.

6. To use the Printer, turn on the **PC-8** Computer power switch, and then the Printer switch.
   Press the CL key.
   If the CL key is not pressed, the Printer may not operate.

Note: If executed when the Printer switch is set at the OFF position, printing causes an error (ERROR code 8). (The low battery indicator may light up at this point.)
In this case, turn the Printer switch ON, and press the CL key. Then, execute the printing again.

Loading the Paper

(1) Turn off the Printer switch.
(2) Open the paper cover. (Fig. 19)

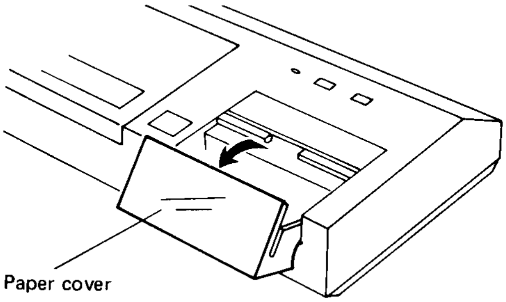


Figure 19.

(3) Insert the leading edge of the roll of paper into the slot located in the paper tape compartment. (Fig. 20) (Fig. 21)
(Any curve or crease near the beginning of the paper makes insertion difficult.)

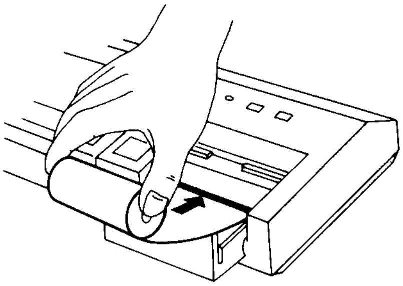


Figure 20.

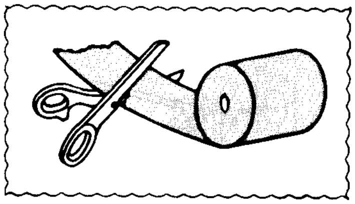


Figure 21.

NOTE: Use of irregular paper tape may cause irregular paper feeding or paper misfeed. Therefore, be sure to tighten the roll before using, as shown in Figure 22.

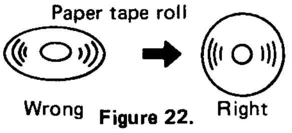


Figure 22.

(4) Turn on the Printer switch and press the paper feed button until the paper comes out of the Printer mechanism. (Fig. 23)

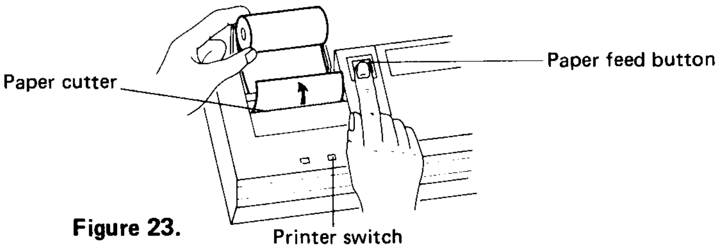


Figure 23.

(5) Install the roll of paper into the compartment.

(6) Close the paper cover. (Fig. 24.)

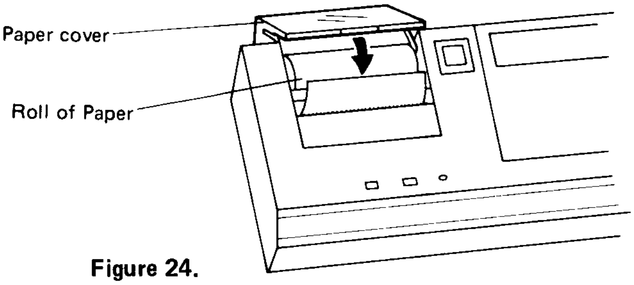


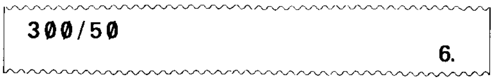
Figure 24.

- To release paper from the printer, cut the paper on the side of the paper roll compartment and then pull it straight out to the cutter side.
  Do not pull the paper backwards, as this may cause damage to the Printer mechanism.

CAUTION:
Paper tape is available wherever the **PC-3** Printer/Cassette Interface is sold. Please order replacement paper tape at your local Radio Shack store. Please specify Model name when ordering. The paper tape is specifically designed for this unique Printer. Use of any other paper tape may cause damage to the unit.

Using the Printer

If you are using the **PC-8** Computer as a calculator, you may use the **PC-3** Printer to simultaneously print your calculations. This is easily accomplished by pressing the SHIFT key and then the ENTER key (P↔NP). (The printer indicator "P" will be displayed. If not, press the SHIFT and ENTER keys. Check to see that the mode switch is set at the RUN position.) After this, when you press ENTER at the end of a calculation, the contents of the display will be printed on one line and the results will be printed on the next. For example:

Input	Paper
3ØØ/5Ø	

```
3ØØ/5Ø
                    6.
```

You may print output on the Printer from within BASIC programs by using the LPRINT statement (see Chapter 8 for details). LPRINT functions in exactly the same fashion as the PRINT statement, except that the Printer accepts text that is 24 characters wide. The only difference is that, if you PRINT something to the display which is longer than 16 characters, there is no way for you to see the extra characters. With the LPRINT verb, the extra characters will be printed on a second and possibly a third line, as is required.

Programs which have been written with PRINT can be converted to work with the Printer by including a PRINT=LPRINT statement in the program (see Chapter 8 for details). ALL PRINT statements following this statement will act as if they were LPRINT statements. PRINT=PRINT will reset this condition to its normal state. This structure may also be included in a program in an IF statement allowing a choice of output at the time the program is used.

```
1Ø INPUT "DISP./PRINTER"; A$
2Ø IF ASC (A$)=8Ø THEN PRINT = LPRINT
```

You may also list your programs on the Printer with the LLIST command (see Chapter 8 for details). If used without line numbers, LLIST will list all program lines currently in memory in their numerical order by line number. A line number range may also be given with LLIST to limit the lines which will be printed. When program lines are longer than 24 characters, two or more lines may be used to print one program line. The second and succeeding lines will be indented four characters so that the line number will clearly identify each separate program line.

Caution:

- In case an error (ERROR code 8) occurs due to a paper misfeed, tear off the paper tape, and pull the remaining part of the paper tape completely out of the Printer. Then press the CL key to clear the error condition.
- When the Printer/Cassette Interface is exposed to strong external electrical noise, it may print numbers at random. If this happens, depress the BRK key to stop the printing; then press the CL key.
  Pressing the CL key will return the Printer to its normal condition.

  > When the Printer misfeeds a paper or is exposed to strong external electrical noise while printing, it may not operate normally and only the symbol "BUSY" is displayed. If this happens, depress the BRK key to stop the printing. (Adjust the paper so that it will feed correctly.) Press the CL key.

- When the PC-3 Printer/Cassette Interface is not in use, turn off the Printer switch to save the battery life.
- Even while printing under the LPRINT command, the entry can be executed when an INPUT, INKEY$ or PRINT command is performed.
  In this case, however, the Printer will stop if the CL key is pressed. Therefore, be sure to press the CL key upon completion of printing.

Using a Cassette Recorder

With the Cassette Recorder connected, you can use the following commands:

CSAVE	Saves the contents of a program on tape.
CLOAD	Retrieves a program from tape.
CLOAD?	Compares the program on tape with the contents of memory to insure that you have a good copy.
MERGE	Combines a program on tape with one already in memory.
PRINT#	Saves the contents of variables on tape.
INPUT#	Retrieves the contents of variables from tape.
CHAIN	Starts execution of a program which has been stored on tape.

Programs may be assigned filenames which will be stored on the tape. This allows the unambiguous storage of many programs on one tape. Programs can then be retrieved by name and the tape will be searched to find the appropriate file. If programs have been password-protected in memory, they cannot be stored on tape, but a password can be assigned at the time that unprotected programs are CSAVEd. Such password-protected programs can be used by another person, but they will not be able to LIST or modify the programs in any way.

See Chapter 8 for details on all these verbs and commands.

When a program or data is recorded on tape, it will be preceded by a high-pitched tone of approximately 7 seconds. This tone serves to advance the tape past any leader and to identify the beginning of each program or set of data.

When searching for a filename, the tape can read only in a forward direction. This search is relatively slow, so it is sometimes preferable to keep track of program locations by using the tape counter. Using fast forward, rewind or play, the tape can be manually positioned to the leader tone area of the correct program before the retrieval is started. While scanning the tapes, you will be able to hear the high tones which begin each program. In between these high tones will be a mixed high and low tone sound which indicates programs or data.

See the Operation Manual supplied with the PC-3 Printer/Cassette Interface for more detailed operating instructions.

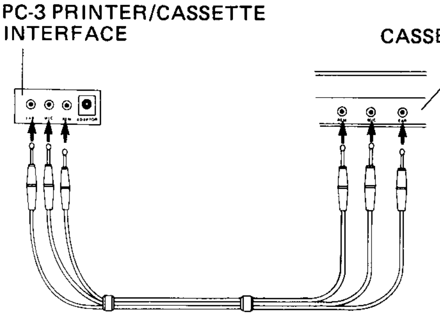


Figure 25. Cassette Cables and Interface Jacks

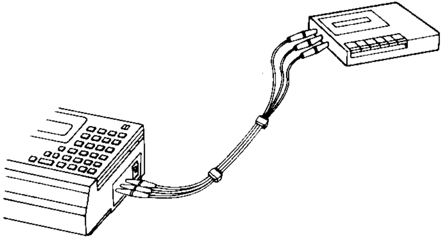


Figure 26. Recorder Connected to Interface

- To transfer programs and data from the tape, use the tape recorder with which the tape was prerecorded. A tape recorder, if different from that used for recording, may cause no transfer of the prerecorded tape.

Care and Maintenance

* Be sure that the power is OFF on both units when connecting or disconnecting the Printer/Cassette Interface and the Computer.
* The Printer should be operated on a level surface.
* The unit should be kept away from extreme temperatures, moisture, dust, and loud noises.
* Use a soft, dry cloth to clean the unit. DO NOT use solvent or a wet cloth.
* Keep foreign objects out of the unit.

Errors

If the batteries become low, or if the Printer/Cassette Interface is subjected to strong noise, the unit may cease to function and the Pocket Computer may "hang up". This can also occur if the units are connected and the power on the Printer/Cassette Interface is not turned on when an LPRINT or LLIST command is used. In some cases, ERROR 8 may be displayed on the Computer.

The CLear key may usually be used to clear this condition, but in some cases a RESET may be required. Be sure to restore adequate power to the Printer/Cassette Interface before attempting to use it again.

Examples

The procedures for the Computer and the Cassette Recorder operation

1. **Saving**
   (1) Turn off the REMOTE switch.
   (2) Put a tape into the Cassette Recorder.
   (3) Turn on the REMOTE switch.
   (4) Depress the RECORD button.
   (5) With the same command which saves your program, you must give the program a "filename". This is for reference purposes. Your filename cannot be longer than 7 characters. To save the program with a filename, type:

```
CSAVE [SHIFT] " PRO-1 [SHIFT] "
```

Your program will be saved with the name "PRO-1". You can assign any name you desire, whatever is easiest for you to keep track of. Also, note that there is a 7-character length limit for your filename. If the name is longer than 7 characters, the excess is ignored. A good practice is to maintain a program log, which includes the program name, starting and stopping locations on tape (use the counter numbers), and a brief description of what the program does.

Press the [ENTER] key. If your tape recorder has a monitor feature, 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.

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. In order to insure that this has in fact been accomplished, read it back into memory from the tape as explained in the next section.

2. Collating the Computer and Tape Contents

Now that your 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.

(1) Turn off the REMOTE switch to clear remote control functions.

(2) Rewind the tape to the place at which you started, again using the number counter.

(3) Turn on the REMOTE switch to set remote control functions.

(4) Depress the PLAY button.

(5) To collate the program with a filename type:

CLOAD SHIFT ? SHIFT "PRO-1 SHIFT "

Press the ENTER key.

The computer compares the CSAVEd program with the one in its memory. If all went well, it will display the "prompt" and end its check. If all did not go well, an error message will be displayed, usually ERROR 8. This tells you that the program on tape is somehow different from the program in the computer's memory. Erase that portion of the tape and start again.

3. Transfer from Tape

(1) Turn off the REMOTE switch.

(2) Rewind the tape to the place at which you started, again using the number counter.

(3) Stop rewinding.

(4) Turn the REMOTE switch back ON.

(5) Press the PLAY button.

(6) Type:

CLOAD SHIFT "PRO-1 SHIFT "

and press the ENTER key.

(Remember "PRO-1" is the filename you have given to your program. If you saved the program under another name, you must use that name instead of PRO-1.)

(7) The "BUSY" indicator will now light up, and the program will be brought back into the Computer's memory for use.

(8) The cassette retains a copy of the program, so you can CLOAD the same program over and over again!

If an error message (ERROR 8) is displayed while loading, start again from step (1) in this section, "Transfer from Tape."

Precautions for collation and transfer

The program is recorded on tape as illustrated below:

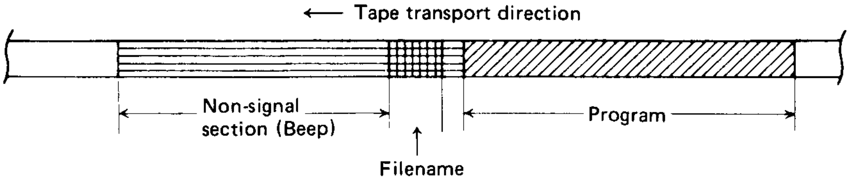


Figure 27.

When the tape is played back, its non-signal section produces a specific continuous beep, while the filename and program-recorded sections cause an intermittent beep.

If collation or transfer was not done properly, the "BUSY" symbol does not disappear and the tape does not stop. To stop the tape operation, press the BRK key. Then, try again from the beginning.

CHAPTER 8
BASIC REFERENCE

The following chapter is divided into three sections:

Commands: Instructions which are used outside a program to change the working environment, perform utilities, or control programs.

Verbs: Action words used in programs to construct BASIC statements.

Functions: Special operators used in BASIC programs to change one variable into another.

Commands and verbs are arranged alphabetically. Each entry is on a separate page for easy reference. The contents of each section is shown in the tables below so that you can quickly identify the category to which an operator belongs. Functions are grouped according to four categories and arranged alphabetically within category.

Commands

Program Control

- CONT
- GOTO*
- NEW
- NEW Ø
- RUN

Cassette Control

- CLOAD
- CLOAD?
- CSAVE
- INPUT#*
- MERGE
- PRINT#*

Debugging

- LIST
- LLIST
- TROFF*
- TRON*

Variables Control

- CLEAR
- DIM*

Angle Mode Control

- DEGREE*
- GRAD*
- RADIAN*

Other

- PASS*
- RANDOM*
- USING*
- WAIT*

*These commands are also BASIC verbs. Their effect as commands is identical to their effect as verbs so they are not described in the command reference section. See the verb reference section for more information.

Verbs

Control and Branching

CHAIN
END
FOR . . . TO . . . STEP
GOSUB
GOTO
IF . . . THEN
NEXT
ON . . . GOSUB
ON . . . GOTO
RETURN
STOP

Assignment and Declaration

CLEAR
DIM
LET

Input and Output

AREAD
CSAVE
DATA
INPUT
INPUT#
LPRINT
PAUSE
PRINT
PRINT#
USING
READ
RESTORE
WAIT

Other

DEGREE
GRAD
RADIAN
RANDOM
REM
TROFF
TRON

Functions

Pseudovariables

INKEY$
MEM
PI

'String Functions

ASC
CHR$
LEFT$
LEN
MID$
RIGHT$
STR$
VAL

Numeric Functions

ABS
ACS
ASN
ATN
COS
DEG
DMS
EXP
INT
LOG
LN
RND
SGN
SIN
SQR
TAN

COMMANDS

1 **CLOAD**
2 **CLOAD** "filename"

Abbreviations: CLO., CLOA.

See also: CLOAD?, CSAVE, MERGE, PASS

Purpose

The CLOAD command is used to load a program saved on cassette tape. It can only be used with the optional **PC-3** Printer/Cassette Interface (26-3591).

Use

The first form of the CLOAD command clears the memory of existing programs and loads the first program stored on the tape, starting at the current position.

The second form of the CLOAD command clears the memory, searches the tape for the program whose name is given by "filename", and loads the program.

Examples

CLOAD	Loads the first program from the tape.
CLOAD "PRO3"	Searches the tape for the program named 'PRO3' and loads it.

Notes:

1. If the designated file name is not retrieved, the computer will continue to search the file name even after the tape reaches the end. In this case, stop the retrieval function by pressing the $\overset{\text{ON}}{\boxed{\text{BRK}}}$ key. This also applies to MERGE, CHAIN, CLOAD? and INPUT # commands to be described later.
2. If an error occurs during CLOAD or CHAIN command (to be described later) execution, the program stored in the computer will be invalid.

1 **CLOAD?**
2 **CLOAD?** "filename"

Abbreviations: CLO.?, CLOA.?

See also: CLOAD, CSAVE, MERGE, PASS

Purpose

The CLOAD? command is used to compare a program saved on cassette tape with one stored in memory. It can only be used with the optional **PC-3** Printer/Cassette Interface.

Use

The first form of the CLOAD? command compares the program stored in memory with the first program stored on the tape, starting at the current position.

The second form of the CLOAD? command searches the tape for the program whose name is given by "filename" and then compares it to the program stored in memory.

Examples

CLOAD?	Compares the first program from the tape with the one in memory.
CLOAD? "PRO3"	Searches the tape for the program named 'PRO3' and compares it to the one stored in memory.

1 **CONT**

Abbreviations: C., CO., CON.

See also: RUN, STOP verb

Purpose

The CONT command is used to continue a program which has been temporarily halted.

Use

When the STOP verb is used to halt a program during execution, the program can be continued by entering CONT in response to the prompt.

When a program is halted using the BRK key, the program can be continued by entering CONT in response to the prompt.

Examples

CONT   Continues an interrupted program execution.

1 **CSAVE**
2 **CSAVE** "filename"
3 **CSAVE** , "password"
4 **CSAVE** "filename", "password"

Abbreviations: CS., CSA., CSAV.

See also: CLOAD, CLOAD?, MERGE, PASS

Purpose

The CSAVE command is used to save a program to cassette tape. It can only be used with the optional **PC-3** Printer/Cassette Interface.

Use

The first form of the CSAVE command writes all of the programs in memory on to the cassette tape without a specified file name.

The second form of the CSAVE command writes all of the programs in memory on to the cassette tape and assigns the indicated file name.

The third form of the CSAVE command writes all of the programs in memory on to the cassette tape without a specified file name and assigns the indicated password. Programs saved with a password may be loaded by anyone, but only someone who knows the password can list or modify the programs. (See discussion under PASS command).

The fourth form of the CSAVE command writes all of the programs in memory on to the cassette tape and assigns them the indicated file name and password.

Examples

CSAVE "PRO3", "SECRET"	Saves the programs now in memory on to the tape under the name 'PRO3', protected with the password 'SECRET'.

1 **GOTO** expression

Abbreviations: G., GO., GOT.

See also: RUN

Purpose

The GOTO command is used to start execution of a program.

Use

The GOTO command can be used in place of the RUN command to start program execution at the line number specified by the expression.

GOTO differs from RUN in five respects:

1) The value of the interval for WAIT is not reset.
2) The display format established by USING statements is not cleared.
3) Variables and arrays are preserved.
4) PRINT = LPRINT status is not reset.
5) The pointer for READ is not reset.

Execution of a program with GOTO is identical to execution with the DEF key.

Examples

GOTO 100	Begins execution of the program at line 100.

1 **LIST**
2 **LIST** expression

Abbreviations: L., LI., LIS.

See also: LLIST

Purpose

The **LIST command** is used to display a program.

Use

The LIST command may only be used in the PROgram mode. The first form of the LIST command displays the statement with the lowest line number.

The second form displays the statement with the nearest line number greater than the value of the expression. The Up Arrow and Down Arrow keys may then be used to examine the program.

Examples

LIST 1ØØ	Displays line number 1ØØ.

1 **LLIST**
2 **LLIST** expression 1 , expression 2

Abbreviations: LL., LLI., LLIS.

See also: LIST

Purpose

The **LLIST** command is used for printing a program on the optional **PC-3** Printer/Cassette Interface.

Use

The LLIST command may only be used in the PROgram mode.

The first form prints all of the programs in memory.

The second form prints the statements from the line number with the nearest line equal to or greater than the value of expression 1 to the nearest line equal to or greater than the value of expression 2. There must be at least two lines between the two numbers.

Examples

LLIST **1ØØ, 2ØØ** Lists the statements between line numbers **1ØØ** and **2ØØ**.

```
1 MERGE
2 MERGE "filename"
```

Abbreviations: MER., MERG.

See also: CLOAD, CLOAD?, CSAVE, PASS verb

Purpose

The MERGE command is used to load a program saved on cassette tape and merge it with programs existing in memory. It can only be used with the optional **PC-3** Printer/Cassette Interface.

Use

The first form of the MERGE command loads the first program stored on the tape starting at the current position and merges it with programs already in memory.

The second form of the MERGE command searches the tape for the program whose name is given by "filename", and merges it with the programs already in memory.

Programs with overlapping line numbers are treated as one program after merging.

If the program in memory is password protected, another password protected program cannot be merged with it. If the program on cassette is not password protected, it becomes protected by the password of the program in memory when merged.

If the program in memory is not password protected, it becomes protected by the password of the program on the cassette when merged.

Examples

MERGE	Merges the first program from the tape.
MERGE "PRO3"	Searches the tape for the program named 'PRO3' and merges it.

Note: For example, let's assume the computer memory contains the following program:

```
10: PRINT "DEPRECIATION"
20: INPUT "METHOD: " ; A
```

MERGE

At this point you remember that you 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: MERGE "DEP1" and press ENTER.

The computer will now load "DEP1" into memory IN ADDITION to the above program. After "DEP1" is loaded, you might find something in memory similar to this:

```
10: PRINT "DEPRECIATION"
20: INPUT "METHOD: " ; A
10: "DEP1" : REM >> SECOND MODULE <<
20: PRINT "INTEREST CHARGES"
30: INPUT "AMT. BORROWED: " ; B
   :
(etc)
```

Note that unlike the CLOAD command, the new program DID NOT replace the existing one and that some line numbers have been duplicated. Also note that a "label" was used on the first line of the merged module. This allows "LINKING" of the modules together (See LINKING MERGED MODULES – on the next page). It is important that you review the following information before proceeding with any further editing or programing:

IMPORTANT NOTES:

Once a MERGE is performed, no INSERTIONS, DELETIONS, or CHANGES are allowed to previously existing program lines.

Example:

```
10 "A" REM THIS IS EXISTING PROGRAM
20 FOR T = 1 TO 100
30 LPRINT T
40 NEXT T
 : (etc)
```

BEFORE doing a MERGE of the next program, make any necessary changes to this program.
Then MERGE the next program: MERGE "PROG2" (example)

```
10 "B"  REM  THIS IS MERGED PROGRAM
20 INPUT  "ENTER DEPRECIATION:  " ; D
30 INPUT  "NUMBER OF YEARS:  " ; Y
40     etc.
```

Now you may make changes to the above program since it was the last MERGED portion.

LINKING MERGED MODULES (programs) TOGETHER

Since the processor executes your program lines in logical sequence, it will stop when it encounters a break in the sequence in line numbering, i.e. if line numbers 10, 20, 30 are followed by duplicate line numbers in a second module, the following techniques are valid: GOTO "B" GOSUB "B", IF. . . THEN "B" (B is used for example only, you can use any label.)

```
1  NEW
2  NEW Ø
Abbreviations: none
```

Purpose

The NEW command is used to clear existing programs.

Use

The NEW command may only be used in the PROgram mode.

The first form of the NEW command clears all programs and data which are currently in memory. (The programs with a password cannot be cleared.)

The second form of the NEW command clears all programs and data which are currently in memory. Note that the programs with a password can be cleared.

The NEW command is not defined in the RUN mode and will result in an ERROR 9.

Examples

NEW	Clears programs or data.

1 **PASS** "character string"

Abbreviations: PA., PAS.

See also: CSAVE, CLOAD

Purpose

The **PASS command** is used to set and cancel passwords.

Use

Passwords are used to protect programs from inspection or modification by other users. A password consists of a character string which is **no more than seven characters long.** The seven characters must be alphabetic or one of the following special symbols: ! # $ % & ( ) * + – / , . : ; < = > ? @ √ π ^

Once a PASS command has been given the programs in memory are protected. A password protected program cannot be examined or modified in memory. It cannot be output to tape or listed with LIST or LLIST, nor is it possible to add or delete program lines. If several programs are in memory and PASS is entered, all programs in memory are protected. If a non-password protected program is merged with a protected program, the merged program is protected. The way to remove this protection is to execute another PASS command with the same password, or execute NEW Ø ENTER in PROgram mode.

Examples

PASS "SECRET"	Establishes the password 'SECRET' for all programs in memory.

1 **RUN**
2 **RUN** line number

Abbreviations: R., RU.

See also: GOTO

Purpose

The **RUN command** is used to execute a program in memory.

Use

The first form of the RUN command executes a program beginning with the lowest numbered statement in memory.

The second form of the RUN command executes a program beginning with the specified line number.

RUN differs from GOTO in five respects:

1) The value of the interval for WAIT is reset.
2) The display format established by USING statements is cleared.
3) Variables and arrays other than the fixed variables are cleared.
4) PRINT = PRINT status is set.
5) The pointer for READ is reset to the beginning DATA statement.

Execution of a program with GOTO is identical to execution with the DEF key. In all three forms of program execution FOR/NEXT and GOSUB nesting is cleared.

Examples

RUN 1ØØ	Executes the program which begins at line number 1ØØ.

VERBS

1 **AREAD** variable name

Abbreviations: A., AR., ARE., AREA.

See also: INPUT verb and discussion of the use of the DEF key in Chapter 6

Purpose

The AREAD verb is used to read in a single value to a program which is started using the DEF key.

Use

When a program is labelled with a letter, so that it can be started using the DEF key, the AREAD verb can be used to enter a single starting value without the use of the INPUT verb. The AREAD verb must appear on the first line of the program following the label. If it appears elsewhere in the program, it will be ignored. Either a numeric or string variable may be used, but only one can be used per program.

To use the AREAD verb, type the desired value in the RUN mode, press the DEF key, followed by the letter which identifies the program. If a string variable is being used, it is not necessary to enclose the entered string in quotes.

Examples

```
1Ø "X": AREAD N
2Ø PRINT N^2
3Ø END
```

Entering "7 DEF X" will produce a display of "49".

Notes:

1. When the display indicates PROMPT (">") at the start of program execution, the designated variable is cleared.
2. When the contents of the display have been displayed by a PRINT verb just prior to the start of program execution, the following is stored:

Verbs
AREAD

- When the display indicates PRINT numeric expression, numeric expression or PRINT "String", "String", the contents on the right of the display are stored.

Example: When the program below is executed;

```
10 "A" : PRINT "ABC", "DEFG"
20 "S" : AREAD A$: PRINT A$
RUN mode
DEF A  →     ABC     DEFG
DEF S  →     DEFG
```

- When the display indicates PRINT Numeric expression; Numeric expression; Numeric expression..., the contents displayed first (on the extreme left) are stored.
- When the display indicates PRINT "String"; "String"; "String"..., the "String" designated last are stored.

1 **CHAIN**
2 **CHAIN** expression
3 **CHAIN** "filename"
4 **CHAIN** "filename", expression

Abbreviations: CH., CHA., CHAI.

See also: CLOAD, CSAVE, and RUN

Purpose

The CHAIN verb is used to start execution of a program which has been stored on cassette tape. It can only be used with the optional **PC-3** Printer/Cassette Interface.

Use

To use the CHAIN verb one or more programs must be stored on a cassette. Then, when the CHAIN verb is encountered in a running program, a program is loaded from the cassette and executed.

The first form of CHAIN loads the first program stored on the tape and begins execution with the lowest line number in the program. The effect is the same as having entered CLOAD and RUN when in the RUN mode.

The second form of CHAIN loads the first program stored on the tape and begins execution with the line number specified by the expression.

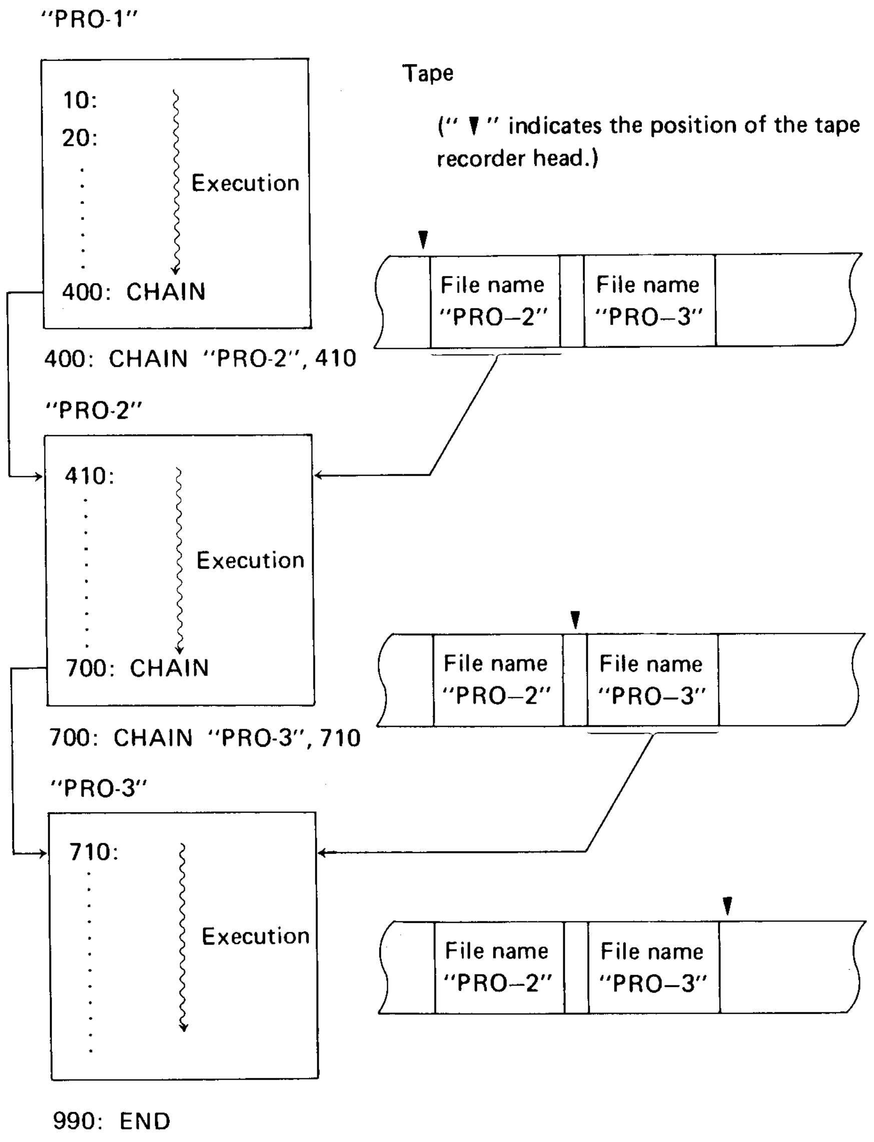
The third form of CHAIN searches the tape for the program whose name is indicated by "filename", loads the program, and begins execution with the lowest line number.

The fourth form of CHAIN will search the tape for the program whose name is indicated by filename, load the program, and begin execution with the line number indicated by the expression.

Examples

1Ø CHAIN	Loads the first program from the tape and begins execution with the lowest line number.
2Ø CHAIN "PRO-2", 48Ø	Searches the tape for a program named PRO-2, loads it, and begins execution with line number 48Ø.

For example, let's assume you have three program sections named PRO–1, PRO–2, PRO–3. Each of these sections ends with a CHAIN statement.



(" ▼ " indicates the position of the tape recorder head.)

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.

1 CLEAR

Abbreviations: CL., CLE., CLEA.

See also: DIM

Purpose

The CLEAR verb is used to erase all variables which have been used in the program and to reset all preallocated variables to zero or NUL.

Use

The CLEAR verb recovers space which is being used to store variables. This might be done when the variables used in the first part of a program are not required in the second part and available space is limited. CLEAR may also be used at the beginning of a program when several programs are resident in memory and you want to clear out the space used by execution of prior programs.

CLEAR does not free up the space used by the variables A – Z, A$ – Z$, or A(1) – A(26) since they are permanently assigned (see Chapter 4). However the contents of these preallocated variables are reset numeric variables to zero and string variables to NUL.

Examples

```
1Ø A = 5 : DIM C(5)
2Ø CLEAR              Frees up the space assigned to C( ) and resets A to zero.
```

1 DEGREE

Abbreviations: DE., DEG., DEGR., DEGRE.

See also: GRAD and RADIAN

Purpose

The DEGREE verb is used to change the form of angular values to decimal degrees.

Use

The **PC-8** has three forms for representing angular values – decimal degrees, radians and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The DEGREE function changes the form for all angular values to decimal degree form until a GRAD or RADIAN verb is used. The DMS and DEG functions can be used to convert decimal degrees to degree, minute, second form and vice versa.

Examples

```
1Ø DEGREE
2Ø X = ASN 1          X now has a value of 9Ø, i.e. 9Ø degrees, the Arcsine of 1.
```

1 **DATA** expression list

Where: expression list is: expression
or: expression , expression list

Abbreviations: DA., DAT.

See also: READ, RESTORE

Purpose

The DATA verb is used to provide values for use by the READ verb.

Use

When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR . . . NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the sequential order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

DATA statements have no effect if encountered in the course of regular execution of the program, so they can be inserted wherever it seems appropriate. Many programmers like to include them immediately following the READ which uses them. If desired, the values in a DATA statement can be read a second time by using the RESTORE statement.

Examples

```
1Ø DIM B(1Ø)            Sets up an array.
2Ø FOR I = 1 TO 1Ø
3Ø READ B(I)            Loads the values from the DATA statement into B( )
4Ø NEXT I               B(1) will be 1, B(2) will be 2, B(3) will be 3, etc.
5Ø DATA 1, 2, 3, 4, 5, 6
6Ø DATA 7, 8, 9, 1Ø
```

1 **DIM** dim list

Where:	dim list	is:	dimension spec.
		or:	dimension spec. , dim list
and:	dimension spec.	is:	numeric dim spec.
		or:	string dim spec.
and:	numeric dim spec	is:	numeric name (size)
and:	string dim spec	is:	string name (dims)
		or:	string name (dims) * len
and:	numeric name	is:	valid numeric variable name
and:	string name	is:	valid string variable name
and:	dims	is:	size
		or:	size, size
and:	size	is:	number of elements
and:	len	is:	length of each string in a string array

Abbreviations: D., DI.

Purpose

The DIM verb is used to reserve space for numeric and string array variables.

Use

Except for A(1) ~ A(26) and A$(1) ~ A$(26), which are predefined (see Chapter 4), a DIM verb must be used to reserve space for any array variable. An array variable and a simple variable may have the same name. A string array and a numeric array may have the same name except for the dollar sign.

The maximum number of dimensions in any array is two; the maximum size of any one dimension is 255. In addition to the number of elements specified in the dimension statement, one additional "zeroeth" element is reserved. For example, DIM B(3) reserves B(Ø), B(1), B(2), and B(3). In two dimensional arrays there is an extra "zeroeth" row and column.

In string arrays one specifies the size of each string element in addition to the number of elements. For example, DIM B$(3)*12 reserves space for 4 strings which are each a maximum of 12 characters long. If the length is not specified each string can contain a maximum of 16 characters.

When a numeric array is dimensioned, all values are initially set to zero; in a string array the values are set to NUL.

A( ) and A$( ) may be dimensioned to sizes larger than 26 with the DIM statement. In this case, part of the array is in the preallocated memory and part is in program memory. See Chapter 4.

Examples

1Ø DIM B(1Ø)	Reserves space for a numeric array with 11 elements.
2Ø DIM C$(4, 4)*1Ø	Reserves space for a two dimensional string array with 5 rows and 5 columns; each string will be a maximum of 1Ø characters.

NOTE:

The **PC-8** makes it possible to use an expression as the suffix of two-dimensional string array variables.
For the second suffix, however, do not use an array variable. However, arrays, such as A(30) can be used with this.

Example 1: $B\ (\underbrace{A*B}_{\text{1st suffix}},\ \underbrace{C\ (\emptyset)}_{\text{2nd suffix}}) = 10$. . . . . Not usable

Example 2: B (C (Ø), 5) = 10. . . . . . . Usable

Example 3: B (4, A(30)) = 10. . . . . Usable

In Example 1, therefore, C (Ø) can be used when replaced by A(30), or if required, A(30) = C (Ø) is placed before it.

1 END

Abbreviations: E., EN.

Purpose

The END verb is used to signal the end of a program.

Use

When multiple programs are loaded into memory at the same time a mark must be included to indicate where each program ends so that execution does not continue from one program to another. This is done by including an END verb as the last statement in the program.

Examples

```
1Ø PRINT "HELLO"
2Ø END
3Ø PRINT "GOODBYE"
4Ø END
```

With these programs in memory a 'RUN 1Ø' prints 'HELLO', but not 'GOODBYE'. 'RUN 3Ø' prints 'GOODBYE'.

1 **FOR** numeric variable = expression 1 **TO** expression 2
2 **FOR** numeric variable = expression 1 **TO** expression 2
STEP expression 3

Abbreviations: F. and FO.; STE.

See also: NEXT

Purpose

The FOR verb is used in combination with the NEXT verb to repeat a series of operations a specified number of times.

Use

The FOR and the NEXT verbs are used in pairs to enclose a group of statements which are to be repeated. The first time this group of statements is executed the loop variable (the variable named immediately following the FOR) has the value of expression 1.

When execution reaches the NEXT verb this value is tested against expression 2. If the value of the loop variable is less than expression 2, the loop variable is increased by the step size and the enclosed group of statements is executed again, starting with the statement following the FOR. In the first form the step size is 1; in the second form the step size is given by expression 3. If the value of the loop variable is greater than or equal to expression 2, execution continues with the statement which immediately follows the NEXT. Because the comparison is made at the end, the statements within a FOR/NEXT pair are always executed at least once.

Expression 1 may have any value in the numeric range. When expression 1 and expression 2 are compared, only the integer part is used in the expression 2. Expression 2 and expression 3 must be an integer in the range of –32768 to 32767; Expression 3 may not be zero.
Expression 1, expression 2 and expression 3 can also be specified in negative.

The loop variable may be used within the group of statements, for example as an index to an array, but care should be taken in changing the value of the loop variable.

Programs should be written so that they never jump from outside a FOR/NEXT pair to a statement within a FOR/NEXT pair. Similarly, programs must never leave a FOR/NEXT pair by jumping out. Always exit a FOR/NEXT loop via the NEXT statement. To do this, set the loop variable to a value higher than expression 2.

The group of statements enclosed by a FOR/NEXT pair can include another pair of FOR/NEXT statements which use a different loop variable as long as the enclosed pair is completely enclosed; i.e., if a FOR statement is included in the group, the matching NEXT must also be included. FOR/NEXT pairs may be "nested" up to five levels deep.

Examples

```
1Ø FOR I = 1 TO 5
2Ø PRINT I
3Ø NEXT I
```

This group of statements prints the numbers 1, 2, 3, 4, 5.

```
4Ø FOR N = 1Ø TO Ø STEP -1
5Ø PRINT N
6Ø NEXT N
```

This group of statements counts down 1Ø, 9, 8, 7, 6, 5, 4, 3, 2, 1, Ø.

```
7Ø FOR N = 1 TO 1Ø
8Ø X = 1
9Ø FOR F = 1 TO N
1ØØ X = X * F
11Ø NEXT F
12Ø PRINT X
13Ø NEXT N
```

This group of statements computes and prints N factorial for the numbers from 1 to 1Ø.

1 **GOSUB** expression

Abbreviations: GOS., GOSU.

See also: GOTO, ON . . . GOSUB, ON . . . GOTO, RETURN

Purpose

The GOSUB verb is used to execute a BASIC subroutine.

Use

When you wish to execute the same group of statements several times in the course of a program or use a previously written set of statements in several programs, it is convenient to use the BASIC capability for subroutines using the GOSUB and RETURN verbs.

The group of statements is included in the program at some location where they are not reached in the normal sequence of execution. A frequent location is following the END statement which marks the end of the main program. At those locations in the main body of the program – where subroutines are to be executed, include a GOSUB statement with an expression which indicates the starting line number of the subroutine. The last line of the subroutine must be a RETURN. When **GOSUB** is executed, the **PC-8** transfers control to the indicated line number and processes the statements until a RETURN is reached. Control is then transferred back to the statement following the GOSUB.

A subroutine may include a GOSUB. Subroutines may be "nested" in this fashion up to 1Ø levels deep.

The expression in a GOSUB statement may not include a comma, e.g., 'A(1, 2)' cannot be used. Since there is an ON . . . GOSUB structure for choosing different subroutines at given locations in the program, the expression usually consists of just the desired line number. When a numeric expression is used it must evaluate to a valid line number, i.e., 1 to 999, or an ERROR 4 will occur.

Examples

```
1Ø GOSUB 1ØØ
2Ø END
1ØØ PRINT "HELLO"
11Ø RETURN
```

When this program is run it prints the word 'HELLO' one time.

1 **GOTO** expression

Abbreviations: G., GO., GOT.

See also: GOSUB, ON . . . GOSUB, ON . . . GOTO

Purpose

The GOTO verb is used to transfer control to a specified line number.

Use

The GOTO verb transfers control from one location in a BASIC program to another location. Unlike the GOSUB verb, GOTO does not "remember" the location from which the transfer occurred.

The expression in a GOTO statement may not include a comma, e.g., 'A(1, 2)' cannot be used. Since there is an ON . . . GOTO structure for choosing different destinations at given locations in the program, the expression usually consists of just the desired line number. When a numeric expression is used, it must evaluate to a valid line number, i.e., 1 to 999, or an ERROR 4 will occur.

Well designed programs usually flow simply from beginning to end, except for subroutines executed during the program. Therefore, the principal use of the GOTO verb is as a part of an IF . . . THEN statement.

Examples

```
1Ø INPUT A$
2Ø IF A$ = "Y" THEN GOTO 5Ø
3Ø PRINT "NO"
4Ø GOTO 6Ø
5Ø PRINT "YES"
6Ø END
```

This program prints 'YES' if a 'Y' is entered and prints 'NO' if anything else is entered.

1 GRAD

Abbreviations: GR., GRA.

See also: DEGREE and RADIAN

Purpose

The GRAD verb is used to change the form of angular values to gradient form.

Use

The **PC-8** has three forms for representing angular values – decimal degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The GRAD function changes the form for all angular values to gradient form until a DEGREE or RADIAN verb is used. Gradient form represents angular measurement in terms of percent gradient, i.e., a 45° angle is a 50^{g} gradient.

Examples

1Ø GRAD	
2Ø X = ASN 1	X now has a value of 1ØØ, i.e., a 100^{g} gradient, the Arcsine of 1.

1 **IF** condition **THEN** statement
2 **IF** condition statement

Abbreviations: none for IF, T., TH., THE.

Purpose

The IF . . . THEN verb pair is used to execute or not execute a statement depending on conditions at the time the program is run.

Use

In the normal running of a BASIC programs, statements are executed in the sequence in which they occur. The IF . . . THEN verb pair allows decisions to be made during execution so that a given statement is executed only when desired. When the condition part of the IF statement is true, the statement is executed; when it is False, the statement is skipped.

The condition part of the IF statement can be any relational expression as described in Chapter 4. It is also possible to use a numeric expression as a condition, although the intent of the statement will be less clear. Any expression which evaluates to zero or a negative number is considered False; any which evaluates to a positive number is considered True.

The statement which follows the THEN may be any BASIC statement, including another IF . . . THEN. If it is a LET statement, the LET verb itself must appear. Unless the statement is an END, GOTO, or ON . . . GOTO, the statement following the IF . . . THEN statement is the next one executed regardless of whether the condition is True.

The two forms of the IF statement are identical in action, but the first form is clearer.

Examples

```
1Ø INPUT "CONTINUE? "; A$
2Ø IF A$ = "YES" THEN GOTO 1Ø
3Ø IF A$ = "NO" THEN GOTO 6Ø
4Ø PRINT "YES OR NO, PLEASE"
5Ø GOTO 1Ø
6Ø END
```

This program continues to ask 'CONTINUE?' as long as 'YES' is entered; it stops if 'NO' is entered, and complains otherwise.

1 **INPUT** input list

Where:	input list	is:	input group
		or:	input group , input list
and:	input group	is:	var list
		or:	prompt , var list
		or:	prompt ; var list
and:	var list	is:	variable
		or:	variable , var list
and:	prompt	is:	any string constant

Abbrevaitions: I., IN., INP., INPU.

See also: INPUT #, READ

Purpose

The INPUT verb is used to enter one or more values from the keyboard.

Use

When you want to enter different values each time a program is run, use the INPUT verb to enter these values from the keyboard.

In its simplest form the INPUT statement does not include a prompt string, instead a question mark is displayed on the left edge of the display. A value is then entered, followed by the ENTER key. This value is assigned to the first variable in the list. If other variables are included in the same INPUT statement, this process is repeated until the list is exhausted.

If a prompt is included in the INPUT statement, the process is exactly the same except that, instead of the question mark, the prompt string is displayed at the left edge of the display. If the prompt string is followed by a semicolon, the cursor is positioned immediately following the prompt. If the prompt is followed by a comma, the prompt is displayed, then when a key is pressed the display is cleared and the first character of the input is displayed at the left edge.

When a prompt is specified and there is more than one variable in the list following it, the second and succeeding variables are prompted with the question mark. If a second prompt is included in the list, it is displayed for the variable which immediately follows it.

If the ENTER key is pressed and no input is provided, the variable retains the value it had before the INPUT statement.

Examples

1Ø INPUT A	Clears the display and puts a question mark at the left edge.
2Ø INPUT "A = "; A	Displays 'A=' and waits for input data.
3Ø INPUT "A = ", A	Displays 'A='. When data is input 'A=' disappears and the data is displayed starting at left edge.
4Ø INPUT "X = ? ";X, "Y = ? "; Y	Displays 'X=?' and waits for first input. After ENTER is pressed, display is cleared and 'Y=?' is displayed at left edge.

1 **INPUT #**
2 **INPUT #** "filename"
3 **INPUT #** var list
4 **INPUT #** "filename" ; var list

Where: var list is: variable
or: variable , var list

Abbreviations: I. #, IN. #, INP.#, INPU. #

See also: INPUT, PRINT #, READ

Purpose

The INPUT # verb is used to enter values from the cassette tape.

Use

PRINT # saves the values of variables on tape. They can then be read back into the same or another program using the INPUT # verb.

With the first form, the values are sequentially read from the tape and assigned to the 26 preallocated storage locations (fixed variables) and A ( ) variables, (A(27) ~). The transfer continues until the values recorded on tape run out or the computer memory is filled to its capacity.

With the second form, the tape is searched for the indicated filename and the variables are loaded as in the first form.

With the third form, if the variable list includes a fixed variable, values are sequentially read from the tape and assigned to the preallocated storage locations, starting at the specified variable.

With the fourth form, the tape is searched for the indicated filename and the variables are loaded as in the third form.

There is a special variable form which may be used in the variable list. It looks like an array variable except that an asterisk is enclosed in the parentheses, e.g., B(*) or F$(*). This form causes all values of the indicated variable to be restored from the tape; i.e., B (*) restores B (Ø), B (1), B (2), . . . etc., for as many values as were originally stored. You may not read a single element of an array.

Examples

1) 2Ø INPUT # A	Reads values from the current position of the tape.
2) 2Ø INPUT # A(3)	Reads values from the current position of the tape and assigns the values to the variables A(3) ~ A(26) (or C ~ Z) and A(27) ~.
3) 2Ø INPUT # "FIL2"; A$	Searches the tape for the file 'FIL2' and reads in values.
4) 2Ø INPUT # "FIL3"; G (*)	Searches the tape for the file 'FIL3' and reads in as many values of G ( ) as are available.

NOTES:

1. When the prerecorded data on tape is transferred to a variable, the data and variable should be coincident in shape (numerical or string variable), size and length. An error (ERROR 8) will result unless they are coincident in size and length. No error will occur when they are not coincident only in shape. In this case, however, the transfer of incorrect data may result when the numerical data is transferred to a string variable or the string data to a numerical variable. Therefore, the data and variable should also be coincident in shape.
2. The data transfer to variables in the fixed variables and/or in the shape of A ( ) terminates when the prerecorded data on tape is out or when the computer memory is filled to capacity.

1 **LET** variable = expression
2 variable = expression

Abbreviations: LE.

Purpose

The LET verb is used to assign a value to a variable.

Use

The LET verb assigns the value of the expression to the designated variable. The type of the expression must match that of the variable, i.e. only numeric expressions can be assigned to numeric variables and only string expressions can be assigned to string variables. In order to convert from one type to the other, one of the explicit type conversion functions, STR$ or VAL, must be used.

The LET verb may be omitted in all LET statements except those which appear in the THEN clause of an IF . . . THEN statement. In this one case the LET verb must be used.

Examples

10 I = 10	Assignes the value 10 to I.
20 A = 5*I	Assigns the value 50 to A.
30 X$ = STR$ (A)	Assigns the value '50' to X$.
40 IF I >= 10 THEN LET Y$=X$+".00"	Assigns the value '50.00' to Y$.

1 **LPRINT** print expr
2 **LPRINT** print expr , print expr
3 **LPRINT** print list

Where: print list is: print expr
or: print expr ; print list
and: print expr is: expression
or: USING clause ; expression

The USING clause is described separately under USING

Abbreviations: LP., LPR., LPRI., LPRIN.

See also: PAUSE, PRINT, USING, and WAIT

Purpose

The LPRINT verb is used to print information on the printer of the PC-3 Printer/Cassette Interface.

Use

The LPRINT verb is used to print prompting information, results of calculations, etc. The first form of the LPRINT statement prints a single value. If the expression is numeric, the value will be printed at the far right edge of the paper. If it is a string expression, the print is made starting at the far left.

With the second form of the LPRINT statement the paper is divided into two 12 character halves and the two values are printed in each half according to the same rules as above.

With the third form the print always starts at the left edge and each value is printed immediately following the previous value from left to right with no intervening space.

It is possible to cause PRINT statements to work as LPRINT statements. See the PRINT verb for details.

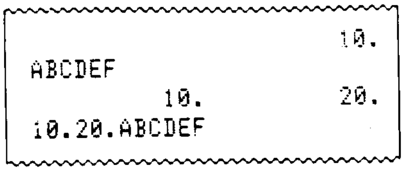
If an LPRINT statement contains more than 24 characters, the first 24 are printed on one line, the next 24 on the next line, and so forth.

Unlike PRINT, there is no halt or wait after execution of an LPRINT statement.

Examples

```
1Ø A=1Ø: B=2Ø: X$ = "ABCDEF"
2Ø LPRINT A
3Ø LPRINT X$
4Ø LPRINT A, B
5Ø LPRINT A; B; X$
```

Paper



```
                         1Ø.
ABCDEF
            1Ø.          2Ø.
1Ø.2Ø.ABCDEF
```

1 **NEXT** numeric variable

Abbreviations: N., NE., NEX.

See also: FOR

Purpose

The NEXT verb is used to mark the end of a group of statements which are being repeated in a FOR/NEXT loop.

Use

The use of the NEXT verb is described under FOR. The numeric variable in a NEXT statement must match the numeric variable in the corresponding FOR.

Examples

```
1Ø FOR I = 1 TO 1Ø                    Print the numbers from 1 to 1Ø.
2Ø PRINT I
3Ø NEXT I
```

1 **ON** expression **GOSUB** expression list

Where: expression list is: expression
or: expression , expression list

Abbreviations: O. ; GOS., GOSU.

See also: GOSUB, GOTO, ON . . . GOTO

Purpose

The ON . . . GOSUB verb is used to execute one of a set of subroutines depending on the value of a control expression.

Use

When the ON . . . GOSUB verb is executed the expression between ON and GOSUB is evaluated and reduced to an integer. If the value of the integer is 1, the first subroutine in the list is executed as in a normal GOSUB. If the expression is 2, the second subroutine in the list is executed, and so forth. After the RETURN from the subroutine execution proceeds with the statement which follows the ON . . . GOSUB.

If the expression is zero, negative, or larger than the number of subroutines provided in the list, no subroutine is executed and execution proceeds with the next line of the program.

NOTE: Commas may not be used in the expressions following the GOSUB. The **PC-8** cannot distinguish between commas **in** expressions and commas **between** expressions.

Examples

```
10 INPUT A
20 ON A GOSUB 100,200,300
30 END
100 PRINT "FIRST"
110 RETURN
200 PRINT "SECOND"
210 RETURN
300 PRINT "THIRD"
310 RETURN
```

An input of 1 prints "FIRST"; 2 prints "SECOND"; 3 prints "THIRD". Any other input does not produce any print.

1 **ON** expression **GOTO** expression list

Where: expression list is: expression
or: expression , expression list

Abbreviations: O.; G., GO., GOT.

See also: GOSUB, GOTO, ON . . . GOSUB

Purpose

The ON . . . GOTO verb is used to transfer control to one of a set of locations depending on the value of a control expression.

Use

When the ON . . . GOTO verb is executed the expression between ON and GOTO is evaluated and reduced to an integer. If the value of the integer is 1, control is transferred to the first location in the list. If the expression is 2, control is transferred to the second location in the list; and so forth.

If the expression is zero, negative, or larger than the number of locations provided in the list, execution proceeds with the next line of the program.

NOTE: Commas may not be used in the expressions following the GOTO. The **PC-8** cannot distinguish between commas **in** expressions and commas **between** expressions.

Examples

```
1Ø INPUT A
2Ø ON A GOTO 1ØØ,2ØØ,3ØØ
3Ø GOTO 9ØØ
1ØØ PRINT "FIRST"
11Ø GOTO 9ØØ
2ØØ PRINT "SECOND"
21Ø GOTO 9ØØ
3ØØ PRINT "THIRD"
31Ø GOTO 9ØØ
9ØØ END
```

An input of 1 prints 'FIRST'; 2 prints 'SECOND'; 3 prints 'THIRD'. Any other input does not produce any print.

1 **PAUSE** print expr
2 **PAUSE** print expr , print expr
3 **PAUSE** print list

Where: print list is: print expr
or: print expr ; print list
and: print expr is: expression
or: USING clause ; expression

The USING clause is described separately under USING

Abbreviations: PAU., PAUS.

See also: LPRINT, PRINT, USING, and WAIT

Purpose

The PAUSE verb is used to print information on the display for a short period.

Use

The PAUSE verb is used to display prompting information, results of calculations, etc. The operation of PAUSE is identical to PRINT except that after PAUSE the **PC-8** waits for a short preset interval of about .85 seconds and then continues execution of the program without waiting for the ENTER key or the WAIT interval.

The first form of the PAUSE statement displays a single value. If the expression is numeric, the value is printed at the far right end of the display. If it is a string expression, the display is made starting at the far left.

With the second form of the PAUSE statement the display is divided into two 8 character halves. The two values are displayed in each half according to the same rules as above.

With the third form the display starts at the left edge and each value is displayed immediately following the previous value from left to right with no intervening space.

PAUSE statements are not affected by the PRINT = LPRINT statement (see PRINT).

While it is possible to write PAUSE statements which would display more than 16 characters only the left-most 16 appear in the display. There is no way to see the other characters.

Examples

```
10 A = 10 : B = 20 : X$ = "ABCDEF"          Display

20 PAUSE A                                                    10.

30 PAUSE X$                      ABCDEF

40 PAUSE A, B                                   10.           20.

50 PAUSE A; B; X$                10.20. ABCDEF
```

1 **PRINT** print expr
2 **PRINT** print expr , print expr
3 **PRINT** print list
4 **PRINT = LPRINT**
5 **PRINT = PRINT**

Where: print list is: print expr
or: print expr ; print list
and: print expr is: expression
or: USING clause ; expression

The USING clause is described separately under USING

Abbreviations: P., PR., PRI., PRIN.

See also: LPRINT, PAUSE, USING, and WAIT

Purpose

The PRINT verb is used to print information on the display or on the printer of the **PC-3** Printer/Cassette Interface (26-3591).

Use

The PRINT verb is used to display prompting information, results of calculations, etc. The first form of the PRINT statement displays a single value. If the expression is numeric, the value is printed at the far right end of the display. If it is a string expression, the display is made starting at the far left.

With the second form of the PRINT statement the display is divided into two 8 character halves and the two values are displayed in each half according to the same rules as above.

With the third form, the display starts at the left edge and each value is displayed immediately following the previous value from left to right with no intervening space.

The fourth and fifth forms of the PRINT statement do no printing. The fourth form causes all PRINT statements which follow it in the program to be treated as if they were LPRINT statements. The fifth form resets this condition so that the PRINT statements will again work with the display.

While it is possible to write PRINT statements which would display more than

16 characters, only the left-most 16 appear in the display. There is no way to see the other characters.

Examples

```
10 A = 10 : B = 20 : X$ = "ABCDEF"                 Display

20 PRINT A                                              10.

30 PRINT X$                      ABCDEF

40 PRINT A, B                                10.        20.

50 PRINT A; B; X$                10.20.ABCDEF
```

1 **PRINT #**
2 **PRINT #** "filename"
3 **PRINT #** "var list"
4 **PRINT #** "filename" ; var list

Where: var list is: variable
or: variable , var list

Abbreviations: P. #, PR. #, PRI. #, PRIN. #

See also: INPUT #, PRINT, READ

Purpose

The PRINT # verb is used to store values on the cassette tape.

Use

Using PRINT # the values of variables can be saved on tape. These can then be read back into the same or another program using the INPUT # verb.

With the first form, the values of the 26 preallocated variables (variables A ~ Z and A$ ~ Z$), A( ) and A$( ) variables are stored on the tape.
Note: Variables A ~ Z and A(1) ~ A(26) are the same.

With the second form, the values are stored on the tape as in the first form under the designated filename.

With the third form, if the variable list includes a fixed variable, the values of the fixed variables starting from the specified variable are saved on tape.

With the fourth form, the values are stored on the tape as in the third form under the designated filename.

There is a special variable form which may be used in the variable list. It looks like an array variable except that an asterisk is enclosed in the parentheses, e.g., B(*) or F$(*). This form causes all values of the indicated variable to be saved on the tape, i.e., B (*) saves B (Ø), B (1), B (2), . . . etc., for as many values as are in the array. You may not save a single element of an array.

Examples

1)	1Ø PRINT # A	Saves values on the tape at the current position.
2)	1Ø PRINT # "FIL2"; A$	Saves values on the tape under the filename 'FIL2'.
3)	1Ø PRINT # "FIL3"; G (*)	Saves values of G ( ) on the tape under the filename 'FIL3'.

Note:
A variable above A(27) or a dimensional variable must be secured into the program/data area before the PRINT #command is executed. If the variable is not designated before the PRINT #command, an error (ERROR 3) will result.

1 RADIAN

Abbreviations: RAD., RADI., RADIA.

See also: DEGREE and GRAD

Purpose

The **RADIAN verb** is used to change the form of angular values to radian form.

Use

The **PC-8** has three forms for representing angular values – decimal degrees, radians, and gradient. These forms are used in specifying the arguments to the SIN, COS, and TAN functions and in returning the results from the ASN, ACS, and ATN functions.

The RADIAN function changes the form for all angular values to radian form until a DEGREE or GRAD verb is used. Radian form represents angles in terms of the length of the arc with respect to a radius, i.e., 36Ø° is 2 PI radians since the circumference of a circle is 2 PI times the radius.

Examples

```
1Ø RADIAN
2Ø X = ASN 1        X now has a value of 1.57Ø796327 or PI/2, the Arcsine of 1
```

1 RANDOM

Abbreviations: RA., RAN., RAND., RANDO.

Purpose

The **RANDOM** verb is used to reset the seed for random number generation.

Use

When random numbers are generated using the RND, function, the **PC-8** begins with a predetermined "seed" or starting number. The RANDOM verb resets this seed to a new randomly determined value.

The starting seed will be the same each time the **PC-8** is turned on, so the sequence of random numbers generated with RND is the same each time, unless the seed is changed. This is very convenient during the development of a program because it means that the behavior of the program should be the same each time it is run even though it includes a RND function. When you want the numbers to be truly random, the RANDOM statement can be used to make the seed itself random.

Examples

```
1Ø RANDOM
2Ø X = RND 10
```

When run from line 2Ø, the value of X is based on the standard seed. When run from line 1Ø, a new seed is used.

1 **READ** variable list

Where: variable list is: variable
or: variable , variable list

Abbreviations: REA.

See also: DATA, RESTORE

Purpose

The READ verb is used to read values from a DATA statement and assign them to variables.

Use

When assigning initial values to an array, it is convenient to list the values in a DATA statement and use a READ statement in a FOR . . . NEXT loop to load the values into the array. When the first READ is executed, the first value in the first DATA statement is returned. Succeeding READs use succeeding values in the sequential order in which they appear in the program, regardless of how many values are listed in each DATA statement or how many DATA statements are used.

If desired, the values in a DATA statement can be read a second time by using the RESTORE statement.

Examples

```
1Ø DIM B(1Ø)            Sets up an array
2Ø FOR I = 1 TO 1Ø
3Ø READ B(I)            Loads the values from the DATA statement into
4Ø NEXT I               B( ) — B(1) is 1, B (2) is 2, B (3) is 3, etc.
5Ø DATA 1, 2, 3, 4, 5, 6
6Ø DATA 7, 8, 9, 1Ø
```

```
1 REM remark

Abbreviations: none
```

Purpose

The **REM verb** is used to include comments in a program.

Use

Often it is useful to include explanatory comments in a program. These can provide titles, names of authors, dates of last modification, usage notes, reminders about algorithms used, etc. These comments are included by means of the REM statement.

The REM statement has no effect on the program execution and can be included anywhere in the program. Everything following the REM verb in that line is treated as a comment, so the REM verb must be the last statement in a line when multiple statement lines are used.

Examples

```
10 REM THIS LINE HAS NO EFFECT
```

1 **RESTORE**
2 **RESTORE** expression

Abbreviations: RES., REST., RESTO., RESTOR.

See also: DATA, READ

Purpose

The RESTORE verb is used to re-read values in a DATA statement or to change the order in which these values are read.

Use

In the regular use of the READ verb the **PC-8** begins reading with the first value in a DATA statement and proceeds sequentially through the remaining values. The first form of the RESTORE statement resets the pointer to the first value of the first DATA statement, so that it can be read again. The second form of the RESTORE statement resets the pointer to the first value of the first DATA statement whose line number is greater than the value of the expression.

Examples

```
1Ø DIM B(1Ø)           Sets up an array
2Ø FOR I = 1 TO 1Ø
3Ø READ B(I)           Assigns the value 1Ø to each of the elements of B( ).
4Ø RESTORE
5Ø NEXT I
6Ø DATA 1Ø
```

Note: The RESTORE verb must be written at the beginning of the line (just after the line number). It cannot be used with a colon (:) following another statement.

1 RETURN

Abbreviations: RE., RET., RETU., RETUR.

See also: GOSUB, ON . . . GOSUB

Purpose

The RETURN verb is used at the end of a subroutine to return control to the statement following the originating GOSUB.

Use

A subroutine may have more than one RETURN statement, but the first one executed terminates the execution of the subroutine. The next statement executed will be the one following the GOSUB or ON . . . GOSUB which calls the subroutine. If a RETURN is executed without a GOSUB, an Error 5 will occur.

Examples

```
1Ø GOSUB 1ØØ
2Ø END
1ØØ PRINT "HELLO"
11Ø RETURN
```

When run this program prints the word "HELLO" once.

1 STOP

Abbreviations: S., ST., STO.,

See also: END; CONT command

Purpose

The STOP verb is used to halt execution of a program for diagnostic purposes.

Use

When the **STOP** verb is encountered in program execution the **PC-8** execution halts and a message is displayed such as 'BREAK IN **200**' where **200** is the number of the line containing the STOP. STOP is used during the development of a program to check the flow of the program or examine the state of variables. Execution may be restarted using the CONT command.

Examples

10 STOP	Causes "BREAK IN 10" to appear in the display.

1 TROFF

Abbreviations : TROF.

See also: TRON

Purpose

The TROFF verb is used to cancel the trace mode.

Use

Execution of the TROFF verb restores normal execution of the program.

Examples

```
1Ø TRON
2Ø FOR I = 1 TO 3
3Ø NEXT I
4Ø TROFF
```

When run, this program displays the line numbers 1Ø, 2Ø, 3Ø, 3Ø, 3Ø and 4Ø as the [↓] is pressed. By pressing the [↑] , you can review the line.

1 TRON

Abbreviations : TR., TRO.

See also: TROFF

Purpose

The TRON verb is used to initiate the trace mode.

Use

The trace mode provides assistance in debugging programs. When the trace mode is on, the line number of each statement is displayed **after** each statement is executed. The **PC-8** then halts and waits for the Down Arrow key to be pressed before moving on to the next statement. The Up Arrow key may be pressed to see the statement which has just been executed. The trace mode continues until a TROFF verb is executed.

Examples

```
1Ø TRON
2Ø FOR I = 1 TO 3
3Ø NEXT I
4Ø TROFF
```

When run this program displays the line numbers 1Ø, 2Ø, 3Ø, 3Ø, 3Ø and 4Ø as the [↓] is pressed. By pressing the [↑] you can review the line.

1 **USING**
2 **USING** "editing specification"
3 **USING** character variable

Abbreviations: U., US., USI., USIN.

See also: LPRINT, PAUSE, PRINT
Further guide to the use of USING is provided in Appendix C

Purpose

The USING verb is used to control the format of displayed or printed output.

Use

The USING verb can be used by itself or as a clause within a LPRINT, PAUSE, or PRINT statement. The USING verb establishes a specified format for output which is used for all output which follows until changed by another USING verb.

The editing specification of the USING verb consists of a quoted string composed of some combination of the following editing characters:

- # Right justified numeric field character
- · Decimal point.
- ^ Used to indicate that numbers should be displayed in scientific notation.
- & Left justified alphanumeric field.

For example, "####" is an editing specification for a right justified numeric field with room for 3 digits and the sign. In numeric fields, a location must be included for the sign, even if it will always be positive.

Editing specifications may include more than one field. For example "####&&&&" could be used to print a numeric and a character field next to each other.

If the editing specification is missing, as in format 1, special formatting is turned off and the built-in display rules pertain.

Examples

```
                                            Display
1Ø A = 125  :  X$ = "ABCDEF"

2Ø PRINT USING "##.## ^";A              1. 25E  Ø2

3Ø PRINT  USING  "&&&&&&&&";X$          ABCDEF

4Ø PRINT USING "####&&&"; A; X$          125ABC
```

Notes: 1. When the total number of digits specified with USING exceeds 16 for "PRINT **expression**", ERROR 7 results.

2. When the number of digits for the integer part (sign and decimal point included) exceeds 8 while using the fixed decimal point system for "PRINT **expression** , **expression**", ERROR 7 results.
When the character string of the expression in the form of "PRINT **expression** , **expression**" exceeds 8 columns, the excess part is not displayed.

3. When the display contents of the form "PRINT **expression** ; **expression**" exceeds 16 columns, the excess part is not displayed.

1 **WAIT**
2 **WAIT** expression

Abbreviations: W., WA., WAI.

See also: PAUSE, PRINT

Purpose

The WAIT verb is used to control the length of time that displayed information is shown before program execution continues.

Use

In normal execution the **PC-8** halts execution after a PRINT command until the [ENTER] key is pressed. The WAIT command causes the **PC-8** to display for a specified interval and then proceed automatically (similar to the PAUSE verb). The expression which follows the WAIT verb determines the length of the interval. The interval may be set to any value from Ø to 65535. Each increment is about one sixty-fourth of a second. WAIT Ø is too fast to be read reasonably; WAIT 65535 is about 17 minutes. WAIT with no following expression resets the **PC-8** to the original condition of waiting until the [ENTER] key is pressed.

Examples

1Ø WAIT 64	Causes PRINT to wait about 1 second.

FUNCTIONS

Pseudovariables

Pseudovariables are a group of functions which take no argument and are used like simple variables wherever required.

1 INKEY$

INKEY$ is a string pseudovariable which gives to the specified variable the value of the key pressed while the INKEY$ function is executed. INKEY$ is used to respond to the pressing of individual keys without waiting for the ENTER key to end the input. The computer just keeps "circling"until it receives a message from the key board.

```
1Ø: A$ = INKEY$
2Ø: B = ASC A$
3Ø: IF B = Ø THEN GOTO 1Ø
4Ø: PRINT B
```

Note: ENTER , SHIFT , DEF , ↑ , ↓ , ▶ , ◀ , MODE , and CL all have a value of NUL.

1 MEM

MEM is a numeric pseudovariable which has the value of the number of characters of program memory remaining. The available program memory will be the total memory less the space consumed by programs and array variables. MEM may also be used as a command. Immediately after reset (Set the mode to PRO and enter NEW ENTER), MEM has a value of 1278 bytes.

1 PI

PI is a numeric pseudovariable which has the value of PI. It is identical to the use of the special PI character (π) on the keyboard. Like other numbers the value of PI is kept to 1Ø digit accuracy (3.141592654).

Numeric Functions

Numeric functions are a group of mathematical operations which take a single numeric value and return a numeric value. They include trigonometric functions, logarithmic functions, and functions which operate on the integer and sign parts of a number. Many dialects of BASIC require that the argument to a function be enclosed in parentheses. The **PC-8** does not require these parentheses, except when it is necessary to indicate what part of a more complex expression is to be included in the argument.

LOG 1ØØ + 1ØØ will be interpreted as:

(LOG 1ØØ) + 1ØØ   not   LOG (1ØØ + 1ØØ).

1 **ABS** numeric expression

ABS is numeric function which returns the absolute value of the numeric argument. The absolute value is the value of a number without regard to its sign. ABS −1Ø is 1Ø.

1 **ACS** numeric expression

ACS is a numeric function which returns the arccosine of the numeric argument. The arccosine is the angle whose cosine is equal to the expression. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. ACS .5 is 6Ø in the decimal degree mode.

1 **ASN** numeric expression

ASN is a numeric function which returns the arcsine of the numeric argument. The arcsine is the angle whose sine is equal to the expression. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. ASN .5 is 3Ø in the decimal degree mode.

1 **ATN** numeric expression

ATN is a numeric function which returns the arctangent of the numeric argument. The arctangent is the angle whose tangent is equal to the expression. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. ATN 1. is 45 in the decimal degree mode.

1 **COS** numeric expression

COS is a numeric function which returns the cosine of the angle argument. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. **COS 6Ø** is .5 in the decimal degree mode.

1 **DEG** numeric expression

The DEG function converts an angle argument in DMS (Degree, Minute, Second) format to DEG (Decimal Degree) form. In DMS format the integer portion of the number represents the degrees, the first and second digits of the decimal represent the minutes, the third and forth digits of the decimal represent the seconds, and any further digits represent decimal seconds. For example, 55° 1Ø′ 44.5″ is represented as 55.1Ø445. In DEG format the integer portion is degrees and the decimal portion is decimal degrees. DEG 55.1Ø445 is 55.179Ø2778.

1 **DMS** numeric expression

DMS is a numeric function which converts an angle argument in DEG format to DMS format (see DEG). DMS 55.179Ø2778 is 55.1Ø445.

1 **EXP** numeric expression

EXP is a numeric function which returns the value of e (2.718281828 – the base of the natural logarithms) raised to the value of the numeric argument. **EXP 1** is 2.718281828.

1 **INT** numeric expression

INT is a numeric function which returns the integer part of its numeric argument. INT PI is 3.

1 **LOG** numeric expression

LOG is a numeric function which returns the logarithm to the base 1Ø of its numeric argument. LOG 1ØØ is 2.

1 **LN** numeric expression

LN is a numeric function which returns the logarithm to the base e (2.718281828) of its numeric argument. LN 1ØØ is 4.6Ø517Ø186.

1 **RND** numeric expression

RND is a numeric function which generates random numbers. If the value of the argument is less than one but greater than or equal to zero, the random number is less than one and greater than or equal to zero. If the argument is an integer greater than or equal to 1, the result is a random number greater than or equal to 1 and less than or equal to the argument. If the argument is greater than or equal to 1 and not an integer, the result is a random number greater than or equal to 1 and less than or equal to the smallest integer which is larger than the argument: (In this case, the generation of the random number changes depending on the value of the decimal portion of the argument.):

	··········· Result ···········	
Argument	Lower Bound	Upper Bound
.5	Ø<	<1
2	1	2
2.5	1	3

The same sequence of random numbers is normally generated because the same "seed" is used each time the **PC-8** is turned on. To randomize the seed, see the RANDOM verb.

1 **SGN** numeric expression

SGN is a numeric function which returns a value based on the sign of the argument. If the argument is positive, the result is 1; if the argument is zero, the result is Ø; if the argument is negative, the result is −1. SGN −5 is −1.

1 **SIN** numeric expression

SIN is a numeric function which returns the sine of the angle argument. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. SIN 3Ø is .5 in the decimal degree mode.

1 **SQR** numeric expression

SQR is a numeric function which returns the square root of its agrument. It is identical to the use of the special square root symbol (√‾) on the keyboard. SQR 4 is 2.

1 **TAN** numeric expression

TAN is a numeric function which returns the tangent of its angle argument. The value returned depends on whether the **PC-8** is in decimal degree, radian, or gradient mode for angles. TAN 45 is 1 in the decimal degree mode.

String Functions

String functions are a group of operations used for manipulating strings. Some take a string argument and return a numeric value. Some take a string argument and return a string. Some take a numeric value and return a string. Some take a string argument and one or two numeric arguments and return a string. Many dialects of BASIC require the argument of a function to be enclosed in parentheses. The **PC-8** does not require these parentheses, except when it is necessary to indicate what part of a more complex expression is to be included in the argument. String functions with two or three arguments all require the parentheses.

1 **ASC** string expression

ASC is a string function which returns the numeric ASCII code value of the first character in its argument. The chart of ASCII codes and their relationship to characters is given in Appendix B. ASC "A" is 65.

1 **CHR$** numeric expression

CHR$ is a string function which returns the character which corresponds to the numeric ASCII code of its argument. The chart of ASCII codes and their relationship to characters is given in Appendix B. CHR$ 65 is "A".

1 **LEFT$** (string expression, numeric expression)

LEFT$ is a string function which returns the leftmost part of the string first argument: The number of characters returned is determined by the numeric expression. LEFT$ ("ABCDEF", 2) is "AB".

1 **LEN** string expression

LEN is a string function which returns the length of the string argument. LEN "ABCDEF" is 6.

1 **MID$** (string expression , num. exp. 1, num. exp. 2)

MID$ is a string function which returns a middle portion of the string first argument. The first numeric argument indicates the first character position to be included in the result. The second numeric argument indicates the number of characters that are to be included. MID$ ("ABCDEF", 2, 3) is "BCD".

1 **RIGHT$** (string expression , numeric expression)

RIGHT$ is a string function which returns the rightmost part of the string first argument. The number of characters returned is determined by the numeric argument. RIGHT$ ("ABCDEF", 3) is "DEF".

1 **STR$** numeric expression

STR$ is a string function which returns a string which is the character representation of its numeric argument. It is the reverse of VAL. **STR$ 1.59** is **"1.59"**.

1 **VAL** string expression

VAL is a string function which returns the numeric value of its string argument. It is the reverse of STR$. The VAL of a non-number is zero. VAL "1.59" is 1.59.

Note: The character-string convertible by VAL function to numerical value consists of numerals (0 to 9), symbols (+ and –) and a symbol (**E**) indicating an exponential portion. No other characters and symbols are included. If a character-string includes other characters and symbols, any character-string on the right of that character-string will be ignored. If included in a character-string, a space is usually regarded as non-existing. If, however, a space is included in the portion (on the right of **E**) corresponding to an exponential part, any character-string on the right of the space will be ignored.

CHAPTER 9 TROUBLESHOOTING

This chapter provides you with some hints on what to do when your **PC-8** does not do what you expect it to do. It is divided into two parts — the first part deals with general machine operation and the second with BASIC programming. For each problem there are a series of suggestions provided. You should try each of these, one at a time, until you have fixed the problem.

Machine Operation

If:	**Then You Should:**
You turn on the machine but there is nothing on the display	1. Check to see that the slide switch is set to ON. 2. Push the ON BRK key to see if AUTO POWER OFF has been activated. 3. Replace the batteries.
There is a display, but no response to keystrokes	1. Press CL key to clear. 2. Press CA (SHIFT CL) to clear. 3. Turn OFF and ON again. 4. Hold down any key and push RESET. 5. Push RESET without any key.
You have typed in a calculation or answer and get no response	1. Push ENTER.
You are running a BASIC program and it displays something, and stops	1. Push ENTER.
You enter a calculation and it is displayed in BASIC statement format (colon after the first number)	1. Change the mode from PROgram to RUN for calculations.
You get no response from any keys.	1. Hold down any key and push RESET. 2. If you get no response from any key even when the above operation is performed, enter NEW Ø ENTER . (at PRO mode) This will clear the program, data and all reserved contents.

BASIC Debugging

When entering a new BASIC program, it is usual for it **not** to work the first time. Even if you are simply keying in a program that you know is correct, such as those provided in this manual, it is usual to make at least one typing error. If it is a new program of any length, it will probably contain at least one logic error as well. Following are some general hints on how to find and correct your errors.

You run your program and get an error message:

1. Go back to the PROgram mode and use the ↑ or the ↓ keys to recall the line with the error. The cursor will be positioned at the place in the line where the **PC-8** got confused.
2. If you can't find an obvious error in the way in which the line is written, the problem may lie with the values which are being used. For example, CHR$ (A) will produce an error if A has a value of 1 because CHR$ (1) is an illegal character. Check the values of the variables in either the RUN or the PROgram mode by typing in the name of the variable followed by ENTER.

You RUN the program and don't get an error message, but it doesn't do what you expect.

3. Check through the program line by line using LIST and the ↓ and ↑ keys to see if you have entered the program correctly. It is surprising how many errors can be fixed by just taking another look at the program.
4. Think about each line as you go through the program as if you were the computer. Take sample values and try to apply the operation in each line to see if you get the result that you expected.
5. Insert one or more extra PRINT statements in your program to display key values and key locations. Use these to isolate the parts of the program that are working correctly and the location of the error. This approach is also useful for determining which parts of a program have been executed. You can also use STOP to temporarily halt execution at critical points so that several variables can be examined.
6. Use TRON and TROFF, either as commands or directly within the program to trace the flow of the program through individual lines. Stop to examine the contents of critical variables at crucial points. This is a very slow way to find a problem, but sometimes it is also the only way.

CHAPTER 1Ø
MAINTENANCE

To insure trouble-free operation of your **PC-8** we recommend the following:

* Always handle the **PC-8** carefully as the liquid crystal display is made of glass.

* Keep the **PC-8** in an area free from extreme temperature changes, moisture, or dust. During warm weather, vehicles left in direct sunlight are subject to high temperature build up. Prolonged exposure to high temperature may cause damage to your **PC-8**.

* Use only a soft, dry cloth to clean the **PC-8**. Do not use solvents, water, or wet cloths.

* To avoid battery leakage, remove the batteries when the **PC-8** will not be in use for an extended period of time.

* The **PC-8** has special soft keys. To avoid scratching the keys, do not push them with hard, sharp objects.

* If service is required, the computer should only be returned to an authorized Radio Shack Service Center.

* If the **PC-8** is subjected to strong static electricity or external noise it may "hang up" (all keys become inoperative). If this occurs, press the RESET button while holding down any key. (See Troubleshooting).

* Keep this manual for further reference.

APPENDIX A
ERROR MESSAGES

There are nine different error codes built into the **PC-8**. The following table will explain these codes.

Error Number **Meaning**

1 Syntax error.

- This means that the **PC-8** cannot understand what you have entered. Check for things such as semicolons on the ends of PRINT statements, misspelled words, and incorrect usages.

```
3 */2
```

2 Calculation error.

Here you have probably done one of three things:

1. Tried to use too large a number.
   Calculation results are greater than 9.999999999E 99.
2. Tried to divide by zero.

```
5 / Ø
```

3. An illogical calculation has been attempted.

```
LN −3Ø      or      ASN 1.5
```

3 DIMension error/Argument error.

- Array variable already exists.

  Array specified without first dimensioning it.

  Array subscript exceeds size of array specified in DIM statement.

```
DIM B(256)
```

- Illegal function argument. This means that you have tried to make the **PC-8** do something that it just cannot handle. An example is specifying a top limit for a FOR . . . NEXT loop that is greater than 32767. The reason for the error in this case is that the top limit for a FOR . . . NEXT loop is stored in just two bytes. The maximum, positive, signed integer value that two bytes can hold is 32767.

```
1Ø FOR A = 1 TO 32768
```

4 Line Number error.

Here you have probably done one of two things:

1. Tried to use an unexsisting line number by the GOTO, GOSUB, RUN, LIST or THEN etc.
2. Tried to use too large a line number. The maximum line number is 999.

5 Nesting error.

Subroutine nesting exceeds 10 levels.

FOR loop nesting exceeds 5 levels.

RETURN verb without a GOSUB, NEXT verb without a FOR, or READ verb without a DATA.

Buffer space exceeded.

6 Memory Overflow.

Generally this error happens when you've tried to DIMension an array that is too big for memory. This can also happen when a program becomes too large.

7 PRINT USING error.

This means that you have put an illegal format specifier into a USING statement.

8 I/O device error.

This error can happen only when you have the optional printer and/or cassette recorder connected to the **PC-8**. It means that there is a problem with communication between the I/O device and the **PC-8**.

9 Other errors.

This code will be displayed whenever the computer has a problem that is not covered by one of the other eight error codes. One of the most common causes for this error is trying to access data in a variable is one fashion (e.g. A$) while the data was originally stored in the variable in another fashion (e.g. A).

APPENDIX B
ASCII CHARACTER CODE CHART

The following chart shows the conversion values for use with CHR$ and ASC. The column shows the first hex character or the first four binary bits, the row shows the second hex character or the second binary bits. The upper left corner of each box contains the decimal number for the character. The lower right shows the character. If no character is shown then it is an illegal character on the **PC-8**. For example, the character 'A' is a decimal 65 or a hex 41 or a binary Ø1ØØØØØ1.

First 4 bits

The **PC-8** does not recognize codes in the shaded area. If you enter a code number in the shaded area, an error will result.

Second 4 Bits

Hex Binary	Ø ØØØØ	1 ØØØ1	2 ØØ1Ø	3 ØØ11	4 Ø1ØØ	5 Ø1Ø1	6 Ø11Ø	7 Ø111
Ø ØØØØ	Ø NUL	16	32 SPACE	48 Ø	64 @	8Ø P	96 Ɇ	112
1 ØØØ1	1	17	33 !	49 1	65 A	81 Q	97	113
2 ØØ1Ø	2	18	34 "	5Ø 2	66 B	82 R	98	114
3 ØØ11	3	19	35 #	51 3	67 C	83 S	99	115
4 Ø1ØØ	4	2Ø	36 $	52 4	68 D	84 T	1ØØ	116
5 Ø1Ø1	5	21	37 %	53 5	69 E	85 U	1Ø1	117
6 Ø11Ø	6	22	38 &	54 6	7Ø F	86 V	1Ø2	118
7 Ø111	7	23	39 ⊐	55 7	71 G	87 W	1Ø3	119
8 1ØØØ	8	24	4Ø (	56 8	72 H	88 X	1Ø4	12Ø
9 1ØØ1	9	25	41)	57 9	73 I	89 Y	1Ø5	121
A 1Ø1Ø	1Ø	26	42 *	58 :	74 J	9Ø Z	1Ø6	122
B 1Ø11	11	27	43 +	59 ;	75 K	91 √‾	1Ø7	123
C 11ØØ	12	28	44 ,	6Ø <	76 L	92 ¥	1Ø8	124
D 11Ø1	13	29	45 –	61 =	77 M	93 π	1Ø9	125
E 111Ø	14	3Ø	46 .	62 >	78 N	94 ^	11Ø	126
F 1111	15	31	47 /	63 ?	79 O	95 _	111	127

APPENDIX C
FORMATTING OUTPUT

It is sometimes important or useful to control the format as well as the content of output. The **PC-8** controls display formats with the USING verb. This verb allows you to specify:

- The number of digits
- The location of the decimal point
- Scientific notation format
- The number of string characters

These different formats are specified with an "output mask". This mask may be a string constant or a string variable:

```
1Ø: USING "####"
2Ø: M$ = "&&&&&&"
3Ø: USING M$
```

When the USING verb is used with no mask, all special formatting is cancelled.

```
4Ø: USING
```

A USING verb may also be used within a PRINT statement:

```
5Ø: PRINT USING M$; N
```

Wherever a USING verb is used, it will control the format of all output until a new USING verb is encountered.

Numeric Masks

A numeric USING mask may only be used to display numeric values, i.e., numeric constants or numeric variables. If a string constant or variable is displayed while a numeric USING mask is in effect, the mask will be ignored. A value which is to be displayed must always fit within the space provided by the mask. The mask must reserve space for the sign character, even when the number will always be positive. Thus a mask which shows four display positions may only be used to display numbers with three digits.

Specifying Number of Digits

The desired number of digits is specified using the '#' character. Each '#' in the mask reserves space for one digit. The display or print always contains as many characters as are designated in the mask. The number appears to the far right of this field; the remaining positions to the left are filled with spaces. Positive numbers therefore always have at least one space at the left of the field. Since the **PC-8** maintains a maximum of 1Ø significant digits, no more than 11 '#' characters should be used in a numeric mask.

> **NOTE:** In all examples in this appendix the beginning and end of the displayed field will be marked with a ' | ' character to show the size of the field.

Statement	Display
1Ø: USING "####"	(Set the **PC-8** to the RUN position, type RUN, and press ENTER .)
2Ø: PRINT 25	| 25|
3Ø: PRINT –35Ø	|–35Ø|
4Ø: PRINT 1ØØØ	ERROR 7 IN 4Ø

Notice that the last statement produced an error because 5 positions (4 digits and a sign space) were required, but only 4 were provided in the mask.

Specifying a Decimal Point

A decimal point character, '.', may be included in a numeric mask to indicate the desired location of the decimal point. If the mask provides fewer significant decimal digits than are required for the value to be displayed, the remaining positions to the right will be filled with zeros. If there are more significant decimal digits in the value than in the mask, the extra digits will be truncated (**not** rounded):

Statement	Display
1Ø: USING "####.##"	
2Ø: PRINT 25	| 25.ØØ|
3Ø: PRINT –35Ø.5	|–35Ø.5Ø|
4Ø: PRINT 2.547	| 2.54|

Specifying Scientific Notation

A " ^ " character may be included in the mask to indicate that the number is to be displayed in scientific notation. The '#' and '.' characters are used in the mask to specify the format of the "characteristic" portion of the number, i.e., the part which is displayed to the left of the E. Two '#' characters should always be used to the left of the decimal point to provide for the sign character and one integer digit. The decimal point may be included, but is not required. Up to 9 '#' characters may appear to the right of the decimal point. Following the characteristic portion, the exponentiation character, E, will be displayed followed by one position for the sign and two positions for the exponent. Thus, the smallest scientific notation field would be provided by a mask of "##^" which would print numbers of the form ' 2 E 99'. The largest scientific notation field would be "##.#########^" which would print numbers such as '−1.234567890 E −12':

```
Statement                        Display

1Ø: USING "###.##^"

2Ø: PRINT 2                      | 2.ØØE ØØ|

3Ø: PRINT −365.278               |−3.65E Ø2|
```

Specifying Alphanumeric Masks

String constants and variables are displayed using the '&' character. Each '&' indicates one character in the field to be displayed. The string will be positioned at the left end of this field. If the string is shorter than the field, the remaining spaces to the right will be filled with spaces. If the string is longer than the field, the string will be truncated to the length of the field:

```
Statement                        Display

1Ø: USING "&&&&&&"

2Ø: PRINT "ABC"                  |ABC   |

3Ø: PRINT "ABCDEFGHI"            |ABCDEF|
```

Mixed Masks

In most applications a USING mask will contain either all numeric or all string formatting characters. Both may be included in one USING mask, however, for certain purposes. In such cases, each switch from numeric to string formatting characters or vice versa marks the boundary for a different value. Thus, a mask of "#####&&&&" is a specification for displaying two separate values – a numeric value which is allocated 5 positions and a string value which is allocated 4 positions:

```
Statement                                          Display

1Ø: PRINT USING "###.##&&"; 25; "CR"    | 25. ØØCR|

2Ø: PRINT –5.789; "DB"                  | –5. 78DB|
```

Remember: Once specified, a USING format is used for all output which follows until cancelled or changed by another USING verb.

APPENDIX D EXPRESSION EVALUATION AND OPERATOR PRIORITY

When the **PC-8** is given a complex expression, it evaluates the parts of the expression in a sequence which is determined by the priority of the individual parts of the expression. If you enter the expression:

1ØØ / 5+45

as either a calculation or as a part of a program, the **PC-8** does not know whether you meant.

$$\frac{1ØØ}{5+45} = 2 \qquad \text{or} \qquad \frac{1ØØ}{5} + 45 = 65$$

Since the **PC-8** must have some way to decide between these options, it uses its rules of operator priority. Because division has a higher "priority" than addition (see the next page), it will choose to do the division first and then the addition, i.e., it will choose the second option and return a value of 65 for the expression.

Operator Priority

Operators on the **PC-8** are evaluated with the following priorities from highest to lowest:

1. Parentheses
2. Variables and Pseudovariables
3. Exponentiation (^) when preceded by a multiplication which omits the operator
4. Multiplication which omits the operator
5. Functions
6. Exponentiation (^)
7. Unary minus, negative sign (–)
8. Multiplication and division (*, /)
9. Addition and subtraction (+, –)
1Ø. Relational operators (<, <=, =, <>, >=, >)
11. Logical operators (AND, OR)

The fourth item refers to usage such as 2A or 5C(2) in which a multiplication operator is implied, but not shown. The third refers to the combination of this with exponentiation, such as 3A^3 or 5D^1.5. In these combined cases the exponentiation will be done first and the multiplication second.

When there are two or more operators at the same priority level the expression will be evaluated from left to right. (The exponentiation will be evaluated from right to left). Note that with A+B–C, for example, the answer is the same whether the addition or the subtraction is done first.

When an expression contains multiple nested parentheses, the innermost set is evaluated first and evaluation then proceeds outward.

Sample Evaluation

Starting with the expression:

```
( (3+5–2)*6+2) / 1Ø ^ LOG 1ØØ
```

The **PC-8** would first evaluate the innermost set of parentheses. Since '+' and '–' are at the same level it would move from left to right and would do the addition first:

```
( (8–2)*6+2) / 1Ø ^ LOG 1ØØ
```

Then it would do subtraction:

```
( (6)*6+2) / 1Ø ^ LOG 1ØØ
```

or:

```
(6*6+2) / 1Ø ^ LOG 1ØØ
```

In the next set of parentheses it would do the multiplication first:

```
(36+2) / 1Ø ^ LOG 1ØØ
```

And then the addition:

```
(38) / 1Ø^LOG 1ØØ
```

or:

```
38 / 1Ø ^ LOG 1ØØ
```

Now that the parentheses are cleared, the LOG function has the highest priority so it is done next:

```
38 / 1Ø ^ 2
```

The exponentiation is done next:

```
38 / 1ØØ
```

And last of all the division is performed:

```
Ø.38
```

This is the value of the expression.

APPENDIX E
FEATURE COMPARISON OF THE PC-1, PC-2, PC-3, AND PC-8

The four **Tandy** pocket computers, the **PC-1**, the **PC-2**, the **PC-3**, and the **PC-8** have many features in common, but there are some significant differences. Sometimes the same features are present, yet act in a slightly different fashion. In order to facilitate the use of programs on different models the following comparison charts are provided.

Verbs and Commands

In the following chart the symbol:

M	indicates that the feature can only be used in manual execution, i.e., as a command;
P	indicates that the feature can only be used within a program;
B	indicates that the feature can be used in both contexts.

When no symbol is shown, the feature is not available on that machine

	PC-1	PC-2	PC-3	PC-8	Comments
AREAD	P	P	P	P	See Note 1
ARUN		P			
BEEP	P	B	B		PC-2 has tone and duration
CHAIN	P	P	P	P	
CLEAR	B	B	B	B	
CLOAD	M	M	M	M	
CLOAD?	M	M	M	M	
CLS		B			
COLOR		B			
CONT	M	M	M	M	
CSAVE	M	B	B	B	
CSIZE		B			
CURSOR		B			
DEGREE	B	B	B	B	
DATA		P	P	P	

Verbs and Commands (continued)

	PC-1	PC-2	PC-3	PC-8	Comments
DEBUG	M				
DIM		B	B	B	
END	P	P	P	P	
FOR...TO...STEP	P	P	P	P	
GOSUB	P	P	P	P	
GOTO	P	B	B	B	
GCURSOR		B			
GPRINT		B			
GRAD	B	B	B	B	
GRAPH		B			
IF...THEN	P	P	P	P	
INPUT	P	P	P	P	
INPUT #	B	B	B	B	
LET	P	P	P	P	
LF		B			
LINE		B			
LIST	M	M	M	M	
LLIST		M	M	M	PC-1 can emulate with LIST
LOCK		B			
LPRINT		B	P	P	See Note 2
MERGE	M	M	M	M	
NEW	M	M	M	M	
NEXT	P	P	P	P	
ON...ERROR		P			
ON...GOSUB		P	P	P	
ON...GOTO		P	P	P	
PAUSE	P	P	P	P	
PASS			M	M	
PRINT	P	B	P	P	See Note 2
PRINT #	B	B	B	B	
RADIAN	B	B	B	B	
RANDOM		B	B	B	
READ		P	P	P	
REM	P	P	P	P	

Verbs and Commamds (continued)

	PC-1	PC-2	PC-3	PC-8	Comments
RESTORE		P	P	P	
RETURN	P	P	P	P	
RLINE		B			
RMTOFF		B			
RMTON		B			
ROTATE		B			
RUN	M	M	M	M	
SORGN		B			
STOP	P	P	P	P	
TAB		B			
TEST		B			
TEXT		B			
TROFF		B	B	B	
TRON		B	B	B	
UNLOCK		B			
USING	P	B	B	B	See Note 3
WAIT		B	B	B	

Note 1: There are some minor differences between the **PC-8** and the **PC-1** in the behavior of AREAD following PRINT, but these are unlikely to cause problems in ordinary usage.

Note 2: Add PRINT = LPRINT and PRINT = PRINT statements to **PC-1** programs to achieve the desired results on the **PC-8**.

Note 3: On the **PC-1** the USING format applies to all displays on the line in which the USING clause appears, even if the variable precedes the verb. On the other models, the USING format applies only to displays which follow the verb and remains in effect until cancelled by another USING verb.

Example:

```
10 A = −123.456
20 PAUSE USING "####.##"; A
30 PAUSE A, USING "####" ; A
```

When excuted, this program displays the following:

- PC-1

```
                              −123.45
               −123              −123
```

- PC-8

```
                              −123.45
            −123.45              −123
```

Pseudovariables

	PC-1	PC-2	PC-3	PC-8	Comments
INKEY$		Y	Y	Y	
MEM	Y	Y	Y	Y	
PI or π	Y	Y	Y	Y	**PC-1** has only π
TIME		Y			

Numeric Functions

	PC-1	PC-2	PC-3	PC-8	Comments
ABS	Y	Y	Y	Y	
ACS	Y	Y	Y	Y	
ASN	Y	Y	Y	Y	
ATN	Y	Y	Y	Y	
COS	Y	Y	Y	Y	
DEG	Y	Y	Y	Y	
DMS	Y	Y	Y	Y	
EXP	Y	Y	Y	Y	
INT	Y	Y	Y	Y	
LOG	Y	Y	Y	Y	
LN	Y	Y	Y	Y	
NOT		Y	Y	Y	
POINT		Y			
RND		Y	Y	Y	
SGN	Y	Y	Y	Y	
SIN	Y	Y	Y	Y	
SQR or √	Y	Y	Y	Y	**PC-1** has only √
STATUS		Y			
TAN	Y	Y	Y	Y	

String Functions

	PC-1	PC-2	PC-3	PC-8	Comments
ASC		Y	Y	Y	
CHR$		Y	Y	Y	
LEFT$		Y	Y	Y	
LEN		Y	Y	Y	
MID$		Y	Y	Y	
RIGHT$		Y	Y	Y	
STR$		Y	Y	Y	
VAL		Y	Y	Y	

Operators

	PC-1	PC-2	PC-3	PC-8	Comments
^	Y	Y	Y	Y	
*, /, +, –	Y	Y	Y	Y	
>, >=, =, <>, <=, <	Y	Y	Y	Y	
AND, OR		Y	Y	Y	
&		Y	Y	Y	

Note: Other Tandy pocket computers (PC-4, PC-5, PC-6 and PC-7) use different dialects of BASIC, so programs written for these computers may not run on the **PC-8.**

APPENDIX F
NUMERIC PRECISION

Accuracy in Computations

While the **PC-8** displays the results of calculations to an accuracy of 1Ø digits, 12 digits are used internally in calculations to provide additional accuracy. For example: 5/9 yields 5.55555555555E−Ø1

internally which is rounded to the 1Øth digit and displayed externally as

5.555555556E−Ø1

Similarly,

5/9 * 9 yields 4.99999999999E ØØ

internally and when this is rounded to 1Ø digits externally, the display will show

5.

The function employs an approximation algorithm.
For example:

SIN 3Ø yields 5.ØØØØØØØØØØ1E−Ø1

internally and when this is rounded to 1Ø digits externally, the display will show

Ø.5

This is very significant in the logical expression.
Since the internal value is used in the logical expression,

SIN 3Ø = Ø.5

will be taken as False (Ø).

Therefore, if you use a logical expression in an IF statement, first enter the result into a variable and then compare. The rounding occurs when the value is assigned to a variable.

```
10: INPUT A
20: B = SIN A
30: IF B = Ø.5 THEN. . .
        :
        :
```

Special Limits

In addition to the general limits described above and in Chapter 4, certain functions of the **PC-8** have their own special limits. These are shown in the chart below.

Functions	Dynamic range
$y \wedge x$ (y^x)	$-1 \times 10^{100} < x \log y < 100$ $\left(\begin{array}{l} y=0,\ x \leqq 0\text{: ERROR 2} \\ y=0,\ x > 0\text{: 0} \\ y < 0,\ x \neq \text{integer: ERROR 2} \end{array}\right)$ (Ex.) 0^0 ENTER → ERROR 2 0^5 ENTER → 0. (−4)^0.5 ENTER → ERROR 2 The value of Y can be negative only if X is an integer.
SIN x COS x TAN x	DEG: $\mid x \mid < 1 \times 10^{10}$ RAD: $\mid x \mid < \frac{\pi}{180} \times 10^{10}$ GRAD: $\mid x \mid < \frac{10}{9} \times 10^{10}$ In TAN x, however, the following cases are excluded. DEG: $\mid x \mid = 90\ (2n-1)$ RAD: $\mid x \mid = \frac{\pi}{2}\ (2n-1)$ GRAD: $\mid x \mid = 100\ (2n-1)$ (n: integer)
ASN x (SIN^{-1} x) ACS x (COS^{-1} x)	$-1 \leqq x \leqq 1$
ATN x (TAN^{-1} x)	$\mid x \mid < 1 \times 10^{100}$
LN x LOG x	$1 \times 10^{-99} \leqq x < 1 \times 10^{100}$
EXP x	$-1 \times 10^{100} < x \leqq 230.2585092$
$\sqrt{x}$	$0 \leqq x < 1 \times 10^{100}$

Functions other than those shown above can be calculated only when x stays within the following range.

$$1 \times 10^{-99} \leqq \mid x \mid < 1 \times 10^{100} \text{ and } 0$$

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.

APPENDIX G
SPECIFICATIONS

Model:	**PC-8** Pocket Computer	
Processor:	4 bit CMOS CPU	
Programming Language:	BASIC	
Memory Capacity:	System ROM:	About 17.4 K Bytes
	RAM	
	System	About 500 Bytes
	User	
	Fixed Memory Area (A ~ Z, A$ ~ Z$)	208 Bytes
	Program/Data Area	1278 Bytes
Stack:	Sub-routine: 10 stacks	Function: 16 stacks
	FOR–NEXT: 5 stacks	Data: 8 stacks
Operators:	Addition, subtraction, multiplication, division, exponentiation, trigonometric and inverse trigonometric functions, logarithmic and exponential functions, angle conversion, square root, sign, absolute, integer, relational operators, logical operators.	
Numeric Precision:	10 digits (mantissa) + 2 digits (exponent).	
Editing Features:	Cursor left and right, line up and down, character insert, character delete.	
Memory Protection:	CMOS Battery backup.	
Display:	16 character liquid crystal display with 5 x 7 dot characters.	
Keys:	53 keys: Alphabetic, numeric, special symbols, and functions. Numeric pad. User defined keys.	
Power Supply:	6.0V DC Lithium cells. Type: CR-2032 Cat. No. 23-162	
Power Consumption:	6.0V DC @ 0.07W Approximately 75 hours of continuous operation under normal conditions (based on 10 minutes of operation or program execution and 50 minutes of display time per hour at a temperature of 20°C). The time may vary slightly depending on usage and the type of battery used.	

Operating Temperature:	55°F ~ 86°F (13°C ~ 30°C).
Dimensions:	5-5/16" (W) x 2-3/4" (D) x 3/8" (H) 135 (W) x 70 (D) x 9.5 (H) mm.
Weight:	Approximately 0.21 lbs. (95 g) with batteries
Accessories:	Hard cover, two lithium batteries (built-in), keyboard template, two strips of double-sided tape and operation manual

APPENDIX H
PROGRAM EXAMPLES

Probably you have acquired knowledge on a number of program commands as you have progressed up to this page. It is necessary, however, to generate actual programs by yourself in addition to those given in the instruction manual, so that you can generate programs freely using BASIC language. Like driving a car or playing tennis that can be improved by actual practice, you can improve your programming only by generating as many programs as possible regardless of your skill. It is also important for you to refer to programs generated by others. For your reference, the following pages contain a variety of programs using BASIC commands.

(Radio Shack and/or its subsidiaries assume no responsibilities or obligations for any losses or damages that could arise through the use of the software programs employed in this operation manual.)

CONTENTS

Showing the bytes used in each program

The number of bytes used in each program is shown at the end of each program listing.

The way to find this out is as follows:

RUN mode

1) CLEAR

2) 1278 – MEM [ENTER] → number of bytes.

Program Title: **NEWTON'S METHOD FOR FINDING ROOTS OF EQUATIONS**

OVERVIEW (mathematical)

Finding the roots of equations is usually troublesome, but by using Newton's Method the approximate roots of equations can be found.

When 1 root is found, depending on the interval width, by using Newton's Method the starting point automatically changes.

CONTENTS

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$$

If the absolute value of the distance between X_n and X_{n+1} is less than 10^{-8}, X_n is considered a root and is displayed. Here the first derivative is defined in the following way:

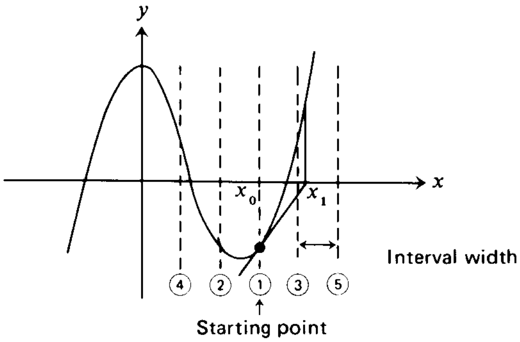
$$f'(X) = \frac{f(X+h)-f(X)}{h} \quad (h \text{ is the minute interval})$$

Change E-8 in line 340 to change the value for 10^{-8}.

INSTRUCTIONS

INPUT

- Starting point
- Minute interval
- Interval



OUTPUTS

Root value (by pressing the ENTER key, the next interval's root is found)

EXAMPLE

$x^3 - 2x^2 - x + 2 = 0$ (the roots are −1, 1, 2)

starting point = 0
minute interval = 10^{-4}
interval = 0.5

The above values are used in the calculation.

The functions are to be written into lines after 500 as subroutines.

How to type in the example:

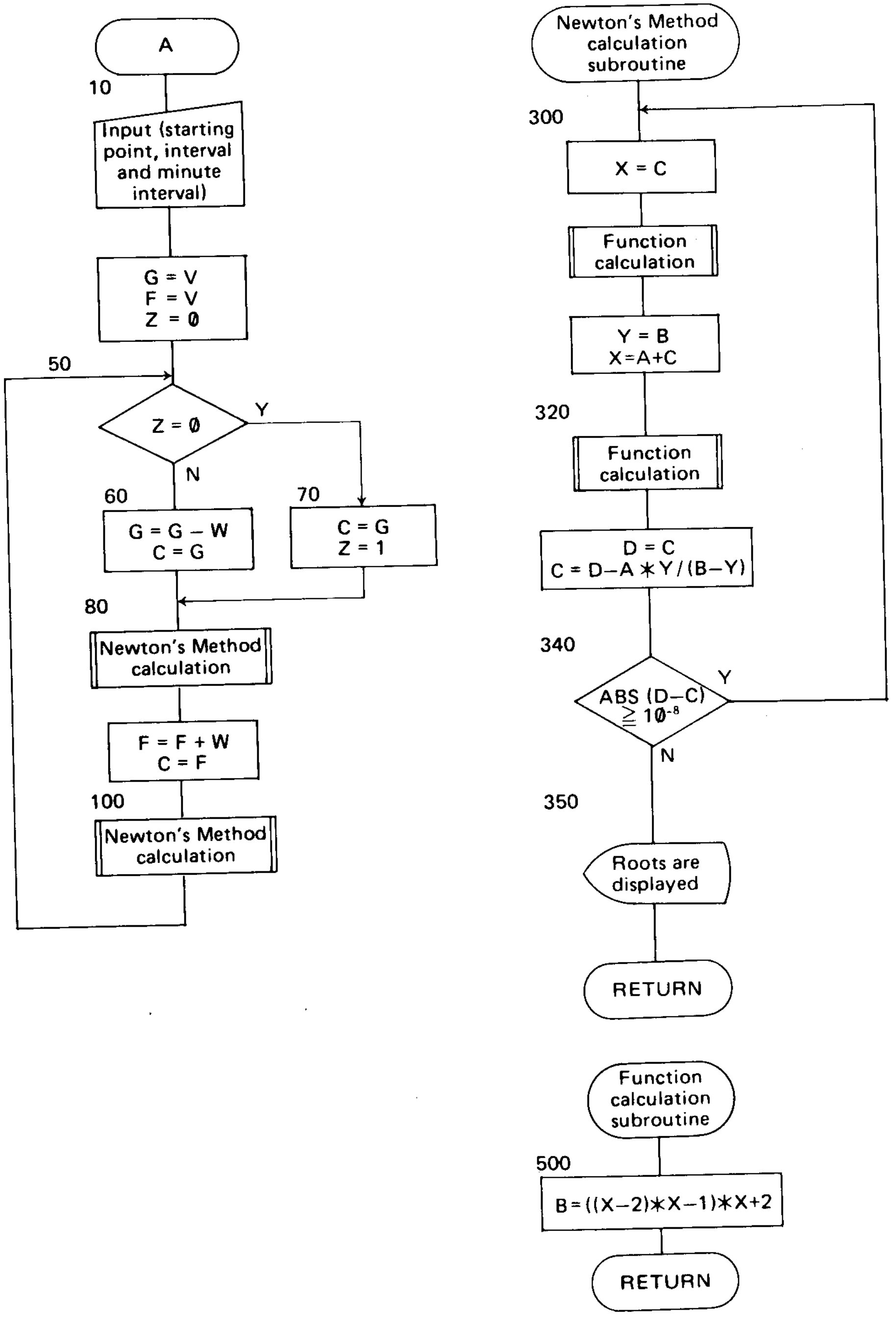
1. Go into PRO mode by operating the mode change key.
2. 500B = ((X−2) ∗ X−1) ∗ X+2 ENTER
   510 RETURN ENTER That is all that had to be done.

Note: This program adopts the basic algorithm of the Newton method. Multipled root may be obtained, but it occurs that one part of the root is not displayed.

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	STARTING POINT = _	Waiting for starting point input
2	0 ENTER	MINUTE INTV. = _	Waiting for minute interval input
3	0.0001 ENTER	INTERVAL = _	Waiting for interval width input
4	0.5 ENTER	2.	Display of roots
5	ENTER	1.	By repeatedly pressing the ENTER key the roots of the function are found.
6	ENTER	−1.	
7	ENTER	1.	
8	ENTER	−1.	
9	ENTER	−1.	
10	ENTER	−1.	
11	ENTER	2.	
	⋮	⋮	
	⋮	⋮	

FLOWCHART


A
10
Input (starting point, interval and minute interval)
G = V
F = V
Z = Ø
50
Z = Ø
Y
N
60
G = G – W
C = G
70
C = G
Z = 1
80
Newton's Method calculation
F = F + W
C = F
100
Newton's Method calculation
Newton's Method calculation subroutine
300
X = C
Function calculation
Y = B
X=A+C
320
Function calculation
D = C
C = D–A ✱Y / (B–Y)
340
ABS (D–C) ≧ 1Ø-8
Y
N
350
Roots are displayed
RETURN
Function calculation subroutine
500
B=((X–2)✱X–1)✱X+2
RETURN

PROGRAM LIST

```
 10:"A": INPUT "STARTING
     POINT=";V
 20:INPUT "MINUTE INTV.=
    ";A
 30:INPUT "INTERVAL=";W
 40:G=V:F=V:Z=0
 50:IF Z=0 GOTO 70
 60:G=G-W:C=G: GOTO 80
 70:C=G:Z=1
 80:GOSUB 300
 90:F=F+W:C=F
100:GOSUB 300
110:GOTO 50
120:END
300:X=C: GOSUB 500
310:Y=B:X=A+C
320:GOSUB 500
330:D=C:C=D-A*Y/(B-Y)
340:IF ABS (D-C)>=E-8
    GOTO 300
350:PRINT C
360:RETURN
500:B=((X-2)*X-1)*X+2
510:RETURN

252
```

MEMORY CONTENTS

A	Minute interval
B	f (x)
C	X_0
D	f (x + h)
E	
F	√
G	√
H	
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	
T	
U	
V	Starting point
W	Interval
X	x
Y	f (x)
Z	Initial flag

Program Title: **AVERAGE, VARIANCE AND STANDARD DEVIATION**

OVERVIEW

If the data are input, the total sum, average, variance, and standard deviation will be calculated for you. Revision of input data as well as data with weights is possible.

CONTENTS

Total sum $\Sigma\, x_i \cdot f_i$

Standard deviation $\sigma = \sqrt{\sigma^2}$

Average $\bar{x} = \dfrac{\Sigma\, x_i \cdot f_i}{\Sigma f_i}$

Variance $\sigma^2 = \dfrac{\Sigma\,(x_i - \bar{x})^2 f_i}{\Sigma f_i - 1}$

Number of data entries (up to 50)

(when there are no weights $f_i = 1$)

INSTRUCTIONS

1. At DEF A , select whether or not there are any weights, then input the data.
2. DEF B is used to find any revision positions in the data. DEF C is used to revise the data.
3. The total sum, average, variance, and standard deviation will be calculated with DEF D.

EXAMPLE

x_i	14.1	14.2	14.3	14.4	14.5
f_i	8	19	23	15	10

(data with weights)

KEY OPERATION SEQUENCE

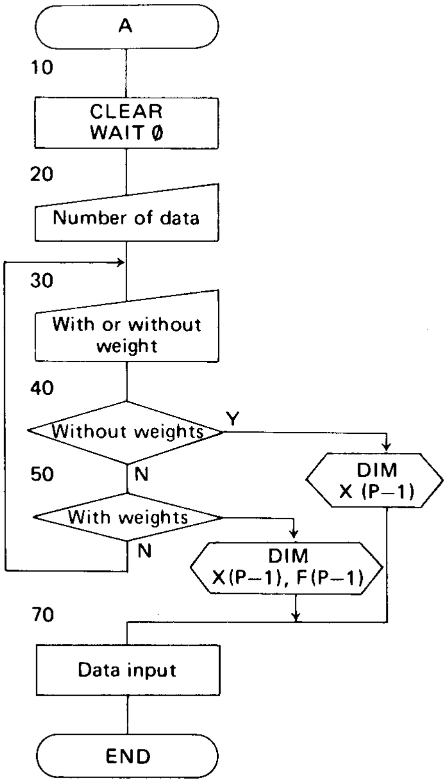
Step No.	Key Input	Display	Remarks
1	DEF A	NO. OF DATA = _	Waiting for number of data input
2	5 ENTER	WEIGHTS	Waiting for the selection of weights/no weights.
		YES = 1/NO = 2? _	
3	1 ENTER	X (1) =	
		?	
4	14.1 ENTER	F (1) =	
		?	
5	8 ENTER	X (2) =	
		?	
	⋮	⋮	
12	14.5 ENTER	F (5) =	
		?	
13	10 ENTER	>	End of the process

KEY OPERATION SEQUENCE

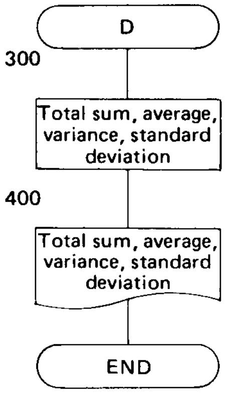
Step No.	Key Input	Display	Remarks
1	DEF B	X (1) = 14.1	
2	ENTER	F (1) = 8	
3	ENTER	X (2) = 14.1	DEF C is used to input the revised values when data errors are found
4	DEF C	X (2) =	
		REVISION VALUE = _	Revised value is input
5	14.2 ENTER	F (2) = 19	
	ENTER		
		⋮	
	⋮	⋮	
1	DEF D	TOTAL SUM	
2	ENTER	1072.5	
3	ENTER	MEAN VALUE	
4	ENTER	14.3	
5	ENTER	VARIANCE	
6	ENTER	1.432432432E–02	
7	ENTER	STD. DEV.	Display of standard deviation
8	ENTER	1.196842683E–01	
9	ENTER	>	Processing finished

FLOWCHART

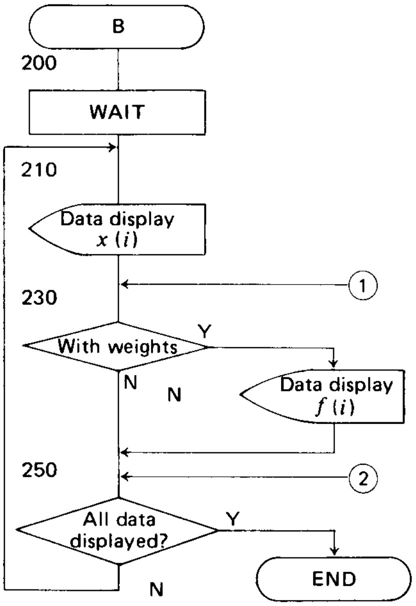
Data input



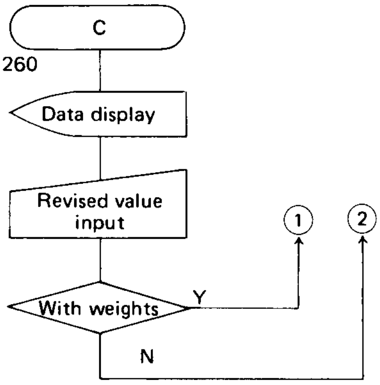
Calculation


D
300
Total sum, average, variance, standard deviation
400
Total sum, average, variance, standard deviation
END

Data revision



Data revision



PROGRAM LIST

```
 10:"A": CLEAR : WAIT 0
 20:INPUT "NO. OF DATA="
    ;P
 30:WAIT 60: PRINT "WEIG
    HTS": INPUT "YES=1/N
    O=2 ? ";A: WAIT
 40:IF A=2 DIM X(P-1):
    GOTO 70
 50:IF A=1 DIM X(P-1),F(
    P-1): GOTO 70
 60:GOTO 30
 70:FOR I=0 TO P-1
 80:B$="X("+ STR$ (I+1)+
    ")="
 85:PAUSE B$: INPUT X(I)
    : GOTO 100
 90:GOTO 85
100:IF A=2 GOTO 150
120:B$="F("+ STR$ (I+1)+
    ")="
130:PAUSE B$: INPUT F(I)
    : GOTO 150
140:GOTO 130
150:NEXT I: END
200:"B": WAIT :I=0
210:B$="X("+ STR$ (I+1)+
    ")=":J=1: PRINT B$;X
    (I)
230:IF A=1 LET B$="F("+
    STR$ (I+1)+")=":
    PRINT B$;F(I):J=2
240:I=I+1
250:IF I=P END
255:GOTO 210
260:"C": PAUSE B$: IF
    LEFT$ (B$,1)="X"
    INPUT "REVISION VALU
    E=";X(I): GOTO 290
270:IF LEFT$ (B$,1)="F"
    INPUT "REVISION VALU
    E=";F(I): GOTO 290
280:GOTO 250
290:IF J=1 GOTO 230
291:GOTO 210
300:"D":N=0:T=0:S=0: FOR
    I=0 TO P-1:X=X(I)
305:F=1: IF A=1 LET F=F(
    I)
310:N=N+F:T=T+F*X:S=S+F*
    X*X: NEXT I
400:WAIT :X=T/N:Q=(S-N*X
    *X)/(N-1):S=√Q:
    PRINT "TOTAL SUM":
    PRINT T: PRINT "MEAN
     VALUE": PRINT X
410:PRINT "VARIANCE":
    PRINT Q: PRINT "STD.
     DEV.": PRINT S: END

649
```

MEMORY CONTENTS

A	√
B$	√
C	
D	
E	
F	√
G	
H	
I	√
J	Flag
K	
L	
M	
N	√
O	
P	Data number
Q	Variance
R	
S	Standard deviation
T	Total sum
U	
V	
W	
X	Average
Y	
Z	
X (P–1)	Data
F (P–1)	Data

Program Title: **INTERSECTION BETWEEN CIRCLES AND STRAIGHT LINES**

OVERVIEW

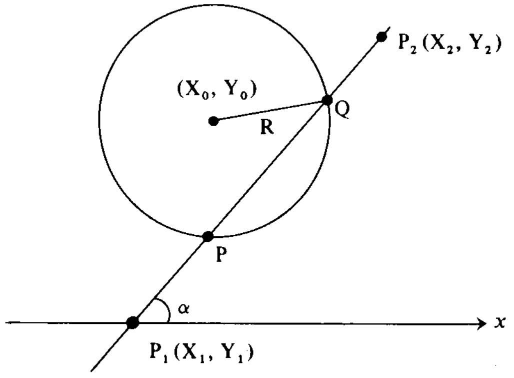
The points of intersection between circles and straight lines in the X–Y plane are found.

CONTENTS

The 2 points of intersection between a circle and a straight line are P and Q.

(Note) The angles are in degrees, minutes, and seconds and are to be input in the following way:

123.1423 = 123 degrees 14 minutes 23 seconds.

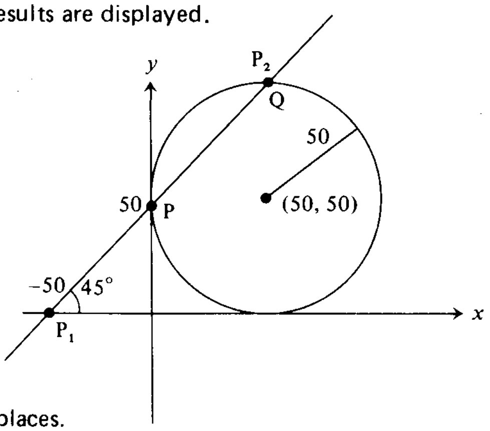


INSTRUCTIONS

1. If the straight line is determined by 2 points, DEF A is used.
   If the line is determined by 1 point and 1 direction angle, DEF B is used.
2. After the data are input, the results are displayed.

EXAMPLE

$X_1 = -50$			
$Y_1 = 0$			
$X_2 = 50$	$X_P = 0$		
$Y_2 = 100$	$Y_P = 50$		
$X_0 = 50$	$X_Q = 50$		
$Y_0 = 50$	$Y_Q = 100$		
$R = 50$			
$\alpha = 45°$			



(Note) The coordinate values are accurate up to 5 decimal places.

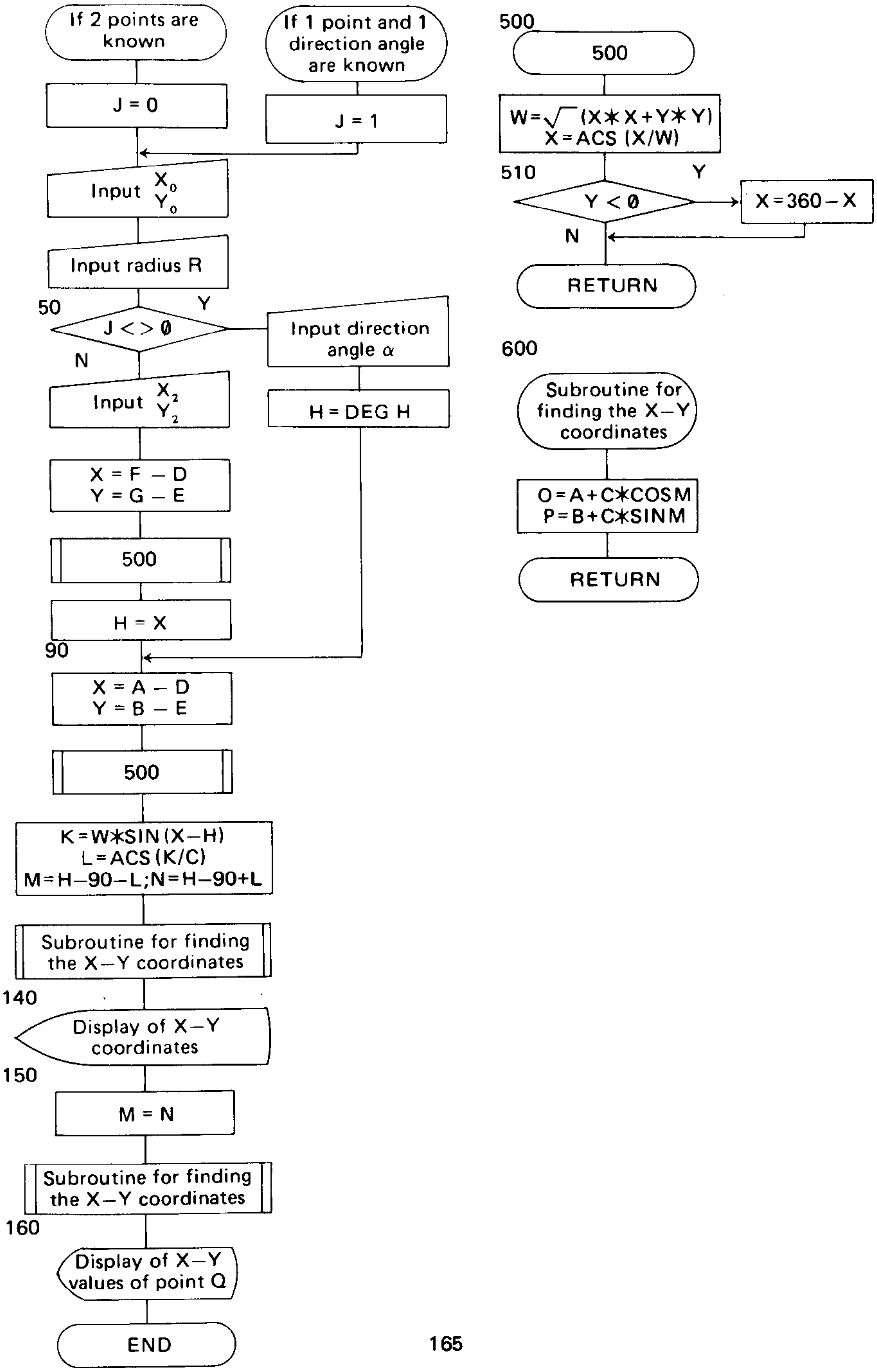
KEY OPERATION SEQUENCE
(when 2 points on the line are known)

Step No.	Key Input	Display	Remarks
1	DEF A	XØ = _	
2	50 ENTER	YØ = _	
3	50 ENTER	R = _	
4	50 ENTER	X1 = _	
5	−50 ENTER	Y1 = _	
6	0 ENTER	X2 = _	
7	50 ENTER	Y2 = _	
8	100 ENTER	P−X 0.0000	(x_P, y_P)
9	ENTER	P−Y 49.9999	
10	ENTER	Q−X 50.0000	(x_Q, y_Q)
11	ENTER	Q−Y 100.0000	
12	ENTER	>	END

(when 1 point on the line and 1 direction angle are known)

Step No.	Key Input	Display	Remarks
1	DEF B	XØ = _	
2	50 ENTER	YØ = _	
3	50 ENTER	R = _	
4	50 ENTER	X1 = _	
5	−50 ENTER	Y1 = _	
6	0 ENTER	A = _	
7	45 ENTER	P−X 0.0000	(x_P, y_P)
8	ENTER	P−Y 49.9999	
9	ENTER	Q−X 50.0000	(x_Q, y_Q)
10	ENTER	Q−Y 100.0000	
11	ENTER	>	END

FLOWCHART


If 2 points are known
J = 0
If 1 point and 1 direction angle are known
J = 1
Input X0 Y0
Input radius R
50
J <> Ø
Y
N
Input direction angle α
H = DEG H
Input X2 Y2
X = F – D
Y = G – E
500
H = X
90
X = A – D
Y = B – E
500
K=W✻SIN (X–H)
L=ACS (K/C)
M=H–90–L;N=H–90+L
Subroutine for finding the X–Y coordinates
140
Display of X–Y coordinates
150
M = N
Subroutine for finding the X–Y coordinates
160
Display of X–Y values of point Q
END
500
500
W=√(X✻X+Y✻Y)
X=ACS (X/W)
510
Y < Ø
Y
X=360–X
N
RETURN
600
Subroutine for finding the X–Y coordinates
O=A+C✻COSM
P=B+C✻SINM
RETURN

PROGRAM LIST

```
 10:"A":J=0: GOTO 30
 20:"B":J=1
 30:DEGREE : INPUT "X0=
    ";A,"Y0= ";B,"R= ";C
 40:INPUT "X1= ";D,"Y1=
    ";E
 50:IF J<>0 INPUT "A= ";
    H:H= DEG H: GOTO 90
 60:INPUT "X2= ";F,"Y2=
    ";G
 70:X=F-D:Y=G-E: GOSUB 5
    00
 80:H=X
 90:X=A-D:Y=B-E: GOSUB 5
    00
100:K=W* SIN (X-H)
110:L= ACS (K/C)
120:M=H-90-L:N=H-90+L
130:GOSUB 600
140:PRINT USING "######.
    ####";"P-X";O: PRINT
    "P-Y";P
150:M=N: GOSUB 600
160:PRINT "Q-X";O: PRINT
    "Q-Y";P
170:END
500:W=√(X*X+Y*Y)
510:X= ACS (X/W): IF Y<0
    LET X=360-X
520:RETURN
600:O=A+C* COS M:P=B+C*
    SIN M: RETURN

351
```

MEMORY CONTENTS

A	X_0
B	Y_0
C	R
D	X_1
E	Y_1
F	X_2
G	Y_2
H	√
I	
J	√
K	*h*
L	α
M	Q_P
N	Q_Q
O	X_P, X_Q
P	Y_P, Y_Q
Q	
R	
S	
T	
U	
V	
W	L
X	ΔX, θ
Y	ΔY
Z	

Program Title: NUMBER OF DAYS CALCULATION

OVERVIEW

How many days has it been since you were born?
This program is convenient for answering such questions. By setting a certain day, this program will output the number of days that have passed since that day.

CONTENTS

[Instructions]

DEF A

BASE YEAR	ENTER
MONTH	ENTER
DAY	ENTER
TARGET YEAR	ENTER
MONTH	ENTER
DAY	ENTER

To end the program, type in DEF Z in place of the year.

[Example]

from 1976 year 10 month 5 day
to 1982 year 6 month 4 day : 2068 days
to 1985 year 1 month 1 day : 3010 days

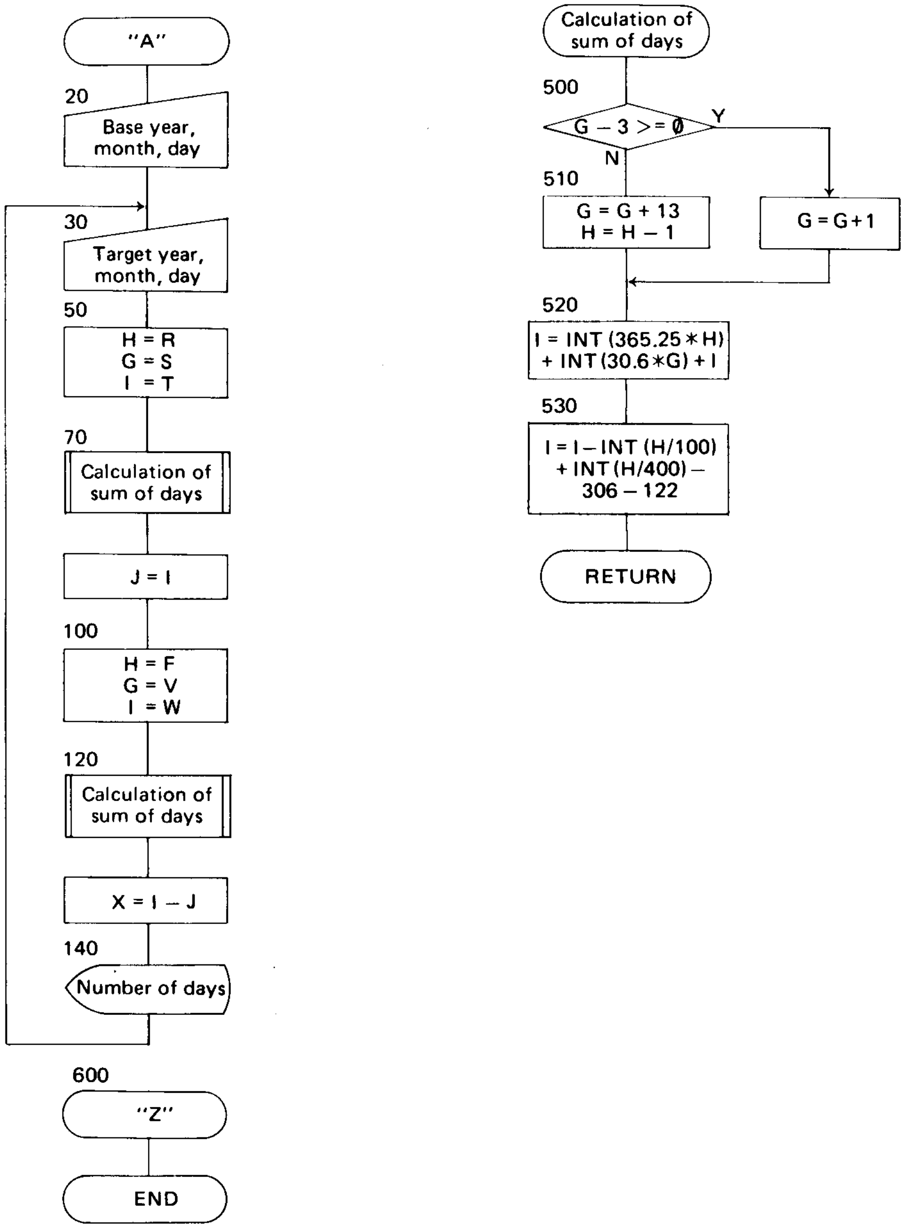
Note: Number of days calculated by this program doesn't include the base day. If you want the number that includes the base day, please change the program as follows:

```
140:WAIT : USING : PRINT
    "DAYS=",X+1
```

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	START YEAR =	
2	1976 ENTER	MONTH =	Base date 1976 year 10 month 5 day input
3	10 ENTER	DAY =	
4	5 ENTER	END YEAR =	
5	1982 ENTER	MONTH =	Target date 1982 year 6 month 4 day input
6	6 ENTER	DAY =	
7	4 ENTER	DAYS = 2068.	
8	ENTER	END YEAR =	
9	1985 ENTER	MONTH =	Target date 1985 year 1 month 1 day input
10	1 ENTER	DAY =	
11	1 ENTER	DAYS = 3010.	
12	ENTER	END YEAR =	
13	DEF Z	>	

FLOWCHART


"A"
20
Base year, month, day
30
Target year, month, day
50
H = R
G = S
I = T
70
Calculation of sum of days
J = I
100
H = F
G = V
I = W
120
Calculation of sum of days
X = I – J
140
Number of days
600
"Z"
END
Calculation of sum of days
500
G – 3 >= Ø
Y
N
510
G = G + 13
H = H – 1
G = G+1
520
I = INT (365.25 * H)
+ INT (30.6 *G) + I
530
I = I – INT (H/100)
+ INT (H/400) –
306 – 122
RETURN

PROGRAM LIST

```
 10:"A"
 20:INPUT "START YEAR=";
    R,"MONTH=";S,"DAY=";
    T
 30:INPUT "END YEAR=";F,
    "MONTH=";V,"DAY=";W
 50:H=R
 60:G=S:I=T
 70:GOSUB 500
 80:J=I
100:H=F
110:G=V:I=W
120:GOSUB 500
130:X=I-J
140:WAIT : USING : PRINT
    "DAYS=",X
150:GOTO 30
500:IF G-3>=0 LET G=G+1:
    GOTO 520
510:G=G+13:H=H-1
520:I= INT (365.25*H)+
    INT (30.6*G)+I
530:I=I- INT (H/100)+
    INT (H/400)-306-122:
    RETURN
600:"Z": END

270
```

MEMORY CONTENTS

A	
B	
C	
D	
E	
F	Year (after calculation)
G	√
H	√
I	√
J	√
K	
L	
M	
N	
O	
P	
Q	
R	Start year
S	Month of base date
T	Day of base date
U	
V	Month of target date
W	Day of target date
X	Number of days
Y	
Z	

Program Title: TYPING PRACTICE

OVERVIEW

Quick key operation!

How fast and accurate is your typing?

If you practice with this program, it will make programming much easier for you. Improve your skill!

CONTENTS (such as calculation contents)

The number of characters (4 ~ 6) is randomly chosen.

The character arrangement (A ~ Z) is done randomly.

The allotted time depends on the number of characters and the grade level.

3 is the shortest time allotment while 1 is the longest.

INSTRUCTIONS

Run the program and 4 to 6 characters will be displayed. You are to type in the same characters within the allotted time.

If they are all correct, you get 10 points.

If more than half are correct, you get 5 points.

After the allotted time is over, the next problem is displayed. The allotted time depends on the grade, which has three levels (1, 2, 3).

3 is the shortest time allotment while 1 is the longest.

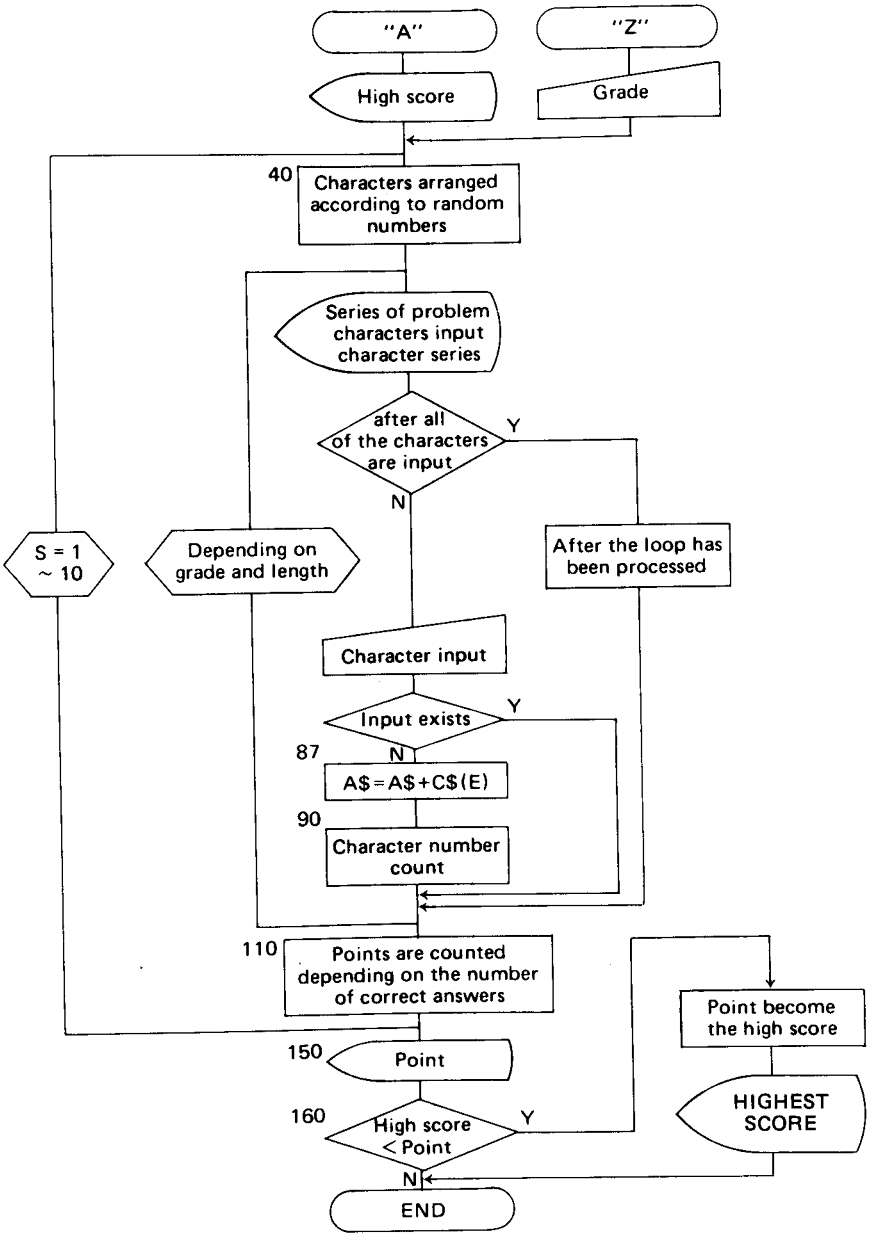
Point competition is done within the same grade category.

There are 10 problems, making the maximum score 100 points.

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF Z	GRADE (1, 2, 3)?	Grade input
2	1 ENTER	A Z B D C	
3	A	A Z B D C A	
4	Z	A Z B D C A Z	
	⋮	⋮	
	⋮	YOUR – SCORE = 80	After the 10 questions are answered the score is displayed
		HIGHEST SCORE	If your score is higher than the high score the guidance is displayed
		>	
1	DEF A	HIGH–SCORE=80	When you want to play in the same grade
		B W V S	
2	B	B W V S B	
	⋮	⋮	
	⋮	YOUR – SCORE = 60	
		>	

FLOWCHART


"A"
"Z"
High score
Grade
40
Characters arranged according to random numbers
Series of problem characters input character series
after all of the characters are input
Y
N
S = 1 ~ 10
Depending on grade and length
After the loop has been processed
Character input
Input exists
Y
87
N
A$=A$+C$(E)
90
Character number count
110
Points are counted depending on the number of correct answers
Point become the high score
150
Point
HIGHEST SCORE
160
High score < Point
Y
N
END

PROGRAM LIST

```
 10:"Z": CLEAR : DIM B$(
    5),C$(5): RANDOM
 15:INPUT "GRADE(1,2,3)?
    ";L: WAIT 0
 17:IF (L=1)+(L=2)+(L=3)
    <>1 THEN 15
 18:GOTO 30
 20:"A": WAIT 0:P=0:
    PAUSE "HIGH-SCORE=";
    X
 30:FOR S=1 TO 10
 40:B= RND 4+2:Y$="":R=
    INT (B/2)
 50:FOR C=0 TO B-1:C$(C)
    =""
 60:D= RND 26:B$(C)=
    CHR$ (D+&40):Y$=Y$+
    CHR$ (D+&40): NEXT C
    :A$=""
 70:E=0: WAIT 30: USING
    "&&&&&&&&"
 80:FOR W=1 TO B*10/L:
    PRINT Y$;A$: IF E=B
    LET W=B*20/L: GOTO 1
    90
 85:C$(E)= INKEY$ : IF C
    $(E)="" THEN 100
 87:A$=A$+C$(E)
 90:E=E+1
100:NEXT W:Q=0
110:FOR W=0 TO B-1: IF B
    $(W)=C$(W) LET Q=Q+1
120:NEXT W: IF Q<=R THEN
    150
130:IF Q=B LET P=P+10:
    GOTO 150
140:P=P+5
150:NEXT S: USING :
    PAUSE "YOUR-SCORE=";
    P
160:IF P>X LET X=P: WAIT
     100: PRINT "HIGHEST
    SCORE"
170:END

459
```

MEMORY CONTENTS

A$	√
B	√
C	Loop counter
D	√
E	√
F	
G	
H	
I	
J	
K	
L	Grade
M	
N	
O	
P	Score
Q	√
R	√
S	Loop counter
T	
U	
V	
W	Loop counter
X	High score
Y$	
Z	
B$ (5)	√
C$ (5)	√

Program Title: SOFTLANDING GAME

OVERVIEW

This game involves landing a rocket, with only a limited amount of fuel, as softly as possible. The rocket is in free fall. The engine is used to slow down the free falling rocket. If ignition takes place too soon or too much fuel is used, then the rocket is thrust back out into space and becomes dust around the planet.
If all the fuel is burned up, the rocket hits the planet and blows up.
The aim is to land the rocket as softly as possible by controlling the engines while watching how much fuel is burned.

CONTENTS

Gravity is set to be 5 m/(unit time)^2.
If 5 units of fuel per a unit time are burnt, then gravity is offset.

Equations

$$H = H_0 + V_0 t + \frac{1}{2} a t^2$$

$$V = V_0 + a t$$

$$V^2 = V_0^2 + 2aH$$

$$H_0 = 500, \quad V_0 = -50, \quad F_0 = 200$$

H :	height	H_0 :	initial height
V :	speed	V_0 :	initial speed
a :	gravitational acceleration	F_0 :	initial fuel
		F :	fuel burned
t :	time		

The initial height, initial fuel level, and the wait time is stored in line 30 as data. By changing these values the above variables can be changed.

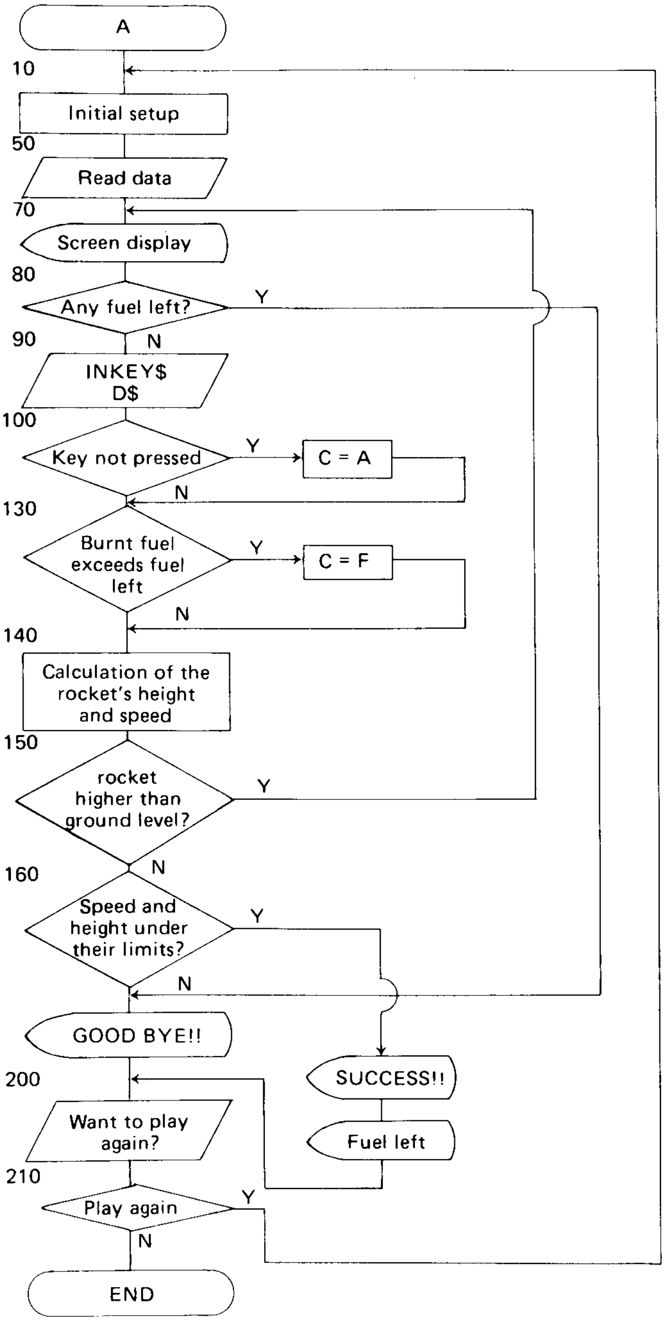
INSTRUCTIONS

1. The program is started by pressing $\boxed{\text{DEF}}$ $\boxed{\text{A}}$. Press $\boxed{0}$ ~ $\boxed{9}$ keys to adjust the amount of fuel used to land the rocket.

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	✱✱✱START✱✱✱	
2	Keys 0 ~ 9 designate fuel burned in unit time	500 −50 2ØØ Ø	Height, speed, fuel left and fuel burnt in unit time are displayed.
	9	452 −46 191 9	
	⋮		
	⋮	Repeat	
	⋮		
	(If successful)	SUCCESS !!	
		FUEL LEFT: F = 15	
	(If failed)	GOOD BYE!!	
		REPLAY (Y/N)?	Wait for input on whether you wish to play again
	Y		Play again
	N	>	End

FLOWCHART


A
10
Initial setup
50
Read data
70
Screen display
80
Any fuel left?
Y
90
N
INKEY$
D$
100
Key not pressed
Y
C = A
N
130
Burnt fuel exceeds fuel left
Y
C = F
N
140
Calculation of the rocket's height and speed
150
rocket higher than ground level?
Y
160
N
Speed and height under their limits?
Y
N
GOOD BYE!!
SUCCESS!!
200
Fuel left
Want to play again?
210
Play again
Y
N
END

PROGRAM LIST

```
 10:"A": WAIT 50: CLEAR
    : USING :S=-50:A=0:D
    $=""
 20:PRINT "  *** START *
    **"
 30:DATA "TIME=",50,"FUE
    L=",200,"HEIGHT=",50
    0
 40:RESTORE
 50:READ B$,W,B$,F,B$,H
 60:WAIT W
 70:PRINT USING "####";H
    ;S;F;C
 80:IF F<=0 GOTO 170
 90:D$= INKEY$
100:IF D$="" LET C=A:
    GOTO 130
110:C= VAL D$
120:A=C
130:IF C>F LET C=F
140:F=F-C:X=C-5:H=H+S+X/
    2:S=S+X
150:IF H>0 GOTO 70
160:IF ( ABS H<5)+( ABS
    S<5)=2 PRINT "SUCCES
    S!!": GOTO 180
170:PRINT "GOOD BYE!!":
    GOTO 190
180:WAIT 150: PRINT
    USING "####";"FUEL L
    EFT:F=";F
190:WAIT 50: PRINT "REPL
    AY (Y/N) ?":Z$=
    INKEY$
200:IF (Z$="Y")+(Z$="N")
    <>1 GOTO 190
210:IF Z$="Y" GOTO 10
220:END

401
```

MEMORY CONTENTS

A	√
B$	√
C	Fuel burned
D$	Fuel burned
E	
F	Initial fuel level, fuel left
G	
H	Initial height, height
I	
J	
K	
L	
M	
N	
O	
P	
Q	
R	
S	Speed
T	
U	
V	
W	Wait time
X	√
Y	
Z$	√

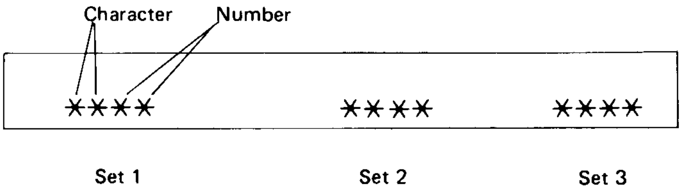
Program Title: MEMORY CHECKER

OVERVIEW

A line of 12 characters will be displayed on the screen for approx. 5 seconds. Your memory will be tested by how well you input the above line after it has disappeared.

CONTENTS

The following type of line will be displayed for approx. 5 seconds. There are 2 characters and 2 numbers in each set.



The 3 sets shown above are to be memorized and then input as answers.

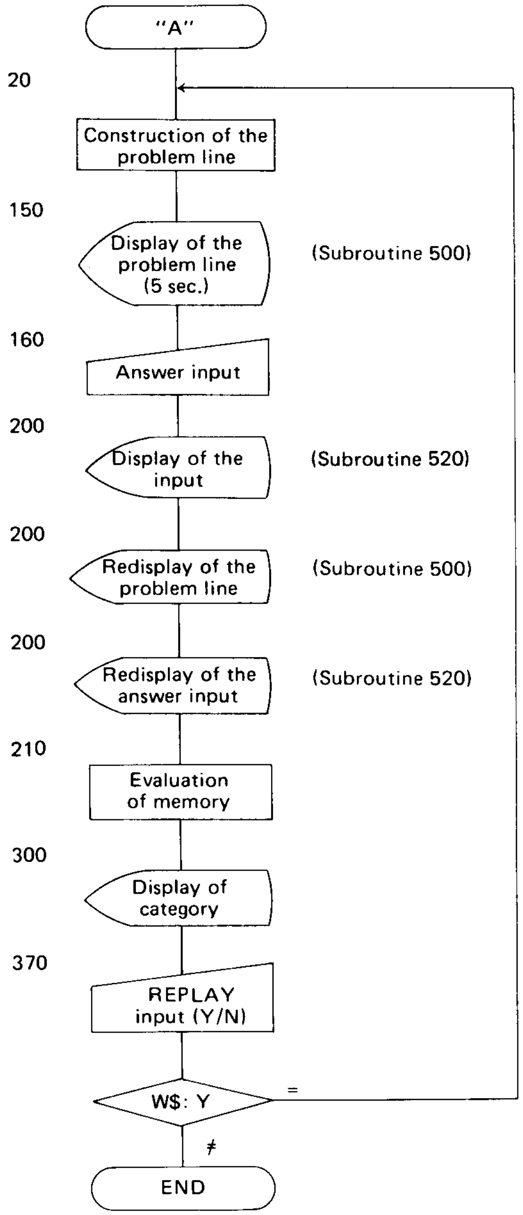
The computer will then analyze your answers and place you in one of the possible 7 categories.
Each set is split into 2 parts of former 2 and latter 2 characters, giving a total of 6 points when all the answers are correct.

Points	Evaluation Message
0	IDIOT
1	BAD
2	AVERAGE
3	OK
4	GOOD !
5	* INTELLIGENT *
6	**GENIUS**

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	MEMORY CHECK	Title
2		**XX **XX **XX	Display of problem line (5 sec.) * . . . character X . . . number
3		ANS. = _	Waiting for the input of set 1
4	(Example) AB12 ENTER	ANS. = _	Waiting for the input of set 2
5	**XX ENTER	ANS. = _	Waiting for the input of set 3
6	**XX ENTER	**XX **XX **XX	Display of the input.
7		**XX **XX **XX	Display of the problem line (1.5 sec.)
8		**XX **XX **XX	Redisplay of the input
9		IDIOT	display of category
		BAD	
		AVERAGE	
		OK	
		GOOD!	
		INTELLIGENT	
		GENIUS	
10		REPLAY (Y/N) ?	Player input request
11	Y or N ENTER		If Y, go to step 2
		>	If N, END

FLOWCHART


"A"
20
Construction of the problem line
150
Display of the problem line (5 sec.)
(Subroutine 500)
160
Answer input
200
Display of the input
(Subroutine 520)
200
Redisplay of the problem line
(Subroutine 500)
200
Redisplay of the answer input
(Subroutine 520)
210
Evaluation of memory
300
Display of category
370
REPLAY input (Y/N)
W$: Y
=
≠
END

Appendix H

PROGRAM LIST

```
 10:"A": USING : WAIT 20
    0: PRINT "MEMORY CHE
    CK": CLEAR : RANDOM
 20:DIM G$(6)*1,N$(10)*1
    ,V$(3)*2,X$(3)*4,Z$(
    3)*2,Y$(3)*4
 30:FOR I=1 TO 9:N$(I)=
    STR$ I: NEXT I:N$(10
    )="0"
 50:FOR I=1 TO 6
 60:J= RND 26:J=J+64
 70:G$(I)= CHR$ (J):
    NEXT I
 80:FOR I=1 TO 3
 90:Y$(I)=" "
100:FOR J=1 TO 2:K= RND
    9
110:Y$(I)=Y$(I)+N$(K):
    NEXT J
120:J=(I-1)*2+1
130:A$(I)=G$(J)+G$(J+1)
140:H$=Y$(I):A$(I+3)=
    RIGHT$ (H$,2): NEXT
    I
150:GOSUB 500
160:FOR I=1 TO 3
170:INPUT "  ANS. = ";X$
    (I):X$(I)= LEFT$ (X$
    (I),4)
180:Z$(I)= LEFT$ (X$(I),
    2)
190:V$(I)= RIGHT$ (X$(I)
    ,2): NEXT I
200:GOSUB 520: GOSUB 500
    : GOSUB 520
210:N=0
220:FOR I=1 TO 3
230:IF A$(I)=Z$(I) LET N
    =N+1
240:IF A$(I+3)=V$(I) LET
    N=N+1
250:NEXT I
260:N=N+1
```

```
270:WAIT 150: ON N GOTO
    300,310,320,330,340,
    350,360
300:PRINT "  IDIOT":
    GOTO 370
310:PRINT "  BAD": GOTO
    370
320:PRINT "  AVERAGE":
    GOTO 370
330:PRINT "   OK": GOTO
    370
340:PRINT "   GOOD!":
    GOTO 370
350:PRINT "* INTELLIGENT
     *": GOTO 370
360:PRINT "**GENIUS**"
370:W$="": INPUT "REPLAY
     (Y/N)?";W$
380:IF W$="N" THEN 600
390:IF W$="Y" THEN 50
395:GOTO 370
400:GOTO 370
500:WAIT 300: PRINT A$(1
    );A$(4);"  ";A$(2);A
    $(5);"  ";A$(3);A$(6
    )
510:RETURN
520:WAIT 80: PRINT USING
    "&&&&&&";X$(1);X$(2)
    ;X$(3)
525:USING
530:RETURN
600:END

808
```

MEMORY CONTENTS

A$	2 columns of characters
B$	
C$	
D$	
E$	
F$	
G	
H$	√
I	Index
J	Random number generation
K	Random number generation
L	Random number generation
M	
N	Counter
O	
P	
Q	
R	
S	
T	
U	
V	
W$	input for REPLAY
X	
Y	
Z	
G$(6)∗1	Characters (1 ~ 6)
N$(10)∗1	Number table (1 ~ 10)
V$(3)∗2	2 columns after answering (1 ~ 3)
X$(3)∗4	Work (1 ~ 3)
Y$(3)∗4	Work (1 ~ 3)
Z$(3)∗2	2 columns before answering (1 ~ 3)

Program Title: DOUBLE ROTATION

OVERVIEW

Quickly put in order A, B, C · · · · ·

This is a game that arranges randomly placed characters (A – J) in alphabetical order. When the letters are arranged in the right order, a score is displayed. The trick is to attack from the best place.

The sooner the characters are arranged, the better.

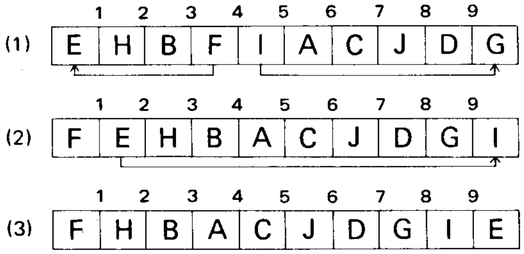
It is fun to race with 2 or 3 of your friends.

INSTRUCTIONS

1. After the program is initiated, by pressing DEF A , "DOUBLE ROTATION" is displayed. A random sequence of characters (A – J) is then displayed.
2. The space in between the characters is taken as the break points (1 – 9) where the numbers are placed. Inputing a break number causes the characters on each side of the breakpoint to be rotated by moving them to the far ends of the row.
3. After the characters have been placed in order, the number of moves required is displayed as the score. The lower the score the better.

EXAMPLE

In (1) 4 is input, "F" and "I" move to each side changing the configuration to (2). If 1 is now input, the "E" moves to the far right but "F" stays in its place because it is already in the far left position, becoming configuration (3).



(1)

E	H	B	F	I	A	C	J	D	G

(2)

F	E	H	B	A	C	J	D	G	I

(3)

F	H	B	A	C	J	D	G	I	E

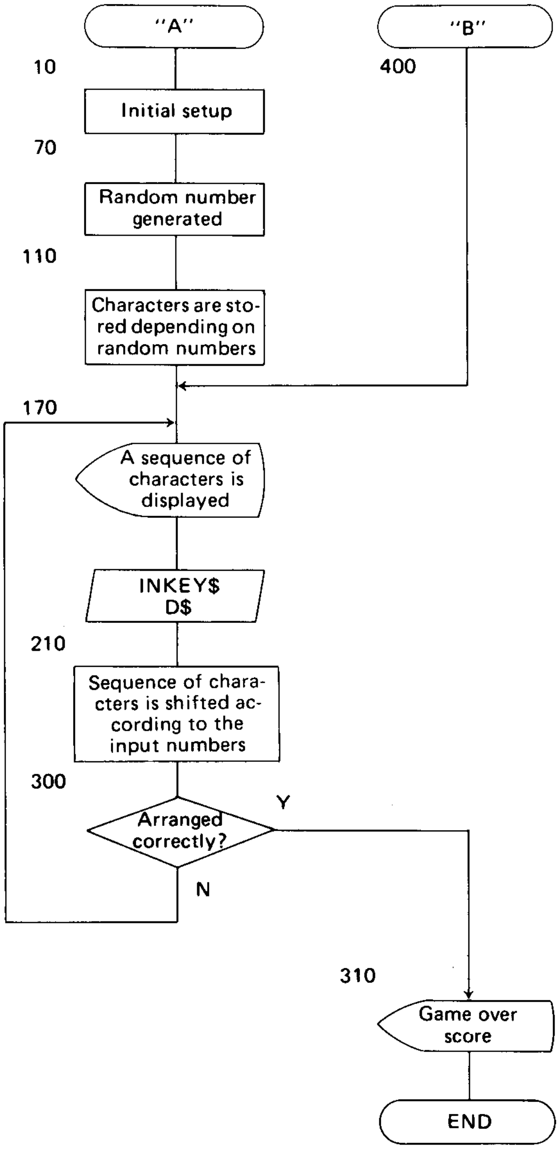
KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	DOUBLE ROTATION	
		A ~ J	Random requence display
2	1 ~ 9	⋮	Numbers between 1 and 9 are selected and input
		Repeated input	
		⋮	
		ABCDEFGHIJ	
		GAME END	
		YOUR SCORE 35	
		>	
			Does player want to play using the same beginning random characters?
1	DEF B	A ~ J	
		Same as DEF A in succession	

If the 2 lines are changed as below, then the game is easier to play and Ø key can be used.

```
180: C = VAL D$: IF D$ = "Ø" GOTO 21Ø
240: IF C <= 1 GOTO 260
```

FLOWCHART


"A"
"B"
10
400
Initial setup
70
Random number generated
110
Characters are stored depending on random numbers
170
A sequence of characters is displayed
INKEY$
D$
210
Sequence of characters is shifted according to the input numbers
300
Arranged correctly?
Y
N
310
Game over score
END

PROGRAM LIST

```
 10:"A": CLEAR : WAIT 50
    : RANDOM : DIM B$(4)
 20:PAUSE "DOUBLE ROTATI
    ON"
 30:B$(0)="ABCDEFGHIJ"
 40:B$(1)=""
 50:A=0
 60:FOR I=1 TO 10
 70:R= RND 10
 80:S=2^(R-1)
 85:B=S AND A
 90:IF B<>0 GOTO 70
100:A=A OR S
110:B$(1)=B$(1)+ MID$ (B
    $(0),R,1): NEXT I
120:B$(2)=B$(1)
130:N=0
170:D$="": PRINT B$(2):D
    $= INKEY$
180:C= VAL D$
190:IF C=0 GOTO 170
210:B$(3)= LEFT$ (B$(2),
    C)
220:B$(4)= RIGHT$ (B$(2)
    ,10-C)
240:IF C=1 GOTO 260
250:B$(3)= RIGHT$ (B$(3)
    ,1)+ LEFT$ (B$(3),C-
    1)
260:IF C=9 GOTO 280
270:B$(4)= RIGHT$ (B$(4)
    ,9-C)+ LEFT$ (B$(4),
    1)
280:B$(2)=B$(3)+B$(4)
290:N=N+1
300:IF B$(2)<>B$(0) GOTO
    150
310:PAUSE "GAME END"
320:WAIT 200: PRINT
    USING "####";"YOUR S
    CORE";N
330:END
400:"B": WAIT 50: GOTO 1
    20
```

```
463
```

MEMORY CONTENTS

A	√
B	√
C	√
D$	Input key
E	
F	
G	
H	
I	√
J	
K	
L	
M	
N	Score
O	
P	
Q	
R	Random numbers
S	√
T	
U	
V	
W	
X	
Y	
Z	
B$ (4)	Character sequences

INDEX

SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

6/86

RADIO SHACK
A Division of Tandy Corporation
Fort Worth, Texas 76102

Printed in Japan