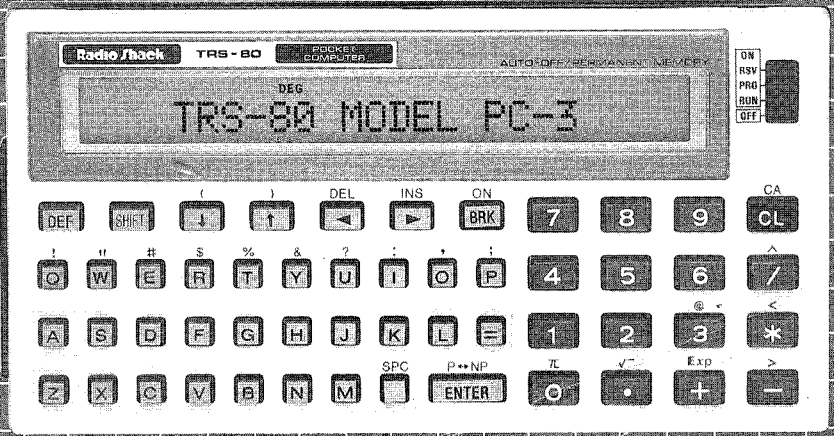
TRS-80®
PC-3
Owner's Manual



CAT. NO. 26-3590

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. Except as provided herein, **RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".

NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

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.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.

2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.

3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

CONTENTS

INTRODUCTORY NOTE

Welcome to the world of **Radio Shack** owners!

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 huge 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 **Radio Shack PC-3** 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 SAFEGUARD – the **PC-3** remembers stored programs and variables even when you turn it off.
* Battery-powered operation for true portability.
* AUTO POWER OFF function which conserves the batteries by turning the power off if no activity takes place within a specified time limit.
* Programmable functions which allow the **PC-3** 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 (Model **PC-3**) for long-term storage and hard-copy printout of programs and data.

Congratulations on entering an exciting and enjoyable new world. The **Radio Shack PC-3** is a powerful tool, designed to meet your specific mathematical, scientific, engineering, business and personal computing needs. With the **Radio Shack PC-3** 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-3** and to serve as a valuable reference tool. Whether you are a "first time user" or an "old hand" with computers, you should acquaint yourself with the **PC-3** by reading and working through Chapters 2 through 6.

* Chapter 2 describes the physical features of the **PC-3.**
* Chapter 3 demonstrates the use of the **PC-3** 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-3.**
* Chapter 5 introduces you to BASIC programming on the **PC-3**, 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 Chapters 8 and 9 to learn the specific features of BASIC as implemented on the **PC-3**. 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.

Chapter 9 provides examples of useful and interesting programs that illustrate some of the techniques of using BASIC on the **PC-3.**

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 **PC-3** Printer/Cassette Interface.
* Chapter 10 – A troubleshooting guide to help you solve some operating and programming problems.
* Chapter 11 – The care and maintenance of your new computer.

Detailed Appendices, at the end of the manual, provide you with useful charts, comparisions, and special discussions concerning the use and operation of the **PC-3.**

CHAPTER 2 INTRODUCTION TO THE PC-3

The **Radio Shack PC-3** Pocket Computer system consists of:

* 52-character keyboard.
* 24-character display.
* Powerful BASIC in 24K ROM.
* 8-bit CMOS processor.
* 2.2KB RAM.
* Optional **PC-3** Printer/Cassette Interface.

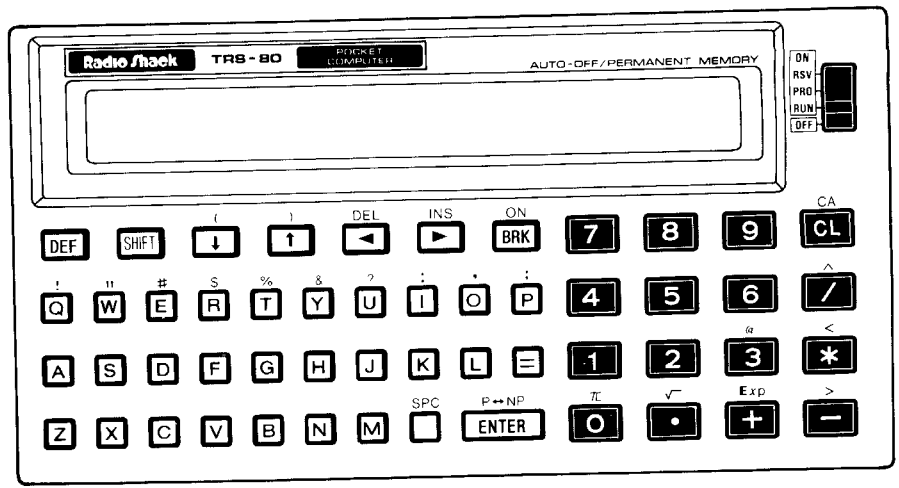


Figure 1. PC-3 (Front View)

To familiarize you with the placement and functions of parts of the **PC-3** keyboard, we will now study each section of the keyboard. For now, just locate the keys and read the description of each. In Chapter 3, we will begin using your new machine.

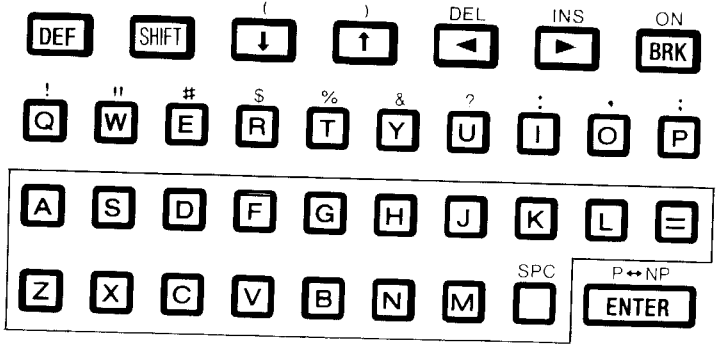


Figure 2.

(A) ~ (Z) Alphabet keys. You are probably familiar with these keys from the standard typewriter keyboard. On the **PC-3** display, the characters always appear in upper case.

(=) Equals key. On the **PC-3** this key is **not** used to indicate the end of a calculation; in BASIC programming, this symbol has a special function.

SPC
( ) SPaCe key. Pressing this key advances the cursor, leaving a blank space. Pressing (SPC) while the cursor is positioned over a character erases that character.

P⟷NP
(ENTER) (ENTER) key. When you press this key, whatever you previously typed is "entered" into the computer's memory. This

key is similar to the Carriage Return key on a typewriter. You must press (ENTER) before the **PC-3** will act on alpha-numeric input from the keyboard. Pressing (SHIFT) before pressing this key will cause the **PC-3** Pocket Computer to switch on and off the printing of calculations on the **PC-3** printer.

(DEF) (DEF) key. This is a special key used to execute BASIC programs.

(SHIFT) (SHIFT) key. Press this key before pressing any key which has a character above it and the character above is displayed. (Note: Not used to capitalize letters as all alphabet keys on the **PC-3** are in upper case).

((↓) Down Arrow key. Press this key to display the next program line. Pressing (SHIFT) before pressing this key produces a left parenthesis.

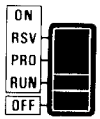
) (↑) Up Arrow key. Press this key to display the previous program line. Pressing (SHIFT) before pressing this key produces a right parenthesis.

DEL (◄) Backspace key. This key allows you to move the cursor to the left without erasing previously typed characters. Pressing (SHIFT) before pressing this key will DELete whatever character the cursor is "on top of".

INS (►) Forward key. This key allows you to move the cursor to the right without erasing previously typed characters. Pressing (SHIFT) before pressing this key makes a space directly before the character the cursor is "on top of". You can then INSert new characters into this space.

ON (BRK) BREAK key. The (BRK) key temporarily interrupts a program which is being executed. Pressing this key after an AUTO OFF turns the computer back on.

! " # $ % & ? : . ; These symbols are found above the top row of alphabet keys. Pressing (SHIFT) and then the alphabet key under the character desired displays these symbols.



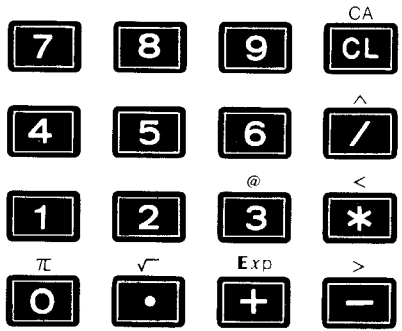
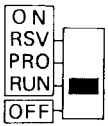


Figure 3.



Use this power slide switch to turn the **PC-3** ON and OFF. Notice that the machine is ON when this switch is positioned in any one of three modes: RUN, PROgram, and ReSerVe.

Ø ~ 9 • Number keys. The layout of these keys is similar to that found on the standard calculator.

CA
CL Clear key. Pressing Clear erases the characters you have just typed in and "releases" errors. Pressing SHIFT before pressing this key activates the CA (reset) function. CA clears the display and resets the computer.

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

(*) Multiplication key. Press this key to include the multiplication operator in calculations. Pressing (SHIFT) and then this key displays the "less than" character.

(−) Subtraction key. Press this key to include the subtraction operator in calculations. Pressing (SHIFT) and then this key displays the "greater than" character.

Exp (+) Addition key. Press this key to include the addition operator in calculations. Pressing (SHIFT) and then this key displays the exponentiation character used in scientific notation.

π √ @ These three characters are found above the zero, decimal point and 3 keys. They are displayed by pressing (SHIFT) and then the character under the symbol desired.

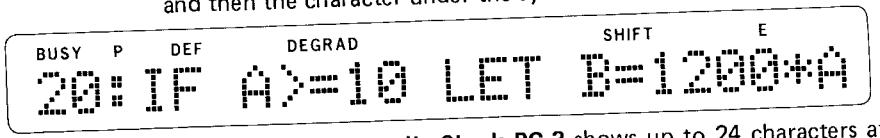


Figure 4. Sample PC-3 Display

The liquid crystal display of the **Radio Shack PC-3** shows up to 24 characters at one time. Although you may input up 80 characters (including (ENTER)) in one line, only the first 24 characters are displayed. To review the remaining characters in a line, move the cursor to the far right and the display will 'scroll' – that is, as characters drop off the left, new characters appear on the right.

The display consists of:

: The prompt. This symbol appears when the computer is awaiting input. As you press, 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. The cursor is also used to position the computer over certain characters when using the INSert and DELete functions.

BUSY Program Execution Indicator. When the **PC-3** is executing a program, this indicator is lit (except when characters are displayed). The **PC-3** will not undergo AUTO OFF while the BUSY indicator is on. BUSY disappears from the display when execution is completed.

P Printer Indicator. This indicator appears whenever you elect the print option when using the **PC-3** as a calculator.

DEF Definable Mode Indicator. This symbol lights up whenever you press the DEF key.

DEGRAD (DEG / RAD / GRAD) Angular Measurement Indicator. This indicator displays the current unit of angles for the input of trigonometric functions. Depending on the mode in use, the display will read DEG (degrees), RAD (radians), or GRAD (gradients).

SHIFT Shift Key Indicator. This indicator lights up when the (SHIFT) key has been depressed. Remember, the (SHIFT) key must be released before depressing any other key.

E Error Indicator. Whenever an error is encountered, this indicator is displayed.

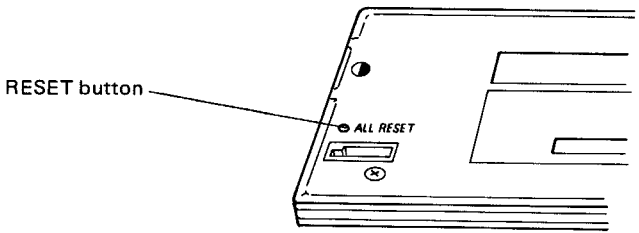
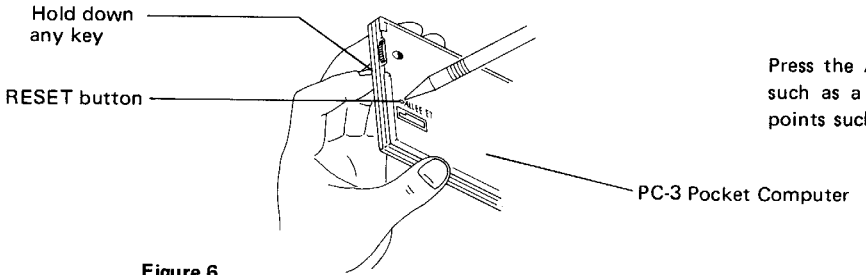


Figure 5. PC-2 (Rear Panel)

ALL RESET: Reset button. This button is used to reset the computer when CLear or CA is not sufficient to correct the problem.

NOTE

To reset the **PC-3**, hold down any key on the keyboard and simultaneously press the RESET button on the back. This preserves all programs, variables, and reserve memory.



Press the ALL 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 6.

If you get no response from any key, even when the above operation is performed, push the RESET button without any key. With this operation, the program, data and all reserved contents are cleared, so do not press the RESET button without any key unless the above trouble occurs.

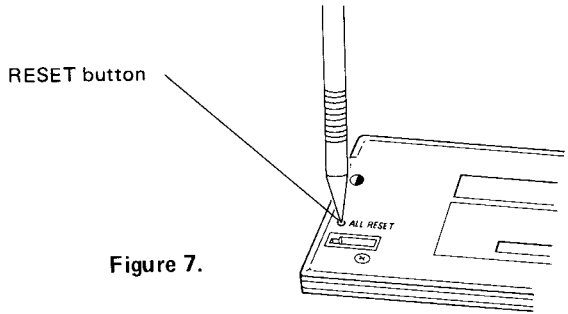


Figure 7.

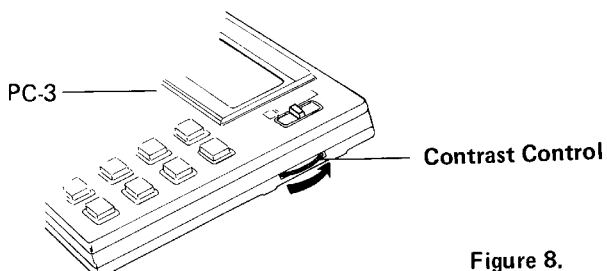


Figure 8.

Turn the control in the direction of the arrow for a brighter display, and turn it in the opposite direction for a dimmer display.
Adjust it so that the display is easy to see.

BATTERY REPLACEMENT FOR THE PC-3

The PC-3 Pocket Computer operates on the lithium battery alone. When connected to the PC-3 Printer/Cassette Interface, the PC-3 can also be supplied from the PC-3 Printer/Cassette Interface if it has enough power voltage and the lithium battery power decrease. This minimizes the power consumption of the lithium battery.

When replacing the batteries, these precautionary instructions will eliminate many problems:

- Always replace both of the batteries at the same time.
- Do not mix a new battery with a used battery.
- Use only: Lithium battery (type CR-2032) x 2

INSTALLING THE BATTERIES

The display is dim and difficult to see when viewed from the front, even after turning the contrast control on the right of the computer counterclockwise as far as it goes. This indicates that the battery power is depleted. In this case, replace the battery promptly. (Using the optional PC-3 Printer/Cassette Interface peripheral equipment, record programs and data on tape in advance.)

(1) Turn off the computer by setting the power slide switch to the OFF position.

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

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

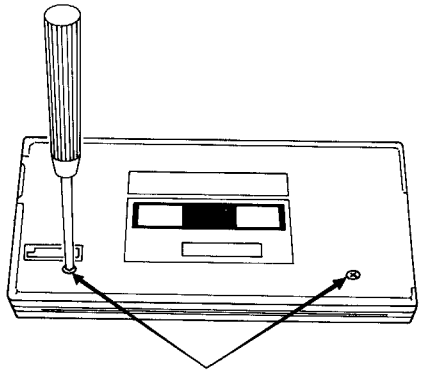


Figure 9

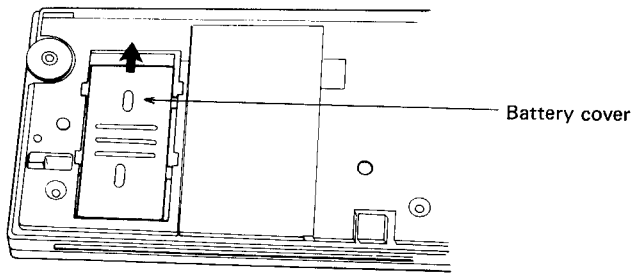


Figure 10

(4) Replace the two batteries observing the correct polarity. (Fig. 11)

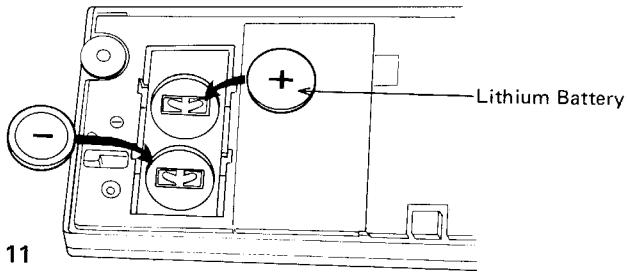


Figure 11

(5) Replace the battery cover by sliding it in the reverse direction of the arrow shown in figure 2.

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

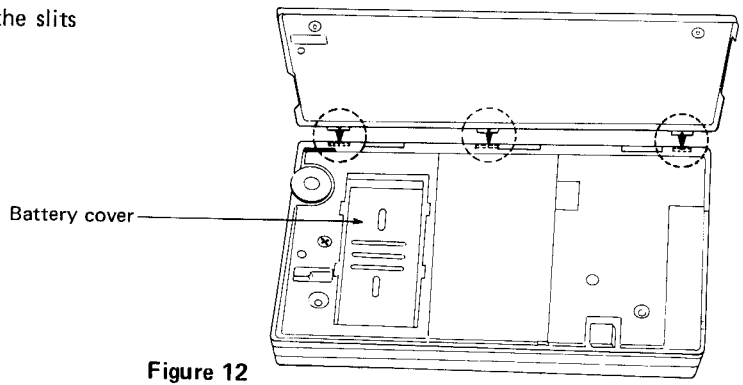


Figure 12

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

(8) Turn on the computer by setting the power slide switch to the ON position and press the RESET button to clear the computer. Then check the following display.

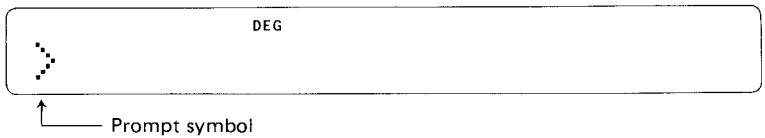


Figure 13. Sample Display On Start-Up

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

NOTE:
Keeping a dead battery may result in damage to the computer due to solvent leakage of the battery. Remove a dead battery promptly.

CAUTION: Keep battery out of reach of children.

CHAPTER 3 USING THE PC-3 AS A CALCULATOR

Now that you are familiar with the layout and components of the **Radio Shack PC-3**, we will begin investigating the exciting capabilities of your new computer.

Because the **PC-3** allows you the full range of calculating functions, plus the increased power of BASIC programming abilities (useful in more complex calculation), it is commonly referred to as a "smart" calculator. That, of course, makes you a "smart" user!

(Before using the **PC-3**, be sure that the batteries are correctly installed.)

Start Up

To turn ON the **PC-3**, slide the power switch up and select one of three modes: RUN, PRO, or RSV. For use as a calculator, the **PC-3** 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-3**, slide the power switch to the OFF position.

When you turn OFF the machine, you clear (erase) the display. However, the **PC-3** does remember all programs, reserve keys, and mode settings which were in use when the computer was turned OFF. All of these settings are still in effect when the machine is turned back ON.
When the BEEP instruction or CLOAD command is executed, stop the execution by pressing the (BRK) key and slide the power switch to the OFF position.

Auto Off

In order to prevent needless battery wear, the **PC-3** automatically powers down when no keys have been pressed for about 11 minutes. (Note: The **PC-3** will not AUTO 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 while entering data. 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 ALL RESET button (See Chapter 2).
The PROMPT (>) tells you that the **PC-3** 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.

(ENTER) informs the **PC-3** that you are 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 ACTED UPON BY THE COMPUTER.

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 access square root, for example), press (SHIFT) , release the (SHIFT) key, 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-3**. These characters have special meanings in the BASIC programming language.

In this manual we use the 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 each keystroke necessary to type in the example calculations. When (SHIFT) is used, we will represent the desired character in the following keystroke. For example, pressing (SHIFT) and (↓) will produce the (character. These keystrokes are written (SHIFT) (().

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

Simple Calculations

The **PC-3** performs calculations with ten-digit precision. If you have not already done so, turn ON your computer by setting it in 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	100.
2 * SHIFT π ENTER	6.283185307
SHIFT √ 6 4 ENTER	8.

Recalling Entries

Even after the **PC-3** has displayed the results of your calculaiton, you can redisplay your last entry. To recall, use the left ◄ and right ► arrows.

The left arrow ◄ recalls the last entry with the cursor positioned over the **last** character.

The right arrow ► recalls the entry 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.

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 300 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

(3) (Ø) (Ø) (/) (6) (ENTER)

Display

```
                                50.
```

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

Input

(◄)

Display

```
300/6_
```

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 the last entry by using the (◄) , the cursor is positioned at the end of the display. Use the (◄) to move the cursor one space to the left.

Input

(◄)

Display

```
300/6
```

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 — one 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)	300/5_
(ENTER)	60.

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 you results do not seem to 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
(►)	30/5

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

Input	Display
►	3Ø/5

Use the INSert key no 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 results seem 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. Use the (►) to move the cursor to the first zero (or any zero). When deleting a character, position the cursor "on top of" the character to be deleted.

Input	Display
(►)	3ØØ/5

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

Input	Display
(SHIFT) (DEL)	3Ø/5

Pressing DELete causes all the characters to shift one space to the left. It deletes the character that the cursor 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 occupies.)

Errors

Recalling your last entry is essential when you get the dreaded ERROR message. Let us image that, unintentionally, you typed this entry into the **PC-3:**

Input	Display
(3) (Ø) (Ø) (/) (/) (5) (ENTER)	ERROR 1

Naturally you are supprised 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 ► .

Input	Display
◄	300/*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	60.

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-3** 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.

Serial Calculations

The **PC-3** allows you to use the results of one calculation as part of the following calculation.

You are planning a special conference and are expecting 300 people to attend. Part of your responsibility in planning this conference is to draw up a detailed budget for approval. You know that your total budget is $150.00 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 serial 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 serial calculations, the entry must begin with an operator. As always, you end the entry with (ENTER) .

NOTE: The (%) key can not be used in the calculation. The (%) key should be used as a character only.
Example: 45ØØØ (*) 15 (SHIFT) (%) → ERROR 1

Input	Display
(ENTER)	675Ø.

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

Input	Display
(−) (4) (Ø) (Ø) (Ø)	6750. −4ØØØ_
(ENTER)	2750.

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 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+6750/45000

Compound calculations, however, must be entered very carefully:

675+6750/45000 might be interpreted as

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

When performing compound calculations, the **PC-3** 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 entries:

(675+6750) / 45000 or 675 + (6750 / 45000)

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 Variable in Calculations

The **PC-3** can store up to 26 simple numeric **variables** under the alphabetic characters A to Z. If you are unfamiliar with the concept of variables, they are more fully explained in Chapter 4. You designate variables with an Assignement 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

(R) (=) (4) (5) (Ø) (Ø) (Ø) (–) (7) (4)
(2) (5)

Display

```
R=45ØØ–7425_
```

(ENTER)	37575.

As you press (ENTER) the **PC-3** performs the calculation and displays the new value of R. You can display the current value of any variable by entering the alphabetic character it is stored under:

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 60% of the remaining money to room rental:

Input	Display
(R) (*) (•) (6) (Ø)	R*.60_
(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 or undergoes an AUTO OFF. Variables are lost only when:

* You assign a new value to the same variable.
* You type in CLEAR (ENTER) (not the CLear key).
* You clear the machine using the ALL RESET button.
* 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-3** 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 80 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)	R=.85*45ØØØ,R*.6Ø_

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

Input	Display
(ENTER)	22950.

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

Input	Display
(R) (ENTER)	38250.

Now It's Your Turn

The concludes our discussion of using the **PC-3** 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-3**. In the next chapter we will examine the concepts and terms of the BASIC language, as it is used by the **PC-3**. Then you can begin to create your own, unique, problem-solving programs.

CHAPTER 4 CONCEPTS AND TERMS OF BASIC

In this chapter we will examine some concepts and terms of the BASIC language. Because the **PC-3** uses many features of BASIC 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 **Radio Shack PC-3** can process. Some of you, however, may need to desire to know more about how this computer uses numbers.

The **Radio Shack PC-3** 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 number of 1's, the second

shows the number of 1Ø's, the third the number of 1Ø's, and the fourth the number 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: one shows what the numbers are; the second 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:

$1.Ø \times 1Ø^{87}$ or 1.Ø E 87

The **PC-3** 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

Those of you who are unfamiliar with this type of notation should take some time to put in a few very large and very small numbers to note how they are displayed.

Limits

The largest number which the **PC-3** can handle is ten significant digits, with two-digit exponents. In other words, the largest number is:

9.999999999 E 99 = 9999999999ØØ
ØØØØØØØØØØØØØØØØØØØØØØ

and the smallest number is:

9.999999999 E –99 = .ØØØ
ØØØØØØØØØØØØØØØØØØØØØØ9999999999

Under certain circumstances, when numbers will be used frequently, the **PC-3** 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 under which this form is used are noted in 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-3** to treat a number as hexadecimal, put an ampersand (&) character in front of the numeral:

```
&A    = 10
&10   = 16
&100  = 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-3**, but there are special applications where it is convenient.

String Constants

In addition to numbers, there are many ways that the **Radio Shack PC-3** uses letters and special symbols. These letters, numbers, and special symbols are called characters. These characters are available on the **PC-3.**

```
1 2 3 4 5 6 7 8 9 0
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-3** 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"
"RADIO SHACK PC-3"
```

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-3** 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$
C$

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, 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 an 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 12 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-3**'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-3** has five **numeric operators.** These are the arithmetic operators which you used when exploring the use of the **PC-3** 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) + 5.0 - 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.

NOTE: Negative numbers may not be raised to a power with the ^ operator since you may obtain incorrect signs. If negative numbers are encountered in a program, convert the numbers to positive numbers using ABS before using the ^ operator. You will then have to change the result to the appropriate sign.

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 use 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-3** for string calculations is limited to only 79 characters.

NOTE: String quantities 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 are either True or False. The **PC-3** 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 explicit 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

(Cf. Values 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 BITS
32767	0111111111111111
⋮	
3	0000000000000011
2	0000000000000010
1	0000000000000001
0	0000000000000000
−1	1111111111111111
−2	1111111111111110
−3	1111111111111101
⋮	⋮
−32768	1000000000000000

The negative (NOT) of a binary number 0000000000000001 is taken as follows:

NOT		0000000000000001
(Negative)	→	1111111111111110

Thus, 1 is inverted to 0, and 0 to 1 for each bit, which is "to take negative (NOT)."
Then the following will result when 1 and NOT 1 are added together:

```
    0000000000000001  (1)
+)  1111111111111110  (NOT 1)
--------------------
    1111111111111111  (−1)
```

Thus, all bits become 1. According to the above number list, the bits become −1 in deciml 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 this equation, the following results are found

NOT 0 = −1
NOT −1 = 0
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 >= 10) AND NOT (A > 20)
(C = 5) OR (C = 6) OR (C = 7)
```

The **PC-3** 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, use 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 evaluation complex expressions the **PC-3** follows a predefined set of priorities which determine the sequence in which operators are evaluated. This can be quite significant.

5 + 2 * 3 could be:

5 + 2 = 7	**or**	**2 * 3 = 6**
7 * 3 = 21		**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 in 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 10 of its argument.

LOG (10) **is** **2**
LOG (1000) **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-3**.

CHAPTER 5 PROGRAMMING THE PC-3

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-3**. Let us reiterate, however, that 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-3.**

Programs

A **program** consists of a set of instructions to the computer. Remember the **PC-3** 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-3** 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:

```
10: PRINT "HELLO"
20: READ B (10)
30: 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-3** the order in which 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 (10, 20, 30, . . . 10, 30, 50 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.

```
10: PRINT "HELLO"
20: READ B(10)
30: END
```

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

```
10: PRINT "HELLO"
20: READ B(10)
30: 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 uses on the **PC-3.**)

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 uses on the **PC-3.**

Modes

You will remember that, when using the **PC-3** 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.

The RSV or ReSerVe mode enables you to designate and store predefined string variables and is used in more advanced programming (see Chapter 6).

Beginning to Program on the PC-3

After all your practice in using the **PC-3** 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 slide switch to the PROgram mode and enter this command:

Input	Display
NEW	>

The NEW command clears the **PC-3**'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-3** is in the PRO mode and enter the following program:

Input	Display
10 PRINT "HELLO"	**10: PRINT "HELLO"**

Notice that when you push (ENTER), the PC-3 displays your input, automatically inserting a colon (:) between the line number and the verb. Verify that the statement is in the correct format.

Now slide the selector switch to the RUN mode:

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 you 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 move 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, switch back to the PROgram mode.

You need to recall your program in order 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 ((BREAK)) 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 verticaly; 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 with to change:

Input	Display
↑	

Input	Display
◄◄◄◄◄	10: 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
GOODBYE"!	10PRINT"GOODBYE"!_

Don't forget to press (ENTER) at the end of the line. Switch into the RUN mode.

Input	Display
RUN	ERROR 1 IN 10

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

Switch back into the PROgram mode. You must be in the PROgram mode to make changes in a program. Using the ↑, recall the last line of your program.

Input	Display
↑	10: PRINT "GOODBYE"!

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	10: PRINT "GOODBYE"_

Now let's 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
◄	10: PRINT "GOODBYE"

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

Input	Display
INS	10: PRINT "GOODBYE "

Type in the !. The display looks like this:

Input	Display
!	10: PRINT "GOODBYE!"

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.

Example 3 — Using Variables in Programming

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

Using variables in programming allows much more sophisticated use of the **PC-3**'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 following 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:

```
10 INPUT "WORD?"; A$
20 B = LEN (A$)
30 PRINT "WORD␣␣IS␣␣"; B; "␣␣LETTERS"
40 END
```

␣ means space

Before you RUN the program, notice several new features. Line 30 of this program exceeds the 24-character maximum of the **PC-3**'s display. When a line is longer than 24 characters (up to the 79-character maximum), **PC-3** moves the characters to the left as the 24-character maximum is exceeded. This does not destroy the previous input. This moves 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	**10: INPUT "WORD?"; A$**

Use the (↑) and (↓) arrows to move through your program until you have reviewed the entire program. To review a line which contains more than 24 characters, move the cursor to the extreme right of the display and the additional characters will appear on the screen.

Input	Display
RUN	**WORD?_**
HELP	**WORD IS 4. LETTERS**
(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 40.

You may type 40 to Delete the entire line or use the (►) to position the cursor over the E in End. Edit line 40 so that it reads:

40: GOTO 10

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 on 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 on line 12Ø and the NEXT verb on line 15Ø 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-3** 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 the 16Ø 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 PC-3's Memory

You will remember that settings, ReSerVe keys, and functions remain in the computer even after it is turned OFF. Programs also remain in memory when you turn off the **PC-3**, or it undergoes an AUTO OFF. Even if you use the (BRK) , CLear or CA

keys, the programs will remain.

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

* You enter NEW before beginning programming.
* You initialize the computer using the ALL RESET button.
* 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-3** should serve to illustrate the exciting programming possibilities of your new computer. For more practice in programming exercises, please see Chapter 9.

CHAPTER 6 SHORTCUTS

The **PC-3** includes several features which make programming more convenient by reducing the number of keystrokes required to enter repetitive material.

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

This chapter discusses two additional features which can eliminate unnecessary typing — the DEF key and the ReSerVe mode.

The DEF Key and Labelled Programs

Often you will want to store several different programs in the **PC-3**'s 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:

* Put a label on the first line of each program that you want to reference. The label consists of a single character in quotes, followed by a colon.

```
10: "A": PRINT "FIRST"
20: END
80: "B": PRINT "SECOND"
90: 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. This area has been darkened on your keyboard to make it easier for you to remember.

* To execute the program, instead of typing RUN 8Ø or GOTO 1Ø, you need only to 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.

ReSerVe Mode

Another timesaving feature of the **PC-3** is the ReSerVe mode.

Within the memory of the **PC-3**, 47 characters are designated for "Reserve Memory". You can use this memory to store frequently-used expressions, which are then recalled by a simple two-keystroke operation.

NOTE: You **store** the strings in the ReSerVe mode and **recall** them for use in the RUN and PROgram modes.

Try this example of storing and recalling a reserved string.

Switch the **PC-3** into ReSerVe mode by moving the slide switch to the RSV position.

Type NEW, followed by the (ENTER) key. This will clear out any previously stored characters in the same way NEW clears out stored programs in the PROgram mode.

Type (SHIFT) followed by 'A'.

Input	Display
(SHIFT) **A**	A: _

Notice that the "A" appears in the display at the left followed by a colon.

Enter the word 'PRINT' and press the (ENTER) key.

Input	Display
PRINT (ENTER)	A: PRINT

A space appears after the colon signalling you that 'PRINT' is now stored in the reserve memory under the letter A.

Switch the **PC-3** into PROgram mode. Type NEW followed by (ENTER) to clear the program memory. Type '1Ø' as a line number and then press (SHIFT) and the 'A' key:

Input	Display
1Ø (SHIFT) A	1Ø PRINT_
(ENTER)	1Ø: PRINT

Immediately the word "PRINT" will appear in the display after the line number.

Any character sequence can be stored in ReSerVe Memory. The stored strings can be recalled at any time in either the program or the RUN mode by typing (SHIFT) and the key that the string is stored under. The keys available are the same as those used with DEF, i.e., those in the dark area of the keyboard.

To edit a stored character sequence, switch into the ReSerVe mode and press (SHIFT) and the key under which the sequence is stored. You can then edit using the Left Arrow, Right Arrow, DEL, and INS keys in the same way as in other modes.

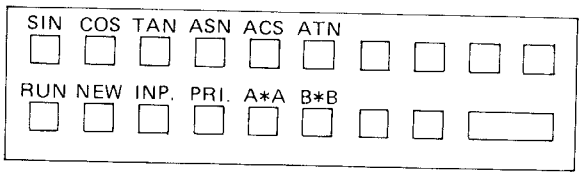
When the last character in a stored sequence is a '@' character, it is interpreted as (ENTER) when the sequence is recalled. For example, if you store the string "GOTO 100@" under the 'G' key, typing (SHIFT) and 'G' in the RUN mode immediately starts execution of the program at line 100. Without the '@' character, you must press (ENTER) after the (SHIFT) and 'G' to begin execution.

Templates

Two templates are provided with the **PC-3**. You can use these templates to help you remember frequently used ReSerVe sequences or DEF key assignments. After you have labelled the programs or created the sequences, mark the templates so you know what is associated with each key. You can then execute programs or recall sequences 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. You might also store frequently-used BASIC commands and verbs in the Reserve Memory and mark a template to speed to entering BASIC programs.

Example:



CHAPTER 7 USING THE PC-3 PRINTER/CASSETTE INTERFACE

The **PC-3** Printer/Cassette Interface allows you to add a printer and cassette interface to your **Radio Shack PC-3** Pocket Computer. The **PC-3** Printer/ Cassette Interface features:

* 24-character-wide thermal printer with approximately a 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, data, and reserve key settings.
* Filenames and passwords on tape for control and security.
* Built-in rechargable Nickel-Cadmium batteries for portability.
* Recharger supplied.

Introduction to the Machine

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:

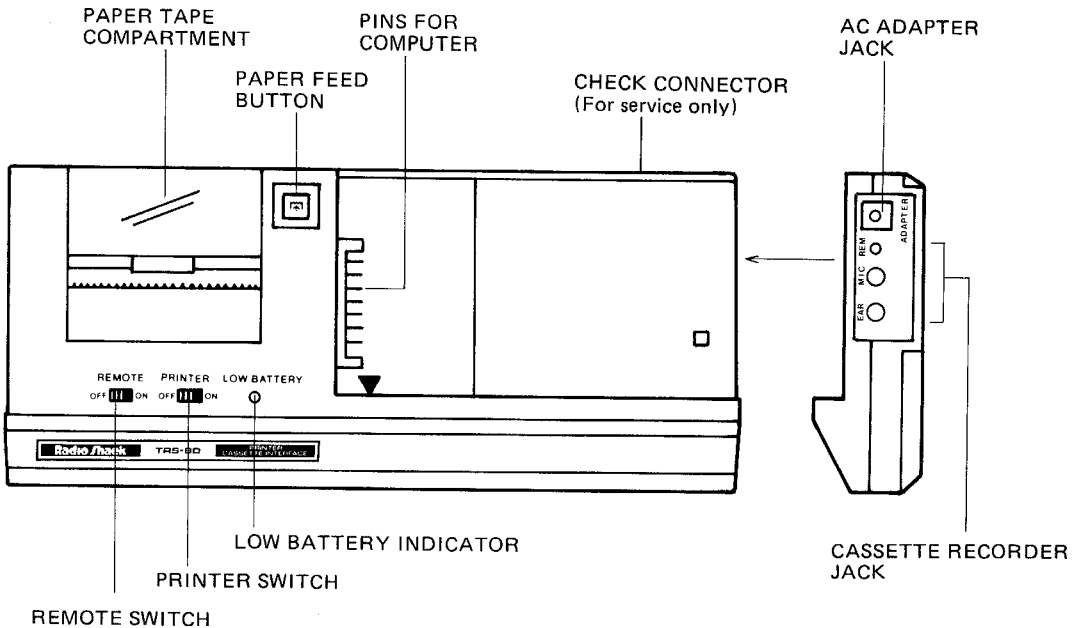


Figure 14. Printer/Cassette Interface (Front View)

* 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.

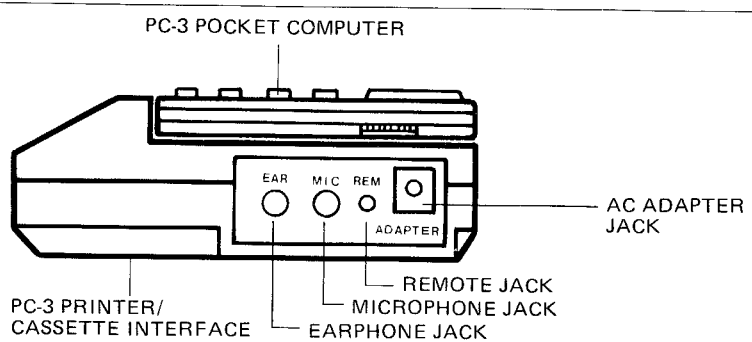


Figure 15. PC-3 Printer/Cassette Interface (Right Side View)

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.

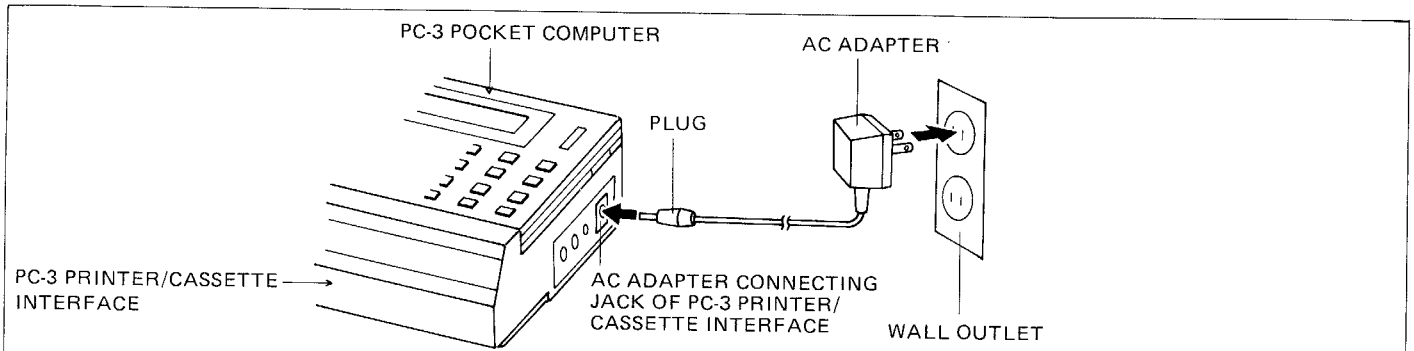


Figure 16. How to Connect the AC Adapter

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-3 Pocket Computer to the PC-3 Printer/Cassette Interface

To connect the **PC-3** 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 ALL RESET button to clear the Computer.

2. Remove the protective pin cover from the left side of the Computer and snap it into place on the bottom of the Printer/Cassette Interface.

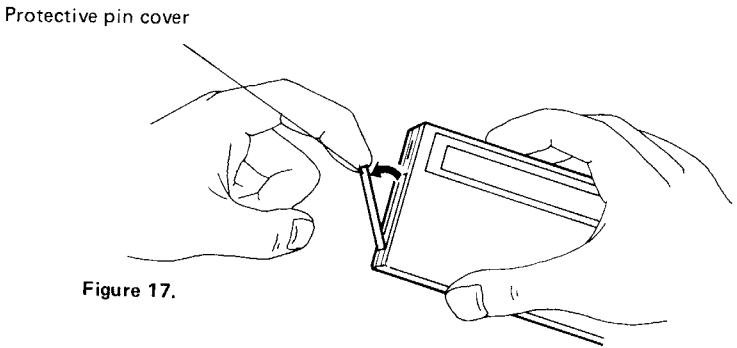


Figure 17.

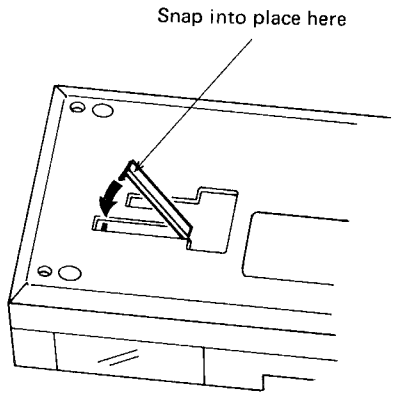


Figure 18.

3. Place the Computer on the Printer/Cassette Interface shown in Fig. 19.

4. Lay the Computer down flat.

5. 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 mate easily, STOP and check to see that the parts are correctly aligned.

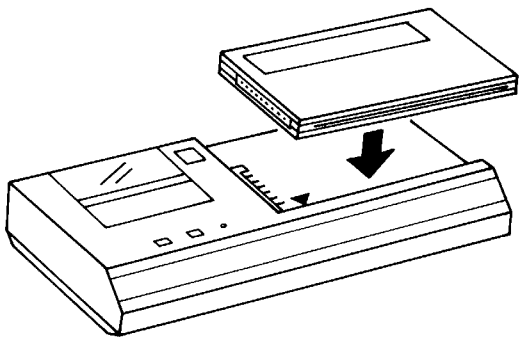


Figure 19.

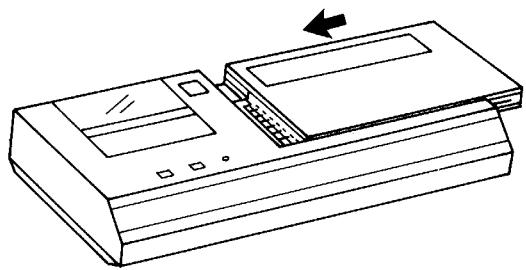


Figure 20.

6. To use the Printer, turn on the **PC-3** 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. 21)

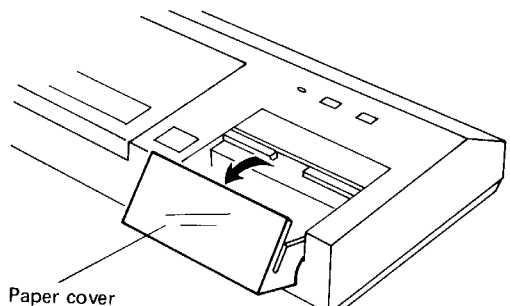


Figure 21.

(3) Insert the leading edge of the roll of paper into the slot located in the paper tape compartment. (Fig. 22) (Fig. 23)
(Any curve or crease near the beginning of the paper makes insertion difficult.)

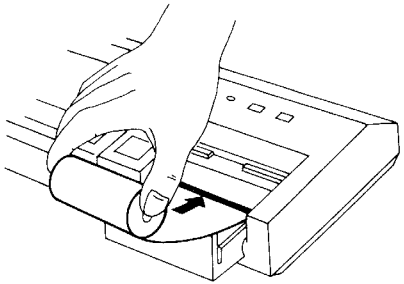


Figure 22.

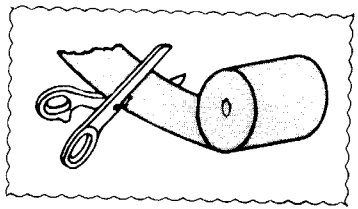


Figure 23.

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 the figure.

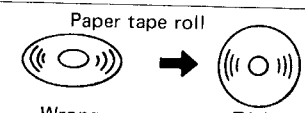


Figure 24.

(4) Turn on the Printer switch and press the paper feed button until the paper comes out of the Printer mechanism. (Fig. 25)

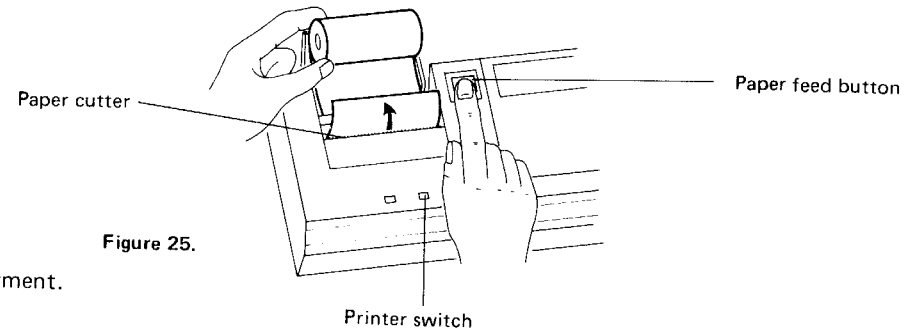


Figure 25.

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

(6) Close the paper cover. (Fig. 26.)

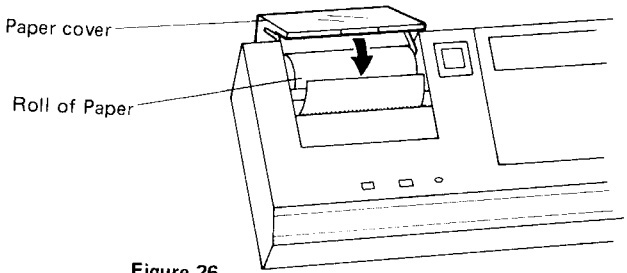


Figure 26.

- 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 to 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

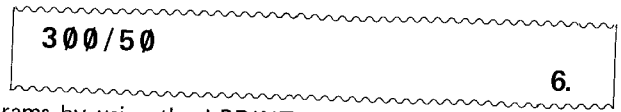
If you are using the **PC-3** Computer as a calculator, you may use the **PC-3** Printer to simultaneously print your calculations. This is easily accomplished by pressing the (SHIFT) key and then the (ENTER) key (P ↔ NP). (The printer indicator "P" will be displayed. If not, press the (SHIFT) and (ENTER) 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

300 / 50

Paper



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, since both the display and the Printer are 24 characters wide. The only difference is that, if you PRINT something to the display which is longer than 24 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 (see Relationship of Two Variables example in Chapter 9).

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 causes a paper misfeed 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. (Release the paper misfeed.) 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 or reserve memory on tape.
CLOAD	Retrieves a program or reserve memory 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 unambigous 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 other 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.

NOTE: Whenever you wish to read in something from tape, it is essential that the tape be positioned on one of these leader tone areas.

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 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.

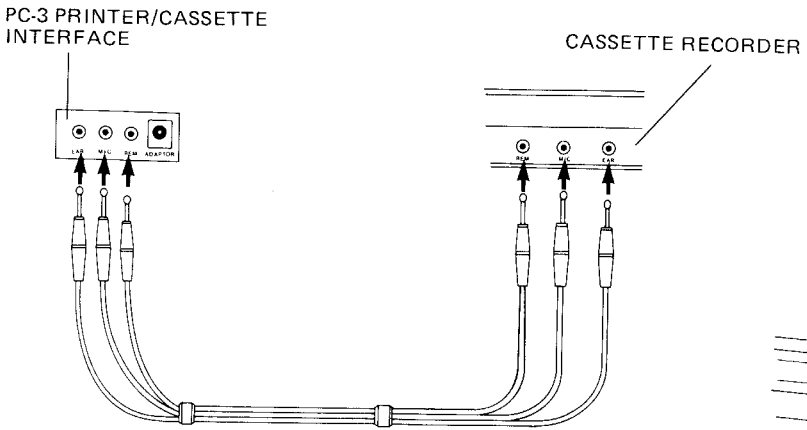


Figure 27. Cassette Cables and Interface Jacks

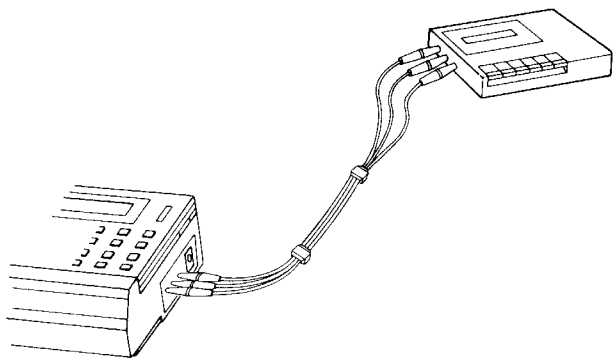


Figure 28. Recorder Connected to Interface

- To transfer program 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 is not turned on the Printer/Cassette Interface when a 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 the ALL RESET may be required. Be sure to restore adequate power to 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. At this time, you should hear a shrill buzzing sound, and the tape should be turning. Also, the "BUSY" indicator should light up. This tells you that the computer is "busy" transferring your program from memory to the tape. If this does not happen, start again from the beginning of the section.

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, we can read it back into memory from the tape as explained in the next section.

2. Collating the Computer and Tape Contents

Now that the 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 computer's memory. Erase that portion of 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 we 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 the above step (1).

Precautions for collation and transfer

The program is recorded on tape as illustrated below:

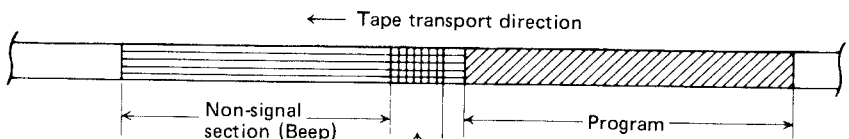


Figure 28.

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 are 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 each category.

Commands

Program Control

- CONT
- GOTO*
- NEW
- 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

- BEEP*
- 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
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

BEEP
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.

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.

If the **PC-3 Computer** is in the PROgram or RUN mode, program memory is loaded from the tape. When the Computer is in the ReSerVe mode, reserve memory is loaded. Care should be taken not to load programs into reserve memory or reserve characters into program memory.

Examples

CLOAD Loads the first program from the tape.

CLOAD "PRO3" Searches the tape for the program named "PRO3" and loads it.

Notes:

1. The computer cannot identify the stored contents as a program or a reserve. Therefore, if a mode is designated incorrectly, the reserved contents may be transferred to the program area or the program to the reserve area, causing the computer to remain inoperative. If this happens, reset the computer by pressing the RESET button on the back of the computer.

2. If the designated filename is not retrieved, the computer will continue to search for the filename even after the tape reaches the end. In this case, stop the retrieval function by pressing the ON (BRK) key. This applies to MERGE, CHAIN, CLOAD? and INPUT# commands to be described later.

3. 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 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 onto the cassette tape without a specified filename.

The second form of the CSAVE command writes all of the programs in memory onto the cassette tape and assigns the indicated filename.

The third form of the CSAVE command writes all of the programs in memory onto the cassette tape without a specified filename 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 onto the cassette tape and assigns the indicated

filename and password.

It the **PC-3** Computer is in PROgram or RUN mode, program memory is loaded to the tape. When the **PC-3** Computer is in the ReSerVe mode, reserve memory is loaded.

Examples

CSAVE "PRO3", "SECRET" Saves the programs now in memory onto 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. If no expression is provided, execution begins with the first line.

GOTO differs from RUN in four 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 100 Displays line number 100.

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

LIST 100, 200 Lists the statements between line numbers 100 and 200.

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.

Example

MERGE Merges the first program from the tape.

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

Note: For example, assume the Computer memory contains the following program:

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

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 ALLOWANCE"
20: INPUT "ENTER METHOD:  "; A
10: "DEP1" : REM >> SECOND MODULE <<
20: PRINT "INTEREST CHARGES"
30: INPUT "AMOUNT 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 – below).
It is important that you review the following information before proceeding with any further editing or programming:

IMPORTANT NOTES:

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

Examples:

```
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**

Abbreviations: none

Purpose

The NEW command is used to clear the existing program or reserve memory.

Use

When used in the PROgram mode, the NEW command clears all programs and data which are currently in memory.

When used in the ReSerVe mode, the NEW command clears all existing reserve memory.

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

Examples

NEW Clears program or reserve memory

1 **PASS** "character string"

Abbreviations: none

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 only way to remove this protection is to execute another PASS statement with the same password or to enter NEW (which erases the programs).

Examples

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

1 **RUN**
2 **RUN** expression

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 lowest numbered line greater than or equal to the value of the expression.

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 100 Executes the program which begins at line number 100.

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 and 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

```
10 "X": AREAD N
20 PRINT N^2
30 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 are displayed by the PRINT verb at the start of program execution, the following is 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 or PRINT "String", "String", the contents on the right of the display are stored.
- 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"..., meaningless contents may be stored.

1 **BEEP** expression

Abbreviations: B., BE., BEE.

Purpose

The BEEP verb is used to produce an audible tone.

Use

The BEEP verb causes the **PC-3** Computer to emit one or more audible tones at 4 kHz. The number of beeps is determined by the expression, which must be numeric. The expression is evaluated, but only the integer part is used to determine the number of beeps.

BEEP may also be used as a command using numeric literals and predefined variables. In this case, the beeps occur immediately after the (ENTER) key is pressed.

Examples

```
10 A = 5 : B$ = "9"
20 BEEP 3          Produces 3 beeps.
30 BEEP A          Produces 5 beeps.
40 BEEP (A+4)/2    Produces 4 beeps.
50 BEEP B$         This is illegal and will produce an ERROR 9 message.
60 BEEP -4         Produces no beeps, but does not produce an error message.
```

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 in connection 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 the 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.

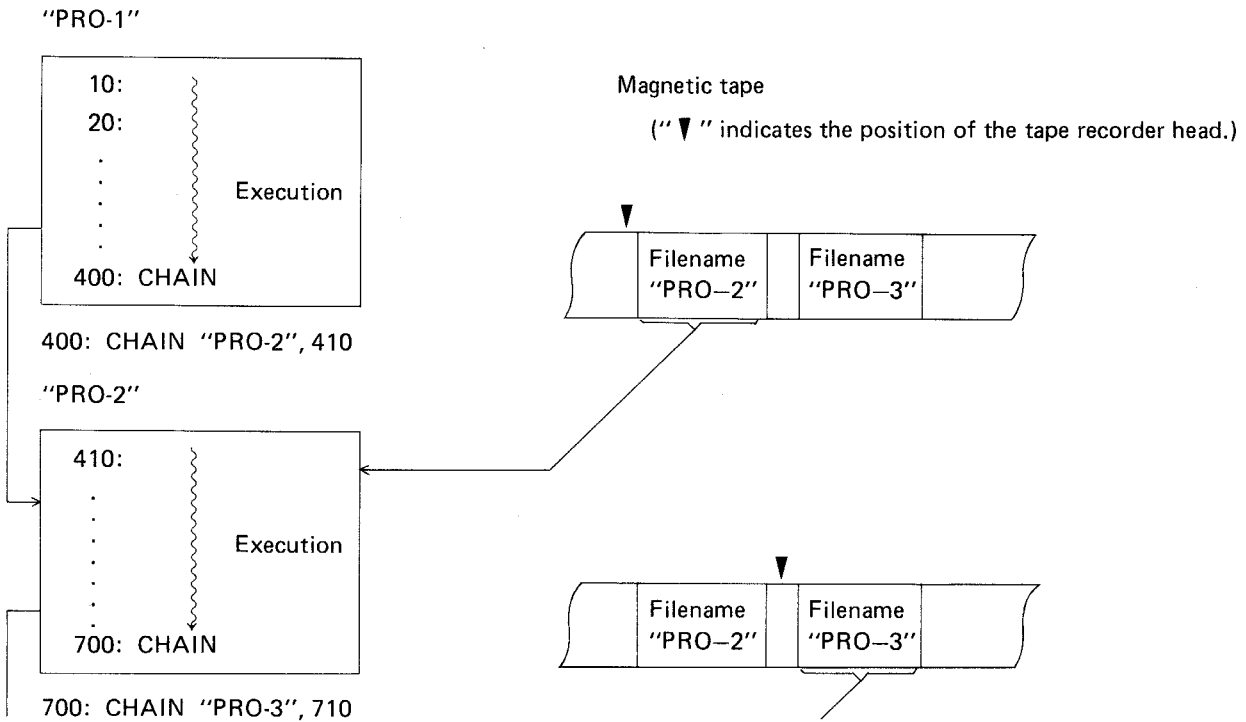
The third form of CHAIN searches the tape for the program whose name is indicated by "filename", loads the program, and begins execution with the lowest line number.

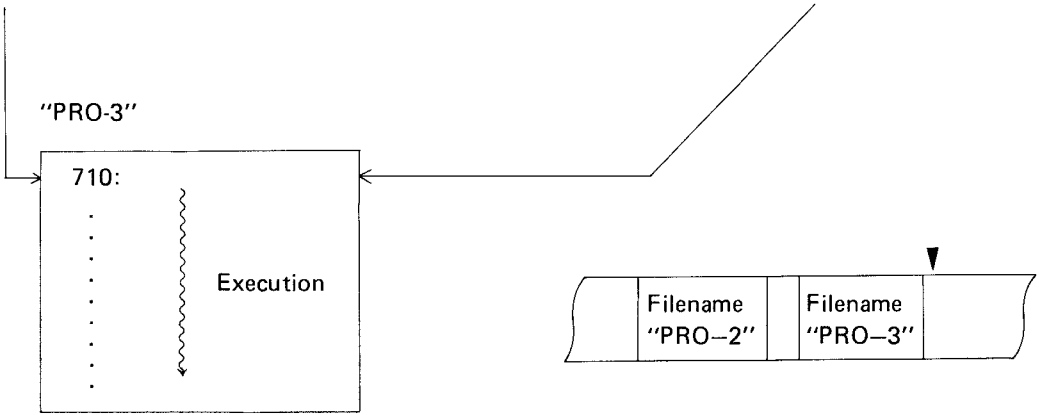
The fourth form of CHAIN will searches the tape for the program whose name is indicated by "filename", loads the program, and begins execution with the line number indicated by the expression.

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.





990: END

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). CLEAR does reset numeric variables to zero and string variables to NUL.

Examples

```
10 A = 5 : DIM C(5)
20 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-3** Computer 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

```
10 DEGREE
20 X = ASN 1
```

X now has a value of 90, i.e., 90 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

Prupose

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 statement 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

```
10 DIM B(10)
20 FOR I = 1 TO 10
30 READ B(I)
40 NEXT I
50 DATA 1, 2, 3, 4, 5, 6
70 DATA 7, 8, 9, 10
```

Sets up an array.

Loads the values from the DATA statement into B( ).
B(1) will be 1, B(2) will be 2, B(3) will be 3, etc.

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(26) and 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 not have the same name. A sting 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 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.

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

```
10 PRINT "HELLO"
20 END
30 PRINT "GOODBYE"
40 END
```

With these programs in memory a 'RUN 10' prints 'HELLO', but not 'GOODBYE'. 'RUN 30' 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 or equal to 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 expression 2, execution continues with the statment 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 and expression 2 may have any value in the numeric range. When expression 1 and expression 2 are compared, only

the integer part is used. Expression 3 must be an integer in the range of −32768 to 32767; it may not be zero.

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

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

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

```
40 FOR N = 10 TO 0 STEP -1
50 PRINT N
60 NEXT N
```

This group of statements counts down 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0.

```
70 FOR N = 1 TO 10
80 X = 1
90 FOR F = 1 TO N
100 X = X * F
110 NEXT F
120 PRINT X
130 NEXT N
```

This group of statements computes and prints N factorial for the numbers from 1 to 10.

1 **GOSUB** expression

Abbreviations: GOS., GOSUB.

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 statments several time 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-3** Pocket Computer 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

```
10 GOSUB 100
20 END
100 PRINT "HELLO"
110 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

```
10 INPUT A$
20 IF A$ = "Y" THEN GOTO 50
30 PRINT "NO"
40 GOTO 60
50 PRINT "YES"
60 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-3** Pocket Computer 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 rusults 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% gradient.

Examples

```
10 GRAD
20 X = ASN 1        X now has a value of 100, i.e., a 100% 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 program, 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 or not the condition is True.

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

Examples

```
10 INPUT "CONTINUE?"; A$
20 IF A$="YES" THEN GOTO 10
30 IF A$="NO" THEN GOTO 60
40 PRINT "YES OR NO, PLEASE"
50 GOTO 10
60 END
```

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

Note:

This unit is capable of calculation for a mantissa of up to 12 digits. To increase accuracy, however, the mantissa is calculated inside the unit up to 12 digits, the result of which is rounded to the 10th digit for display. For example, 5/9 and 5/9 * 9 are calculated as follows:

5/9 ⟶ 5.55555555555E−01
↑ This is rounded to the 10th digit.

Display ⟶ 5.555555556E−01

5/9 * 9 ⟶ 4.99999999999E00
↑ This is rounded to the 10th digit.

Display ⟶ 5.

Thus, calculations are carried out for mantissas of up to 12 digits. This may cause a difference in the results of calculations when performed in succession and independently.

[Example 1] $3^2 - 9 =$

Successive calculation:	3 (SHIFT) (^) 2 (−)	9 (ENTER)	→ −9.E−11
Independent calculation:	3 (SHIFT) (^) 2 (ENTER)		→ 9.
	(−) 9 (ENTER)		→ Ø.

Even in the IF statement, this difference may cause the program not to work as planned for any successive calculations.

[Example 2]
```
10 INPUT A
20 IF A^2>=9 THEN 5Ø
```

With A = 3, 3 ^ 2 results in 8.99999999991E ØØ, making an IF statement unformulated.

In this case, reprogram the calculation by using variables so that it is independent, as follows:

```
10 INPUT A
15 B=A^2
20 IF B>=9 THEN 5Ø
```

The result of A ^ 2 is substituted for a variable, which is used to formulate conditional expression.

Power calculations are based on log x and 10^x, thus tending to cause a difference in the results from those calculated inside the computer.

$$A \wedge B \longrightarrow 10^{B \log A}$$

- When the A is negative, B must be an integer.

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

Abbreviations: 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 with the cursor next to it. 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 alphabetic characters are entered for a numeric variable, the variable is assigned a value of zero. if the (ENTER) key is pressed and no input is provided, the variable retains the value it had before the INPUT statement.

Examples

10 INPUT A	Clears the display and puts a question mark at the left edge.
20 INPUT "A = " ; A	Displays "A =" and then displays the input data continuously.
30 INPUT "A = " , A	Displays 'A =' When the data is input, "A =" disappears and then the data is displayed.
40 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. L

See also: INPUT, PRINT #, READ

Purpose

The INPUT # verb is used to enter one or more 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 read from the tape and assigned to the 26 preallocated storage locations. They can be used by referring to variables A ~ Z and A$ ~ Z$, as appropriate.

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, variables are read from the tape, starting at the current location, and loaded into the variables in the order in which they appear in the variable list. If there are not enough values on the tape for the number of variables in the list, then zero or NUL values are assigned to the remainder.

With the fourth form, the tape is searched for the indicated filename and the variables are loaded from the values saved in that file.

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, including the simple variable of the same name; i.e., B(*) restores B and 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

1Ø INPUT # A,B,C,	Reads three values from the current position of the tape.
2Ø INPUT # "FIL2"; D, E, F	Searches the tape for the file 'FIL2' and reads in three values.
3Ø INPUT # "FIL3"; G(*)	Searches the tape for the file 'FIL3' and reads in G and 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 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	Assigns 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$.
40 IFI > 10 THEN LET Y$ = 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, USINT, and WAIT

Purpose

The LPRINT verb is used to print information on the Printer of the optional **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.

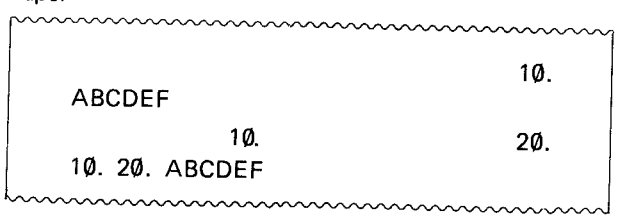
If an LPRINT statement contains more than 24 characters, the first 24 are printed on one line, the next 24 on the next line, and so forth.

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

Examples

```
10 A=10 : B=20 : X$="ABCDEF"
20 LPRINT A
30 LPRINT X$
40 LPRINT A, B
50 LPRINT A; B; X$
```

Paper



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 generally described under FOR. The numeric variable in a NEXT statement must match the numeric variable in the corresponding FOR.

Examples

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
```

Prints the numbers from 1 to 10.

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-3** Computer 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 Computer can not distinguish between commas in expressions and commas **between** expressions.

Examples

```
10 INPUT A
20 ON A GOTO 100, 200, 300
30 GOTO 900
100 PRINT "FIRST"
110 GOTO 900
200 PRINT "SECOND"
210 GOTO 900
300 PRINT "THIRD"
310 GOTO 900
900 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: PA., 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-3** Computer 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 value is printed at the far left end of the display.

With the second form of the PAUSE statement, the display is divided into two 12-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.

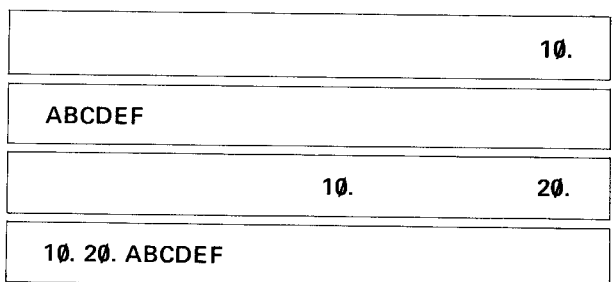
PAUSE statements are not affected by the PRINT=LPRINT statement (see PRINT).

While it is possible to write PAUSE statements which would display more than 24 characters, only the leftmost 24 appear in the display. There is no way to see the other characters.

Examples

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

20 PAUSE A

30 PAUSE X$

40 PAUSE A, B

50 PAUSE A; B; X$
```

Display



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 Printer of the **PC-3** Printer/Cassette Interface.

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 expres-

sion, the value is printed at the far left end of the display.

With the second form of the PRINT statement, the display is divided into two 12-characters 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.

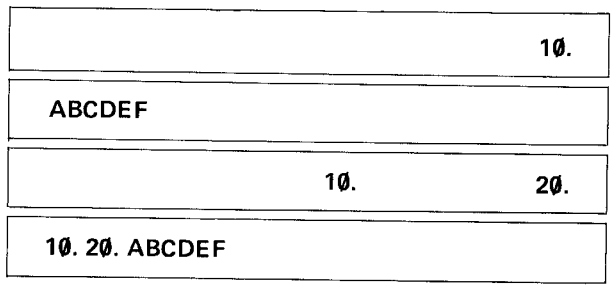
The fourth and fifth forms of the PRINT statement do no printing. The fourth form causes all PRINT statements which follow it in the program to be treated as if they were LPRINT statements. The fifth form resets the fourth condition so that the PRINT statements will again work with the display.

While it is possible to write PRINT statements which would display more than 24 characters, only the leftmost 24 appear in the display. There is no way to see the other characters.

Examples

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

20 PRINT A

30 PRINT X$

40 PRINT A, B

50 PRINT A; B; X$
```

Display



1 **PRINT #**
2 **PRINT #** "filename"
3 **PRINT #** "filename"
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 one or more 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 storage locations (variables A ~ Z and A$ ~ Z$) are stored on the tape.

With the second form, the 26 preallocated storage locations are stored on the tape under the designated filename.

With the third form, the indicated variables are stored on the tape, starting at the current location.

With the fourth form, indicated variables are stored on the tape 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, including the simple variable of the same name, i.e., B(*) saves B and 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

1Ø PRINT # A, B, C	Saves three values on the tape at the current position.
2Ø PRINT # "FIL2" ; D, E, F	Saves three values on the tape under the filename "FIL2".
3Ø PRINT # "FIL3" ; G(*)	Saves G and all 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-3 Pocket Computer 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 repect to a radius, i.e., 360° is 2PI radians, since the circumference of a circle is 2 PI times the radius.

Examples

```
10 RADIAN
20 X = ASN 1
```

X now has a value of 1.570796327 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-3** Computer 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-3** Computer 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

```
10 RANDOM
20 X = RND 1
```

When run from line 20, the value of X is based on the standard seed. When run from line 10, a new seed is used.

1 **READ** variable list

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

Abbreviations: REA.

Sea 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 in 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

```
10 DIM B(10)
20 FOR I = 1 TO 10
30 READ B(I)
40 NEXT I
50 DATA 1, 2, 3, 4, 5, 6
60 DATA 7, 8, 9, 10
```

Sets up an array

Loads the values from the DATA statement into B( ) — B(1) is 1, B(2) is 2, B(3) is 3, etc.

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 reread 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-3** Pocket Computer 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

```
10 DIM B(10)
20 FOR I = 1 TO 10
30 READ B(I)
40 RESTORE
50 NEXT I
60 DATA 10
```

Set up an array.

Assigns the value 10 to each of the elements of B( ).

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

```
10 GOSUB 100
20 END
100 PRINT "HELLO"
110 RETURN
```

When run, this program prints the word "HELLO" one time.

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-3** Computer execution halts and a message is displayed, such as 'BREAK IN 2ØØ' where 2ØØ 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. Pressing the Left Arrow or Right Arrow keys after a STOP restores the display to its condition prior to the STOP.

Examples

1Ø STOP Causes "BREAK IN 1Ø" 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

```
10 TRON
20 FOR I = 1 TO 3
30 NEXT I
40 TROFF
```

When run, this program displays the line numbers 10, 20, 30, 30, 30 and 40.

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-3** Computer 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

```
10 TRON
20 FOR I = 1 TO 3
30 NEXT I
40 TROFF
```

When run, this program displays the line numbers 10, 20, 30, 30, 30 and 40.

1 **USING**
2 **USING** "editing specification"

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

Examples

```
10 A = 125 : X$ = "ABCDEF"

20 PRINT USING "##.##^^^";A

30 PRINT USING "&&&&&&&&&";X$

40 PRINT USING "####&&&";A;X$
```

Display

```
 1.25E Ø2
ABCDEF
125ABC
```

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-3** Pocket Computer halts execution after a PRINT command until the (ENTER) key is pressed. The WAIT command causes the **PC-3** Computer 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-3** Computer 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 has the value of the last key pressed on the keyboard. Enter, CL, CA, SHIFT, DEF, Up Arrow, Down Arrow, Left Arrow, and Right Arrow all have a value of NUL. INKEY$ is used to respond to the pressing of individual keys without waiting for the ENTER key to end the input. For example, these statements "wait" for a non-NUL key to be pressed:

```
10 A$ = INKEY$
20 B = ASC INKEY$
30 IF B = 0 THEN GOTO 10
40 IF B....
```

Lines 40 and beyond contain tests for the key and the actions to be taken. On first executing the program, the value of INKEY$ is NUL, since the last key pressed was (ENTER) . If INKEY$ is used following PRINT or PAUSE, the contents of the display are read instead of a key pass.

```
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 following reset, MEM has a value of 1438.

```
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 10-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-3** Pocket Computer 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).

If the same function is to be used two or more times in succession, parentheses must be used:

LOG (LOG 1ØØ) not LOG LOG 1ØØ

1 **ABS** numeric expression

ABS is a 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-3** Computer 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-3** Computer is in decimal degree, radian, or gradient mode for angles. ASN .5 is 30 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-3** Pocket Computer 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-3** Computer is in decimal degree, radian, or gradient mode for angles. COS 60 is .5 in the decimal degree mode.

1 **DEG** numeric expression

The DEG function coverts 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° 10' 44.5'' is represented as 55.10445. In DEG format, the integer portion is degrees and the decimal is decimal degrees. DEG 55.10445 is 55.17902778.

1 **DMS** numeric expression

DMS is a numeric function which converts an angle argument is DEG format to DMS format (see DEG). DMS 55.17902778 is 55.10445.

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 10 of its numeric argument. LOG 100 is 2.

1 **LN** numeric expression

LN is a numeric function which retuns the logarithm to the base e (2.718281828) of its numeric argument. LN 1Ø 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 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:

	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-3** Pocket Computer 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-3** Computer is in decimal degree, radian, or gradient mode for angles. SIN 3Ø is .5

1 **SQR** numeric expression

SQR is a numeric function which returns the square root of its argument. It is identical to the use of the special square root symbol ($\sqrt{\ }$) 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-3 Computer** is in decimal degree, radian, or gradient mode for angles. TAN 45 is 1.

(CALCULATION RANGE)

Functions	Dynamic range
$y \wedge x \quad (y^x)$	$-1 \times 10^{100} < x \log y < 100$ $\begin{pmatrix} y = 0,\ x \leqq 0\text{: ERROR 2} \\ y = 0,\ x > 0\text{: 0} \end{pmatrix}$ (Ex.) 0^0 [ENTER] → ERROR 2 0^5 [ENTER] → 0. (−4)^0.5 [ENTER] → ERROR 2 If $y < 0$ x must be an integer.
SIN x COS x TAN x	DEG: $\lvert x \rvert < 1 \times 10^{10}$ RAD: $\lvert x \rvert < \frac{\pi}{180} \times 10^{10}$ GRAD: $\lvert x \rvert < \frac{10}{9} \times 10^{10}$ In TAN x, however, the following cases are excluded. DEG: $\lvert x \rvert = 90\ (2n-1)$ RAD: $\lvert x \rvert = \frac{\pi}{2}\ (2n-1)$ GRAD: $\lvert x \rvert = 100\ (2n-1)$ (n: integer)

Functions	Dynamic range
$\text{SIN}^{-1}\ x$ $\text{COS}^{-1}\ x$	$-1 \leqq x \leqq 1$
$\text{TAN}^{-1} x$	$\lvert x \rvert < 1 \times 10^{100}$
LN x LOG x	$1 \times 10^{-99} \leqq x < 1 \times 10^{100}$
EXP x	$-1 \times 10^{100} < x \leqq 230.2585092$
$\sqrt{x}$	$0 \leqq x < 1 \times 10^{100}$

- 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.

Note: Power calculation is performed from m logx and 10^x calculations.

$Y \wedge X \rightarrow 10^{x \log y}$

Therefore, there is inevitably a difference, to some extent, from the true value in the computer. This difference does not usually appear on the display. However, it is accumulated depending on calculation contents, such as continuous calculations, and may appear on the display.

Example: 16−2^2^2 (ENTER) → 6.E−10

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 argumennt 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-3** 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. For example, CHR$ 65 + 4 is interpreted as (CHR$ 65) + 4, which is an illegal mixture of string and numeric quantities; CHR$ (65 + 4) is valid.

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.
If the string contains alphanumeric character, any numeric character on the right of the alphanumeric is ignored.
VAL (2 LBS 5 OZ) will return "2".
Space is usually regarded as non-existing. However, if space is included in the exponent portion (after E), any numeric character on the right of space is ignored.

CHAPTER 9 PROGRAMMING EXAMPLES

This chapter presents a series of programming examples which illustrate some of the potential programming capabilites of your **PC-3** Computer. Each example is briefly discussed to indicate the logic and structure of the program and the way in which the **PC-3** Computer is being used. This discussion is not meant to be a complete guide to programming. New programmers should consult a separate book on how to program.

Loan Payments

This program illustrates how the **PC-3** Computer can be used to calculate the size of a loan payment and the total cost of the loan. The program first solicits the amount borrowed, the rate of interest, and the number of months that the loan will run. It then calculates the loan payment using this formula:

$$A = \frac{P * (1 + I)^N * I}{(1 + I)^N - 1}$$

Where: A is the monthly mortgage payment
P is the Principal; the amount borrowed
I is the interest for 1 month expressed as a decimal fraction (i.e., 1% = .Ø1)
N is the number of months

Then the program computes the total cost of the loan over the entire loan period and the total amount of interest.

The program asks for the interest for a whole year because this is the basis usually used to discuss interest.

Loan Payment Calculator Program

```
10: INPUT "PRINCIPAL? "; P

20: INPUT "YEARLY_INTEREST?"; I
30: I = 1/12
40: I = 1/100

50: INPUT "MONTHS? "; N

60: T = (P*((1 + I)^N)) *I
70: B = ((1 + I)^N) -1
80: A = T/B
90: A = ( INT ((A *100) + .5))/100
100: PRINT "MO. PAYMENT = " ; A

110: Z = A *N
120: PRINT "TOTAL_COST = " ; Z

130: X = Z-P
140: PRINT "TOTAL_INTEREST =" ; X

150: INPUT "ANOTHER? "; Q$
160: IF ( LEFT$ (Q$,1) = "Y") THEN GOTO 10
170: END
```

Get the amount borrowed

Get the interest for a whole year
Divide by 12 to get the interest for a month
Divide by 100 to turn percent into a decimal fraction

Get number of months

Compute top half of formula
Compute bottom half of formula
Divide top by bottom
Convert to even cents
Display monthly payment amount

Multiply monthly amount times months for total
Display total cost

Subtract principal from total cost to get interest
Print interest

Ask for repeat
Go back to top if first character is 'Y'
Otherwise end

Note: The computation in line 90 is a little programming "trick" for rounding off numbers to a desired precision. Multiplying by 100 moves the first two decimal digits to the left of the decimal point. Taking the integer part of this with the INT function throws away any extra decimal digits. .5 is added first so that it will round up if the part which is to be thrown away is over .5. The amount is then divided by 100 again to restore its prior scale.

Sort writing

When writing programs, you often need to get items into a particular order, i.e., to sort them. Many different sorting techniques have been developed, each of which is better or worse for particular circumstances. One of the simplest sorting techniques is the "sort by search". In this technique the program scans an array of unsorted data looking for the largest item. It puts this in the top element of a new array and goes back to look for the next largest item. It puts this item in the next element of the array, and so on. Each element selected in the unsorted array is then set to a very small number so it won't be found on the next search.

Sorting Program

```
 10: INPUT "HOW MANY_ ITEMS TO SORT? "; N
 20: DIM 0(N), S(N)

 30: FOR I = 1 TO N
 40: INPUT O (I)
 50: NEXT I

 60: PAUSE "SORTING"
 70: FOR I = 1 TO N

 80: T = 1
 90: FOR J = 1 TO N
100: IF (0 (J) > 0 (T)) THEN LET T = J
110: NEXT J

120: S (I) = 0 (T)
130: 0 (T) = – 9.999999999E99

140: NEXT I
```

Find out how many values this time
Allocate space for an array to hold the data-0 ( ) is for unsorted data, S ( ) for sorted
Lines 30–50 are a loop to read in the data

Read in each value

Announce that the sort is starting
Outside loop indexes through S ( ) indicating where the next largest value is to be put
Arbitrarily set pointer for the largest value to the first element in 0 ( )
Loop through array of unsorted data
If a larger value is found, change T to point to the largest so far

Put this value in the next element of the sorted array
Set that element in the unsorted array to the smallest possible number so that it won't be used again
End of loop on sorted array

```
150: BEEP 2
160: PAUSE "DONE SORTING"

170: INPUT "DISPLAY OR PRINTER? "; A$
180: IF ( LEFT$ (A$,1) = "P") THEN PRINT = LPRINT

190: FOR I = 1 TO N
200: PRINT S (I)
210: NEXT I

220: PRINT = PRINT
230: END
```

Announce that sort is done

Ask where output should go
If printer is selected, set output to printer

Loop through sorted array
Print largest through smallest

Reset to original condition

Slot Machine Simulation

This program simulates the behavior of a simple slot machine. The model simulated is based on three wheels covered with pictures of objects. One object from each wheel shows through the window at a time. In this simulation there are three objects which alternate around each of the three wheels. On each play the wheels are spun and travel freely for a period of time. Then they begin to slow down and gradually come to a stop. The stopping point of each wheel is random. If the same three objects show in all three windows at the end of the play, then you win.

Notice that this program has been written so that it is easy for you to modify the number of objects, the speed of play, and the combinations which produce a win. At the beginning, variables are set to control the speed during the spin (i.e., the amount of time it takes for each new object to appear in the window). One controls the rate during the initial free-wheeling portion, and another controls the rate during the slow down period. A third variable is used to control the length of other displays.

Another variable determines the number of objects to be used. Although only three have been used here, it is easy to change this variable and provide more names in the data list, if you would like to try more. The amount of the win is calculated so that the

expected payoff will be slightly less than the expected cost. If desired, it is also easy to add more complex winning rules with different payoffs. Variables are also set to control the length of the free-wheeling and slow-down periods.

Slot Machine Simulation Program

```
10: S = 0: F = 10: G = 90: H = 200
```

Set initial variables — S is amount of winnings, F is amount of PRINT WAIT during first part of spin, G is longest print WAIT in spin, and H is PRINT WAIT for information massages

```
20: T = 3: Z = INT (((T^3) - 1)/T): C = 3*T: B = 6*T
```

T is number of objects, Z is earning from a win, C is length of fast spin, B is maximum length of slow spin portion

```
30: DIM R(3), D(3), P$(T)*8
```

Allocate space for arrays — R ( ) is how many positions each wheel will turn, D ( ) is number of current objects, p$ ( ) is names of objects

```
40: RANDOM
```

Randomize seed for RND ( )

```
50: FOR I = 1 TO 3
60: D(I) = RND(T)
70: NEXT I
```

Initialize the object for each "window" to a random value

```
80: FOR I = 1 TO T
90: READ P$(I)
100: NEXT I
110: DATA " ORANGE ", " LEMON ", " CHERRY "
```

Read in the names of the objects

```
120: WAIT H
```

Spaces are added to make each 8 characters

```
130: PRINT "1$ SLOT MACHINE"
140: WAIT
150: PRINT "PRESS ENTER TO START"
```

Set WAIT for start message
Announce start
Reset WAIT to wait for Enter
Wait for Enter to begin

```
200: REM***START OF TURN***
210: M = 0
```

Initialize M

```
220: FOR I = 1 TO 3
230: R(I) = B + RND (C)
240: IF (R(I) > M) THEN LET M = R(I)
250: NEXT I

260: WAIT F
270: E = (G–F)/(M–C)

280: FOR A = 1 TO M
290: IF (A> C) THEN WAIT F + ((A–C)* E)
300: FOR I = 1 TO 3
310: IF (R(I)>0) THEN LET D(I) = D(I) + 1
320: IF (D(I)>T) THEN LET D(I) = 1
330: R(I) = R(I)–1
340: NEXT I

350: PRINT P$(D(1); P$(D(2)); P$(D(3)
360: NEXT A

370: WAIT H
380: PRINT P$(D(1)); P$(D(2)); P$(D(3))
390: W = –1
400: IF (D(1) = D(2)) AND (D(2) = D(3)) THEN LET W = Z
410: IF W<0 THEN PRINT "YOU_LOSE"
420: IF W>0 THEN PRINT "YOU_WIN"; Z; "DOLLARS"
430: S = S + W

440: ON (2↑ SGN (S)) GOTO 450, 470, 490
450: PRINT 'SO__FAR__YOU__HAVE__LOST $"; ABS (S)
460: GOTO 500
```

Determine random stopping time for each window
Set M to longest stopping time

Reset WAIT to fastest interval
Compute the amount to slow the interval
during each turn of the slow down phase
Loop from 1 to longest stopping time
If in slow-down phase, then slow down rotation by one increment
Loop through each window
If window it still turning, then advance to next object
Cycle back to first object if over top
Reduce count of remaining turns

Show current objects

Reset WAIT to longest interval
Redisplay ending position
Set winnings this turn to expected loss of $1
If all objects are the same, then set to win amount
If loss, then say so
If win, then say so
Add this turn to total winnings

Jump to message depending on sign of winnings
Message for S<0
Go to common end

```
470: PRINT "YOU ARE BREAKING EVEN"
480: GOTO 500
490: PRINT "SO FAR YOU HAVE WON $"; S

500: INPUT "ANOTHER TRY?"; Q$
510: IF ( LEFT$(Q$,1) = "Y") THEN GOTO 200
520: END
```

Message for S = 0
Go to common end
Message for S>0

Ask about another turn
Check first character of answer

Federal Tax Estimator

This program provides an estimate of United States Federal tax liability. It requests filing status, number of exemptions, salary income, self-employment income, other income, and amount of itemized deductions. Based on this information, it computes the taxable income. Next, it reads in the tax table appropriate to the filing status and determines the amount of Federal Income Tax. If there is self-employment income, the Social Security tax is computed. Any additional tax obligations and credits are solicited and a total tax liability is calculated.

The method of computing Federal Income Tax is based on "Tax Rate Schedules X, Y and Z". These schedules break down the income for each filing status into tax brackets. For each tax bracket, there is a tax computation rule of the form: "If income is over B(I), but not over B(I + 1), then the tax is M(I) plus P (I) percent of the excess over B(I)". The program scans the baseline values, B(I), for the largest value which is not greater than the income. It determines the appropiate bracket and applies the rule to compute tax liability.

NOTE: The values included in this program are based on those in the 1982 Declaration of Estimated Tax for Individuals (Form 1040-ES). This program is presented here to illustrate the capabilities of the **PC-3** Pocket Computer and is not intended to be an authoritative basis for any individuals actual tax liability. Individual circumstances and changing laws provide too many circumstances for a simple program such as this one to be complete. Consult a tax professional if you have questions about your own tax liability.

United States Federal Tax Estimator Program

```
 10: DIM B(15), M(15), P(15)

 20: USING " ######,"

 30: WAIT 128
 40: PRINT "FILING STATUS:"
 50: PRINT "1 = SINGLE"
 60: PRINT "2 = MARRIED FILING SEPAR."
 70: PRINT "3 = MARRIED FILING JOINT"
 80: PRINT "4 = HEAD OF HOUSEHOLD"
 90: WAIT
100: INPUT "STATUS? "; F
110: IF ((F<1) OR (F>4)) THEN GOTO 40
120: INPUT "NO. OF EXEMPTIONS? ";E
130: INPUT "EST. SALARY INCOME? "; I
140: INPUT "EST. S.E. INCOME? "; S
150: INPUT "EST. OTHER INCOME? "; 0
160: I = I + S + 0
170: PRINT "TOTAL  INCOME= "; I

180: INPUT "WILL  YOU  ITEMIZE? "; Q$
190: IF (LEFT$ (Q$,1) <> "Y") THEN GOTO 270
200: INPUT "EST. TOTAL DEDUCT.? "; D
210: IF  ((F = 1) OR (F = 4)) THEN LET D = D−2300
220: IF  (F = 2) THEN LET D = D−3400
230: IF  (F = 3) THEN LET D = D−1700
240: IF  (D<0) THEN LET D = 0
250: I = I−D
260: PRINT "INC. LESS DED. ="; I
```

Allocate arrays — B ( ) is baseline for tax bracket, M ( ) is minimum tax in bracket, P ( ) is percent within bracket
Set format for all displays

Set WAIT for Status options display
Display options for filing status

Reset WAIT so that Enter is required after each display
Get filing status
Check if valid
Get number of exemptions
Get salary income
Get self-employment income
Get other income
Total the incomes
And display

Ask about itemizing deductions
If not then skip itemizing section
Get total itemized deductions
Subtract standard deduction according to filing status

Minimum deduction is zero
Reduce income by excess over standard
And display

```
270: I = I-(E * 1000)
280: IF (I<0) THEN LET I = 0
290: PRINT "INC. LESS EXMP. ="; I

300: W = 750 + (50 * F)
310: RESTORE W
320: READ L
330: FOR X = 1 TO L
340: READ B(X), M(X), P(X)

350: IF  (I>B(X)) THEN LET J = X

360: NEXT X

370: T = M(J) + (P(J) * (I-B(J)))
380: PRINT "FIT = "; T
390: INPUT "AMT. OF ADD. TAX? "; A

400: INPUT "AMT. OF TAX CREDITS? ";C

410: Z = S*.0935
420: IF Z>3029.40 THEN LET Z = 3029.40
430: IF Z>0 THEN PRINT "S.S. TAX = ";Z

440: T = T + A-C + Z
450: PRINT "EST. TOTAL TAX = "; T
460: END
```

Note: Tax tables have standard deduction built-in
Compute income less exemptions
Minimum income is zero
Display

Compute line number of appropriate tax table for filing status
And restore so READ will get right table
Read number of lines in table
Loop to read in table
Read baseline, minimum tax, and percent for each tax bracket

Save pointer to highest applicable bracket

Compute FIT (see text)
Display FIT
Get any miscellaneous tax obligations

Get any miscellaneous tax credits

Compute social security tax on self-employment income
Limit S.S. tax to maximum
Display if S.S. tax is greater than zero

Total tax
And display

```
800: REM TABLE FOR SINGLE TAXPAYERS
801: DATA 14
802: DATA 0,0,0
803: DATA 2300,0,.12
804: DATA 3400,132,.14
805: DATA 4400,272,.16
806: DATA 6500,608,.17
807: DATA 8500,948,.19
808: DATA 10800,1385,.22
809: DATA 12900,1847,.23
810: DATA 15000,2330,.27
811: DATA 18200,3194,.31
812: DATA 23500,4837,.35
813: DATA 28800,6692,.40
814: DATA 34410,8812,.44
815: DATA 41500,12068,.50

850: REM TABLE FOR MARRIED FILING SEPAR.
851: DATA 13
852: DATA 0,0,0
853: DATA 1700,0,.12
854: DATA 2750,126,.14
855: DATA 3800,273,.16
856: DATA 5950,617,.19
857: DATA 8000,1006,.22
858: DATA 10100,1468,.25
859: DATA 12300,2018,.29
860: DATA 14950,2787,.33
861: DATA 17600,3661,.39
862: DATA 22900,5728,.44
```

```
863: DATA 30000,8852,.49
864: DATA 42800,15124,.50

900: REM TABLE FOR MARRIED FILING JOINT
901: DATA 13
902: DATA 0,0,0
903: DATA 3400,0,.12
904: DATA 5500,252,.14
905: DATA 7600,546,.16
906: DATA 11900,1234,.19
907: DATA 16000,2013,.22
908: DATA 20200,2937,.25
909: DATA 24600,4037,.29
910: DATA 29900,5574,.33
911: DATA 35200,7323,.39
912: DATA 45800,11457,.44
913: DATA 60000,17705,.49
914: DATA 85600,30249,.50

950: REM TABLE FOR HEAD OF HOUSEHOLD
951: DATA 14
952: DATA 0,0,0
953: DATA 2300,0,.12
954: DATA 4400,252,.14
955: DATA 6500,546,.16
956: DATA 8700,898,.20
957: DATA 11800,1518,.22
958: DATA 15000,2222,.23
959: DATA 18200,2958,.28
960: DATA 23500,4442,.32
```

```
961: DATA 28800, 6138, .38
962: DATA 34100, 8152, .41
963: DATA 44700, 12498, .49
964: DATA 60600, 20289, . 50
```

Relationship of Two Variables

The **PC-3** Computer an excellent tool for making many small statistical tests. As an example of this capability, this program calculates the basic tests which are often used to compare a series of paired observations. The program solicits the observations which are entered in pairs. When there are an independent and a dependent variable, the dependent variable is X and the independent is Y. If the variables are independent, then simply assign one to X and one to Y.

The program loops through the observations and computes several quantities which are used to calculate the desired statistics. These quantities are the Sum of X, the Sum of X^2, the Sum of Y, the Sum of Y^2, and the Sum of X*Y. The mean of X is then computed with the formula:

$$\text{Mean}_x = \frac{\text{Sum of X}}{N}$$

Where N is the number of observation pairs. The standard deviation of X is then calculated with these formulas:

$$\text{Sum of Squares}_x = \text{Sum of } X^2 - \frac{(\text{Sum of X})^2}{N}$$

$$\text{Standard Deviation}_x = \sqrt{\frac{\text{Sum of Squares}_x}{(N-1)}}$$

The mean and standard deviation of Y are computed with the same formulas. These quantities are then used to calculate the correlation coefficient between the two variables using the formulas:

$$\text{Sum of Products}_{x,y} = \text{Sum of X} * \text{Y} - \frac{(\text{Sum of X}) * (\text{Sum of Y})}{\text{N}}$$

$$\text{Correlation}_{x,y} = \frac{\text{Sum of Products}_{(x,y)}}{\sqrt{(\text{Sum of Squares}_x) * (\text{Sum of Squares}_y)}}$$

Finally the program computes the coefficients for the linear regression equation using the formulas:

$$b_{x \cdot y} = \frac{\text{Sum of Products}_{x,y}}{\text{Sum of Squares}_y} \qquad a = \text{Mean}_y - (b_{x \cdot y} * \text{Mean}_x)$$

The coefficients are then shown in the regression equation:

$$Y = a + b_{x \cdot y} X$$

Relationship of Two Variables Program

```
10: A = 0, B = 0, C = 0, D = 0, V = 0
20: INPUT "NUMBER OF OBSERV.? " ; N
30: DIM X(N), Y(N)
40: WAIT 48
50: PAUSE "ENTER "; N; " PAIRS OF OBS."
```

Initialize variables to accumulate sums
Get number of observations
Allocate arrays to hold observations
Set 3/4 second wait for prompts during data entry
Prompt start of data entry

```
 60: FOR I = 1 TO N
 70: PRINT "PAIR  "; I
 80: INPUT "X?  "; X(I)
 90: INPUT "Y?  "; Y(I)
100: NEXT I

110: WAIT 128

120: INPUT "DISPLAY OR PRINTER?  "; W$
130: IF ( LEFT$ (W$, 1) = "P") THEN PRINT = LPRINT

140: INPUT "LIST OF DATA?  "; W$
150: IF ( LEFT$ (W$, 1) = "Y") THEN LET V = 1

160: FOR I = 1 TO N
170: IF (V = 1) THEN PRINT X(I), Y(I)
180: A = A + X(I)
190: B = B + X(I) ^ 2
200: C = C + Y(I)
210: D = D + (Y(I) ^ 2)
220: E=E+(X(I)*Y(I))
230: NEXT I

240: WAIT

250: F = A/N
260: PRINT "MEAN OF X = "; F
270: G = C/N
280: PRINT "MEAN OF Y = "; G
```

Loop for number of observations
Prompt with number of pair
Prompt and input X
Prompt and input Y

Reset WAIT time for data listing

Ask if output is to display or printer
If printer, then switch

Ask if listing of data is desired
If so, set flag; default V=0 set in line 10

Loop through data
If flag is set, then print observation pair
Accumulate the sum of X
Accumulate the sum of the squares of X
Accumulate the sum of Y
Accumulate the sum of the squares of Y
Accumulate the sum of the products of the pairs

Reset WAIT so that Enter is required

Compute mean of X
And display
Compute mean of Y
And display

```
290: J = B-((A ^ 2)/N)
300: K = SQR (J/(N - 1))
310: PRINT "STD. DEV. X = "; K

320: L = D-((C ^ 2)/N)
330: M = SQR (L/(N - 1))
340: PRINT "STD. DEV. Y = "; M

350: O = E - ((A * C)/N)
360: R = O/ SQR (J * L)
370: PRINT "CORREL. COEF. = "; R

380: P = O/J
390: Q = G - (P * F)
400: WAIT 128
410: PRINT "REGRESSION EQUATION IS"
420: WAIT
430: PRINT "Y = "; Q; " + "; P; "X"
440: PRINT = PRINT
450: END
```

Compute the sum of squared deviates of X
Compute the standard deviation of X
And display

Compute the sum of squared deviates of Y
Compute the standard deviation of Y
And display

Compute the sum of products of the deviates
Compute the correlation
And display

Compute the regression coefficient
Compute the Y-intercept
Reset WAIT for leadin display
Print leadin display for equation
Reset WAIT for leadin display
Print the regression equation
Reset output to display

Minefield Game

This program provides a simple minefield game. The game is played on a 10 x 10 set of squares like a checkerboard. The top of the board is north, the bottom south, the left west, and the right east. The columns are numbered horizontally from left to right, from 1 to 10. The rows are numbered vertically from bottom to top, also from 1 to 10. You begin the game in the southwest corner, square (1, 1). The object is to move to the northeast corner, square (10, 10). You make moves by entering a number from 1 to 9 to indicate the direction you want to take. The directions are indicated by the position of the key on the numeric pad, i.e.

Northwest	North	Northeast
7	8	9
West		East
4	5	6
Southwest	South	Southeast
1	2	3

At random squares on the board there are "mines". If you enter a square with a mine, you lose. To help you avoid the mines, the **PC-3** Computer will beep as it enters a square. It will make one beep for each adjacent square with a mine, but it won't tell you the direction. The program checks to insure that no mines are laid so close to the corners that it is difficult or impossible to start or finish. You determine the number of mines by responding with a number in response to the question "DIFFICULTY?".

Notice the use of subroutines in this program to produce a short, simple main program with special functions isolated in separate subroutines.

Minefield Game Program

SEE "PROGRAM LIBRARY" FOR ADAPTATIONS (Pg. 19)

```
10: S = 10
20: DIM F (S + 1, S + 1)

30: WAIT 128
40: INPUT "DIFFICULTY LEVEL? "; B
50: PRINT "ONE MINUTE FOR SETUP"
```

Set size of field to 10 by 10
Allocate array 1 larger in both directions — Extra size facilitates loop at line 220
Set WAIT for 2 seconds
Ask for number of mines
Display warning about setup time

```
 60: FOR I = 1 TO S
 70: FOR J = 1 TO S
 80: F (S, S) = 0
 90: NEXT J
100: NEXT I

110: FOR I = 1 TO B
120: X = RND (S)
130: Y = RND (S)
140: IF ((X + Y) < 5) THEN GOTO 120
150: IF (((1 + S – X) + (1 + S – Y)) < 5) THEN GOTO 120
160: F (X, Y) = 1
170: NEXT I
180: F (S, S) = 9

190: X = 1 : Y = 1

200: PRINT "YOU ARE AT (";X;", ";Y;")"
210: GOSUB 400
220: BEEP C

230: INPUT "WHICH WAY NOW? "; A$
240: D = VAL (A$)
250: IF ((D < 1) OR (D > 9)) THEN PRINT
     "1 TO 9 ONLY, PLEASE" : GOTO 280

260: IF (D > 6) THEN GOSUB 500
270: IF (D < 4) THEN GOSUB 550
280: IF ((D = 3) OR ((D= 6) OR (D = 9)) THEN GOSUB 600
```

Loop through entire array and set each entry to zero – Needed to clear out mines from prior games

Loop for number of mines
Get random X coordinate
Get random Y coordinate
Check it too close to starting corner
Check if too close to ending corner
Mark mine

Mark goal

Set start to (1, 1)

Start of turn – show location
Count the number of nearby mines
Beep to indicate number of nearby mines

Ask for direction from numeric pad
Convert keystroke to number
If not from numeric pad, then ask again

If from top row then GOSUB to north subroutine
If from bottom row, then GOSUB to south subroutine
If from right side, then GOSUB to east subroutine

```
290: IF ((D = 1) OR (D = 4) OR (D = 7)) THEN GOSUB 650

300: IF (F(X, Y) = 1) THEN GOTO 700
310: IF (F(X, Y) = 9) THEN GOTO 750
320: GOTO 200

400: C = 0
410: FOR I = X – 1 TO X + 1
420: FOR J = Y – 1 TO Y + 1
430: IF (F(I, J) = 1) THEN LET C = C + 1
440: NEXT J
450: NEXT I
460: RETURN

500: IF (Y = S) THEN PRINT
     "YOU ARE AT NORTH EDGE" : GOTO 520
510: Y = Y + 1
520: RETURN

550: IF (Y = 1) THEN PRINT
     "YOU ARE AT SOUTH EDGE" : GOTO 570
560: Y = Y – 1
570: RETURN

600: IF (X = S) THEN PRINT
610: "YOU ARE AT EAST EDGE" : GOTO 620
610: X = X + 1
620: RETURN
```

If from left side, then GOSUB to west subroutine

If new square contains a bomb, then GOTO losing message
If new square is the goal, then GOTO winning message
Otherwise loop

Mine counting subroutine
Set counter for mines
Loop through neighboring squares
If there is a mine, increment count by one

North subroutine
If at north edge, don't do anything
Move one square north
And return

South subroutine If at south edge, don't do anything

Move one square south
And return

If at east edge, don't do anything

Move one square east
And return

```
650: IF (X = I) THEN PRINT
     "YOU ARE AT WEST EDGE" : GOTO 670
660: X = X – 1
670: RETURN

700: PRINT "BOOM!!!!!!!  YOU LOSE"
710: GOTO 800

750: PRINT "CONGRATULATIONS, YOU WIN"

800: INPUT "ANOTHER GAME?  "; A$
810: IF ( LEFT$ (A$, 1) = "Y") THEN GOTO 40
820: END
```

If at west edge, don't do anything

Move one square west
And return

Display losing message
And go to common end

Display winning message

Ask about another game
Go again if first letter is "Y"
Otherwise end

CHAPTER 1Ø TROUBLESHOOTING

This chapter provides you with some hints on what to do when your **Radio Shack PC-3** Pocket Computer 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 RUN, PRO, or RSV. 2. Push the (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. Switch from the PROgram into the RUN mode 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, push the RESET without any key. (With this operation, the program, data, and all reserved contents are cleared.)

BASIC Debugging

A newly entered BASIC program may not always work for 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-3** Computer 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 11 MAINTENANCE OF THE PC-3 POCKET COMPUTER

To insure trouble-free operation of your **Radio Shack PC-3** Pocket Computers, we recommend the following:

* Always handle the Pocket Computer carefully, as the liquid crystal display is made of glass.
* Keep the Computer 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 Computer.
* Use only a soft, dry cloth to clean the Computer. Do not use solvents, water, or wet cloths.
* To avoid battery leakage, remove the batteries when the Computer will not be in use for an extended period of time.
* If service is required, the Computer should only be returned to an authorized **Radio Shack** Service Center.
* If the Computer is subjected to strong static electricity or external noise, it may "hang up" (all keys become inoperative). If this occurs, press the ALL RESET button while holding down any key. (See Troubleshooting.)
* Keep this manual for further reference.

(NOTE: For maintenance of the **PC-3** Printer/Cassette Interface, please see Chapter 7.)

APPENDIX A ERROR MESSAGES

The **PC-3** Pocket Computer has nine different error messages.

Error Number	Meaning
1	Syntax Error. 3 ✳ / 2
2	Calculation error. Either you have tried to use a number which exceeds the capacity of the **PC-3** Computer: 9.9 ⁞E 99 ✳ 1Ø or you have tried to divide by zero: 5 / Ø
3	DIM Error/Range over Error. Array variable already exists, array specified without first DIMensioning it or array subscript exceeds size of array specified in DIM statement. DIM B(256) Specified numeric value is outside permitted range, etc. 1Ø: FOR I = 1 TO 35ØØØ
4	Line number error. You have specified an invalid line number.
5	Nesting Error. Buffer space exceeded or FOR statement nested too deeply, etc.

6 Memory overflow error. You have exceeded the momory capacity of the **PC-3** Pocket Computer with some combination of programs and data:

```
10: DIM B (100, 100)
```

7 Print format error.

Data cannot be displayed in accordance with the format specified by USING command.

```
10: USING "####"
20: A = 123 * 20
30: PRINT A
```

8 I/O device error. An error has occurred in sending information between the **PC-3** Computer and another device, such as the Printer or tape on the **PC-3** Printer/Cassette Interface, check the low battery indicator on the **PC-3** Printer/Cassette Interface. Check all the connections.

9 Other. Some other error has occurred which is not one of the above errors. Often this is due to an illegal value:

```
CHR$ (1)
```

or a misuse of preallocated variables:

```
10: A = 5 : PRINT 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 corner shows the character. If no character is shown, then it is an illegal character on the **PC-3** Pocket Computer. For example, the character 'A' is a decimal 65 or a hex 41 or a binary Ø1ØØØØØ1.

First 4 bits

Second 4 Bits	Hex	0	1	2	3	4	5	6	7
	Binary	0000	0001	0010	0011	0100	0101	0110	0111
	0 0000	0 NUL	16	32 SP	48 0	64 @	80 P	96 `	112
	1 0001	1	17	33 !	49 1	65 A	81 Q	97	113
	2 0010	2	18	34 "	50 2	66 B	82 R	98	114
	3 0011	3	19	35 #	51 3	67 C	83 S	99	115
	4 0100	4	20	36 $	52 4	68 D	84 T	100	116
	5 0101	5	21	37 %	53 5	69 E	85 U	101	117
	6 0110	6	22	38 &	54 6	70 F	86 V	102	118
	7 0111	7	23	39 '	55 7	71 G	87 W	103	119
	8 1000	8	24	40 (	56 8	72 H	88 X	104	120

9 1001	9	25	41)	57 9	73 I	89 Y	105	121
A 1010	10	26	42 *	58 :	74 J	90 Z	106	122
B 1011	11	27	43 +	59 ;	75 K	91 √	107	123
C 1100	12	28	44 ,	60 <	76 L	92 ¥	108	124
D 1101	13	29	45 –	61 =	77 M	93 π	109	125
E 1110	14	30	46 .	62 >	78 N	94 ^	110	126
F 1111	15	31	47 /	63 ?	79 O	95 _	111	127

PC-3 does not recognize codes in shaded area. If you enter the code number in the shaded area, an error will result.

APPENDIX C FORMATTING OUTPUT

It is sometimes important or useful to control the format as well as the content of output. The **PC-3** Pocket Computer 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 diffrerent formats are specified with an "output mask". This mask may be a string constant or a string variable:

```
10: USING "####"
20: M$ = "&&&&&&"
30: USING M$
```

When the USING verb is used with no mask, all special formatting is cancelled.

```
40: USING
```

A USING verb may also be used within a PRINT statement:

```
50: 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-3** Pocket Computer maintains a maximum of 10 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 an 'I' character to show the size of the field.

Statement	Display
10: USING "####"	(Set the **PC-3** Computer to the RUN position, type RUN, and press (ENTER) .)
20: PRINT 25	25
30: PRINT −350	−350
40: PRINT 1000	ERROR 7 IN 40

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
10: USING "####.##"	
20: PRINT 25	25.00
30: PRINT −350.5	−350.50
40: 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
10: USING "###.##^"	
20: PRINT 2	2.00↑E00
30: PRINT −365.278	−3.65↑E02

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
10: USING "&&&&&&"	
20: PRINT "ABC"	ABC
30: PRINT "ABCDEFGHI"	ABCDE

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
10: PRINT USING "###.##&&"; 25; "CR"	**25.00CR**
20: 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 **Radio Shack PC-3** Pocket Computer 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:

```
100/5+45
```

as either a calculation or as a part of a program, the **PC-3** Computer does not know if you mean:

$$\frac{100}{5+45} = 2 \qquad \text{or} \qquad \frac{100}{5} + 45 = 65$$

Since the **PC-3** Computer 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 below), 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 **Ratio Shack PC-3** Computer 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 (+, –)
10. 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. 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) / 10^LOG 100
```

The **PC-3** Computer 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) / 10^LOG 100
```

Then it would do subtraction:

```
( (6)*6+2) / 10^LOG 100
```

or:

```
(6*6+2) / 10^LOG 100
```

In the next set of parentheses, it would do the multiplication first:

```
(36+2) / 10^LOG 100
```

And then the addition:

```
(38) / 10^LOG 100
```

or:

```
38 / 10^LOG 100
```

Now that the parentheses are cleared, the LOG function has the highest priority, so it is done next:

```
38 / 10^2
```

The exponentiation is done next:

```
38 / 100
```

And last of all, the division is performed:

```
0.38
```

This is the value of the expression.

APPENDIX E FEATURE COMPARISON OF THE PC-1, PC-2, AND PC-3

The three **Radio Shack** Pocket Computers, the **PC-1**, the **PC-2**, and the **PC-3**, 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	Comments
AREAD	P	B	P	See Note 1
ARUN		P		
BEEP	P	B	B	**PC-2** has tone and duration
CHAIN	P	P	P	
CLEAR	B	B	B	
CLOAD	M	M	M	
CLOAD?	M	M	M	
CLS		B		

	PC-1	PC-2	PC-3	Comments
COLOR		P		
CONT	M	M	M	
CSAVE	M	B	B	
CSIZE		B		
CURSOR		B		
DEGREE	B	B	B	
DATA		P	P	
DEBUG	M			
DIM		B	B	
END	P	P	P	
FOR ... TO ... STEP	P	P	P	
GOSUB	P	P	P	
GOTO	P	B	B	
GCURSOR		B		
GPRINT		B		
GRAD	B	B	B	
GRAPH		B		
IF ... THEN	P	P	P	
INPUT	P	P	P	
INPUT #	B	B	B	
LET	P	P	P	

	PC-1	PC-2	PC-3	Comments
LF		B		
LINE		B		
LIST	M	M	M	
LLIST		B	M	**PC-1** can emulate with LIST
LOCK		B		
LPRINT		B	P	See Note 2
MERGE	M	M	M	
NEW	M	M	M	
NEXT	P	P	P	
ON ... ERROR		P		
ON ... GOSUB		P	P	
ON ... GOTO		P	P	
PAUSE	P	B	P	
PASS			M	
PRINT	P	B	P	See Note 2
PRINT #	B	B	B	
RADIAN	B	B	B	
RANDOM		B	B	
READ		P	P	
REM	P	P	P	
RESTORE		P	P	

	PC-1	PC-2	PC-3	Comments
RETURN	P	P	P	
RLINE		B		
RMTOFF		B		
RMTON		B		
ROTATE		B		
RUN	M	M	M	
SORGN		B		
STOP	P	P	P	
TAB		B		
TEST		B		
TEXT		B		
TROFF		B	B	
TRON		B	B	
UNLOCK		B		
USING	P	B	B	See Note 3
WAIT		B	B	

Note 1: There are some minor differences between the **PC-3** 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-3.**

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 executed, this program displays the following:

- PC-1

```
                    -123.45
        -123          -123
```

- PC-3

```
                    -123.45
    -123.45           -123
```

Note 4: As compared with the **PC-1**, the **PC-3** is faster in processing speed for calculations. Therefore, when game programs for the **PC-1** are used with the **PC-3**, adjust the game speed, etc.

Pseudovariables

In this and the following charts, the features are simply marked with a 'Y' when the machine has the feature.

	PC-1	PC-2	PC-3	Comments
INKEY$		Y	Y	
MEM	Y	Y	Y	
PI or π	Y	Y	Y	**PC-1** has only π
TIME		Y		

Numeric Functions

	PC-1	PC-2	PC-3	Comments
ABS	Y	Y	Y	
ACS	Y	Y	Y	
ASN	Y	Y	Y	
ATN	Y	Y	Y	
COS	Y	Y	Y	
DEG	Y	Y	Y	
DMS	Y	Y	Y	
EXP	Y	Y	Y	
INT	Y	Y	Y	
LOG	Y	Y	Y	
LN	Y	Y	Y	
NOT		Y	Y	
POINT		Y		
RND		Y	Y	
SGN	Y	Y	Y	
SIN	Y	Y	Y	
SQR or $\sqrt{\ }$	Y	Y	Y	PC-1 has only $\sqrt{\ }$
STATUS		Y		
TAN	Y	Y	Y	

String Functions

	PC-1	PC-2	PC-3	Comments
ASC		Y	Y	
CHR$		Y	Y	
LEFT$		Y	Y	
LEN		Y	Y	
MID$		Y	Y	
RIGHT$		Y	Y	
STR$		Y	Y	
VAL		Y	Y	

Operators

	PC-1	PC-2	PC-3	Comments
^	Y	Y	Y	See Note 4
*, /, +, –	Y	Y	Y	
>, >=, =, <>, <=, <	Y	Y	Y	
AND, OR,		Y	Y	
&		Y	Y	

Note 4: Raising a negative number to a power with the ^ operator can result in incorrect signs. See Chapter 4.

Precautions

Programs for the PC-1, when loaded from its tape, can be used with the PC-3. When entering the PC-1 programs into this unit from PC-3 keyboard, however, the following precautions should be observed:

For example, the following are keyed-in for program entry:

10 IF N = LPRINT A (ENTER)

With the PC-1, this results in a command for "If N=L, display A" (IF N=L PRINT A). With the PC-3, however, it becomes a command for "If N=, print A" (IF N=LPRINT A), causing a syntax error (ERROR 1) to occur when executed. This is because the PC-3 has an LPRINT command unavailable from the PC-1. Therefore, an IF statement should be keyed-in as:

10 IF N=L THEN PRINT A

Thus, a character string for "variable and command" with the PC-1 may be regarded as "a command".

APPENDIX F SPECIFICATIONS

Model:	**PC-3** Pocket Computer	
Processor:	8-bit CMOS CPU	
Programming Language:	BASIC	
Memory Capacity:	System ROM:	24 K Bytes
	RAM	
	System	About 500 Bytes
	User	
	Fixed Memory Area (A ~ Z, A$ ~ Z$)	208 Bytes
	Reserve Area	48 Bytes
	Program/Data Area	1438 Bytes
Stack:	Subroutine: 10 stacks	
	FOR–NEXT: 5 stacks	
	Function: 16 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:	24-character liquid crystal display with 5 x 7 dot characters
keys:	52 keys: Alphabetic, numeric, special symbols, and functions; Numeric pad; User-defined keys.
Power Supply:	6.ØV DC Lithium batteries Type: CR-2032
Power Consumption:	6.ØV DC @ 0.03W Batteries are sufficient for approximately 300 hours usage without external power supply.
Operating Temperature:	0°C ~ 40°C (32°F ~ 104°F)
Dimensions:	135(W) x 70(D) x 9.5(H) mm 5-5/16"(W) x 2-3/4"(D) x 3/8"(H)
Weight:	Approximately 115g (0.25 lbs.) (with batteries)
Accessories:	Wallet, two lithium batteries (built-in), two keyboard templates, and owner's manual
Option:	Printer/Cassette Interface

INDEX

Program Examples

In the preceding pages, you have probably acquired some new information on a number of program commands. Like driving a car or playing tennis, things 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. The following pages contain a variety of suggestions for programs using the BASIC commands. We provide you with the necessary equations and also inclube flowcharts. The rest is up to you!

(Radio Shack and/or its franchises assume no responsibilities or obligations to any losses or damages that could arise through the use of the software programs employed in this owner's manual.)

CONTENTS

Showing the bytes used in each program itself

The number of bytes used in each program are shown at the end of each program listing. For instance, at the end of the TYPING PRACTICE program, 475 were used bytes. The way to find this out follows:

At RUN position

1) CLEAR (ENTER)
2) 1438 (−) 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;

$$f'(X) = \frac{f(X+h) - f(X)}{h}$$ (h is the minute interval)

Change E-8 in line 340 to change the value for 10^{-8}.

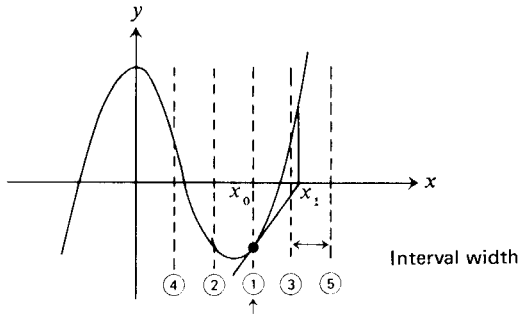
INSTRUCTIONS

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.

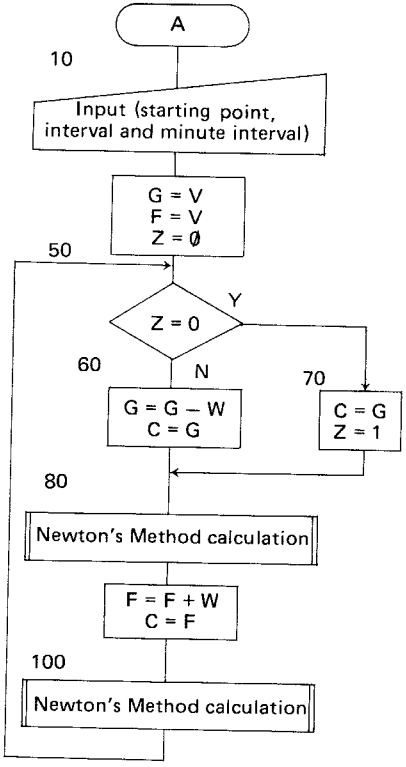
How to type in the example:

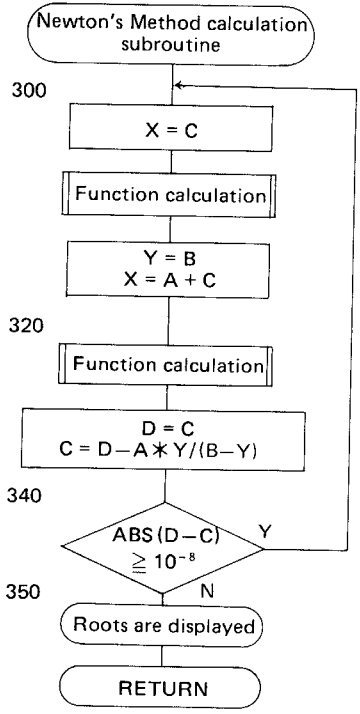
1. Go into PRO mode by operating the mode change key.
2. 500B = ((X−2) ∗ X−1) ∗ X+2 ENTER
   510 RETURN ENTER That is all that has to be done.

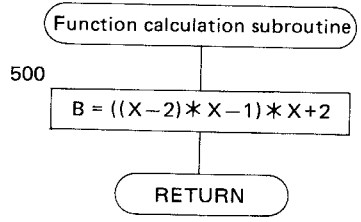
KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	STARTING POINT = _	Waiting for starting point input
2	Ø ENTER	MINUTE INTERVAL = _	Waiting for minute interval input
3	Ø.ØØØ1 ENTER	INTERVAL = _	Waiting for interval width input
4	Ø.5 ENTER	ANSWER = 2.	Display of roots
5	ENTER	ANSWER = 1.	By repeatedly pressing the ENTER key, the roots of the function are found.
6	ENTER	ANSWER = –1.	
7	ENTER	ANSWER = 1.	
8	ENTER	ANSWER = –1.	
9	ENTER	ANSWER = –1.	
10	ENTER	ANSWER = –1.	
11	ENTER	ANSWER = 2.	
	⋮	⋮	
	⋮	⋮	

FLOWCHART


A
10
Input (starting point, interval and minute interval)
G = V
F = V
Z = Ø
50
Z = 0
Y
60
N
70
G = G – W
C = G
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) ≧ 10⁻⁸
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 INTERV
   AL=";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:BEEP 3: PRINT "ANSWE
   R=",C
360:RETURN
500:B=((X-2)*X-1)*X+2
510:RETURN
```

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 $\Sigma x_i \cdot f_i$

Average $\bar{x} = \dfrac{\Sigma x_i \cdot f_i}{\Sigma f_i}$

Variance $\sigma^2 = \dfrac{\Sigma (x_i - \bar{x}) f_i}{\Sigma f_i - 1}$

(when there are no weights $f_i = 1$)

Standard deviation $\sigma = \sqrt{\sigma^2}$

Number of data entries (up to 50)

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 = 1 / NO WEIGHTS = 2 ?_	Waiting for the selection of weights/no weights
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

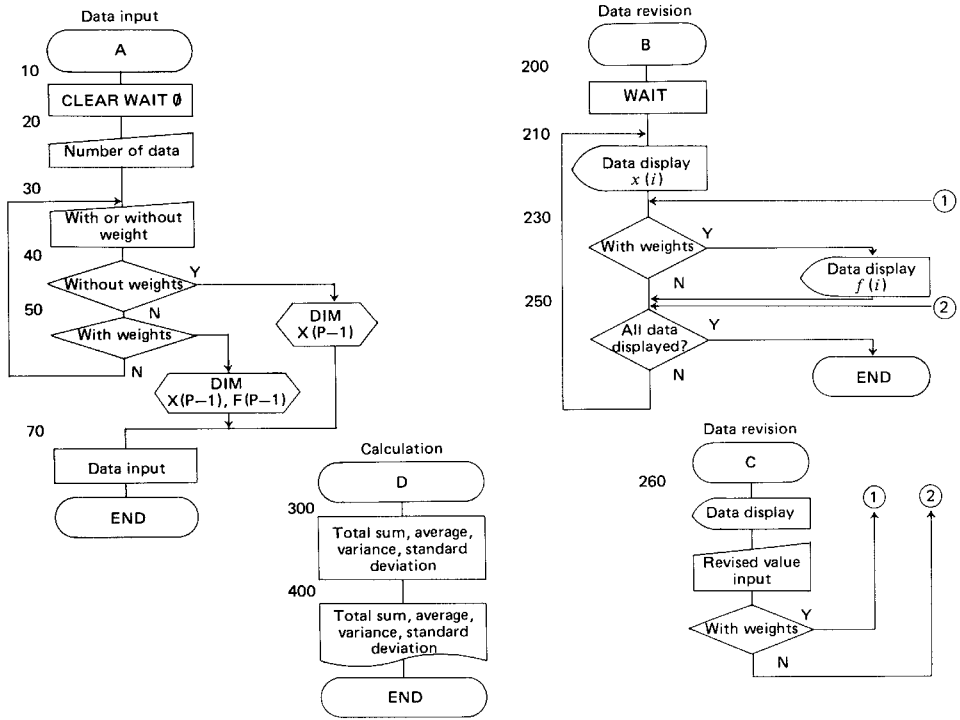
KEY OPERATION SEQUENCE

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)	⋮	

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF D	TOTAL SUM = 1072.5	Display of total sum
2	ENTER	MEAN VALUE = 14.3	Display of average
3	ENTER	VARIANCE = 1.432432432 E−02	Display of variance
4	ENTER	STD. DEV. =	Display of standard deviation
5	ENTER	1.196842693 E−01	
6	ENTER	>	Processing finished

FLOWCHART


Data input
A
10
CLEAR WAIT ∅
20
Number of data
30
With or without weight
40
Without weights
Y
N
50
With weights
N
DIM X(P−1)
DIM X(P−1), F(P−1)
70
Data input
END
Calculation
D
300
Total sum, average, variance, standard deviation
400
Total sum, average, variance, standard deviation
END
Data revision
B
200
WAIT
210
Data display x (i)
230
With weights
Y
N
Data display f (i)
250
All data displayed?
Y
N
END
Data revision
C
260
Data display
Revised value input
With weights
Y
N
1
2

PROGRAM LIST

```
 10:"A": CLEAR : WAIT 0
 20:INPUT "NO. OF DATA="
    ;P
 30:INPUT "WEIGHTS=1/NO
    WEIGHTS=2?";A
 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=";T
    : PRINT "MEAN VALUE=
    ";X
410:PRINT "VARIANCE=";Q:
    PRINT "STD. DEV.=":
    PRINT S: END

643
```

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

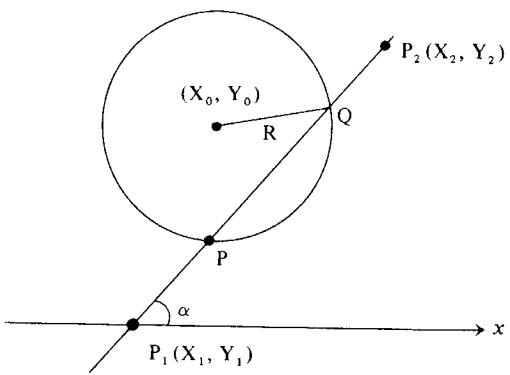
The points of intersection between circles and straight lines in the X–Y plane are found.

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.



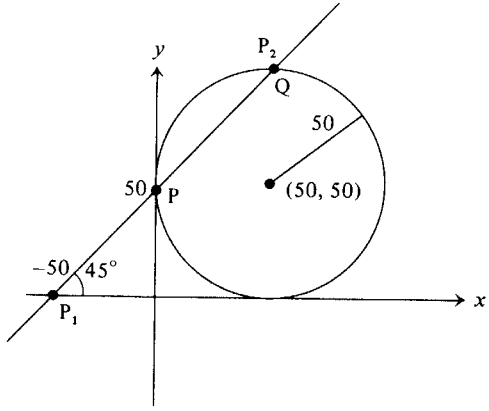
INSTRUCTIONS

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

$X_1 = -50$
$Y_1 = 0$
$X_2 = 50$
$Y_2 = 100$
$X_0 = 50$
$Y_0 = 50$
$R = 50$
$\alpha = 45°$

$X_P = 0$
$Y_P = 50$
$X_Q = 50$
$Y_Q = 100$



(Note) The coordinate values are accurate up to 5 decimal places.

KEY OPERATION SEQUENCE

(when 2 points on the line are known)

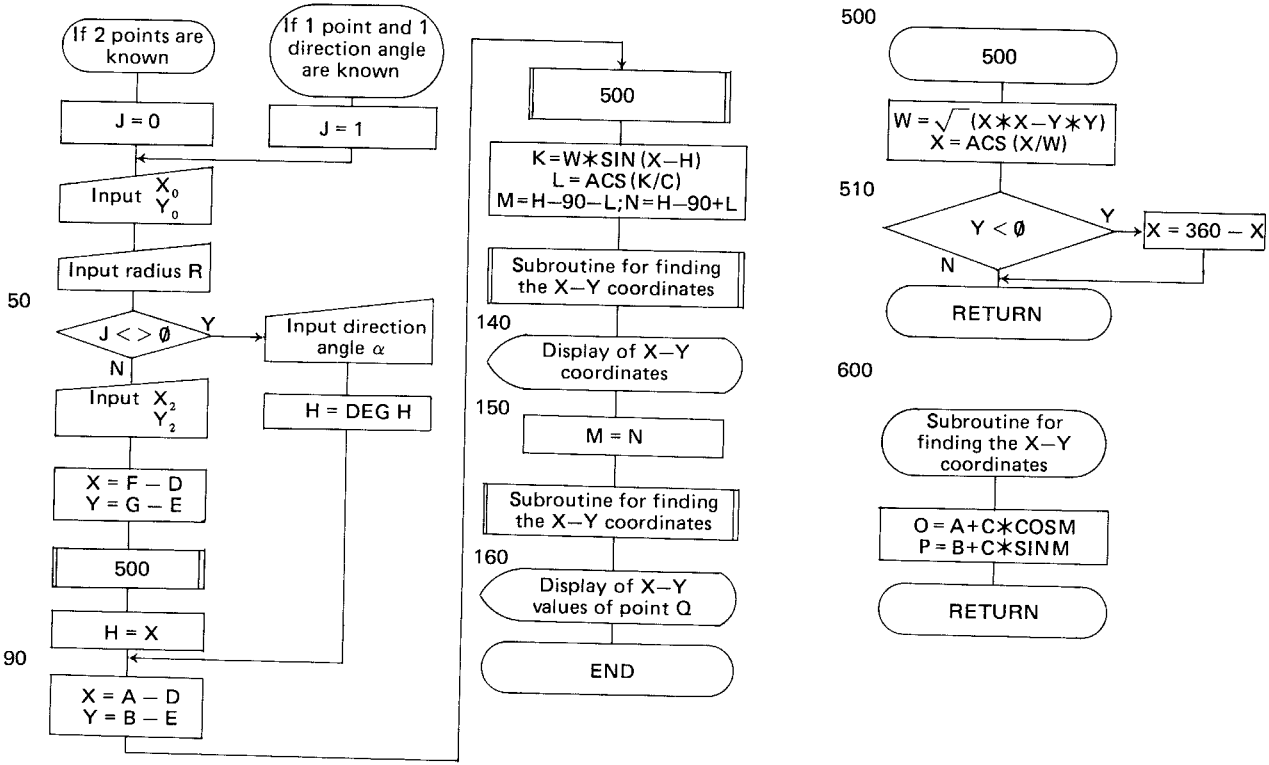
Step No.	Key Input	Display	Remarks
1	DEF A	XØ = _	
2	50 ENTER	YØ = _	
3	50 ENTER	R = _	
4	50 ENTER	X1 = _	
5	−50 ENTER	Y1 = _	
6	0 ENTER	X2 = _	
7	50 ENTER	Y2 = _	
8	100 ENTER	P: 0.0000 49.9999	(xp, yp)
9	ENTER	Q: 50.0000 100.0000	(x_Q, y_Q)

KEY OPERATION SEQUENCE

(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: 0.0000 49.9999	(x_p, y_p)
8	ENTER	Q: 50.0000 100.0000	(x_q, y_q)

FLOWCHART


If 2 points are known
If 1 point and 1 direction angle are known
J = 0
J = 1
Input X_0 Y_0
Input radius R
50
J <> Ø
Y
N
Input direction angle α
H = DEG H
Input X_2 Y_2
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:";O;P
150:M=N: GOSUB 600
160:PRINT "Q:";O;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

335
```

MEMORY CONTENTS

A	X_0
B	Y_0
C	R
D	X_1
E	Y_1
F	X_2
G	Y_2
H	$\sqrt{}$
I	
J	$\sqrt{}$
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.

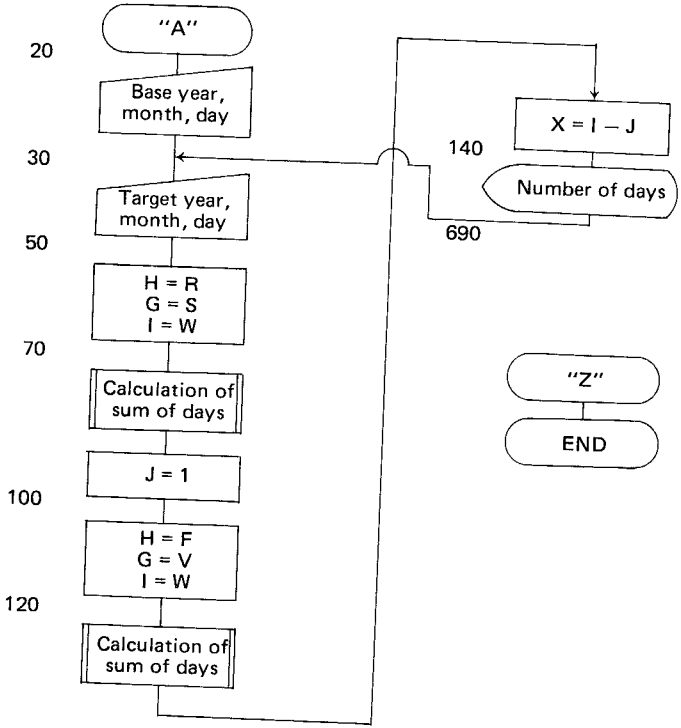
[Example]

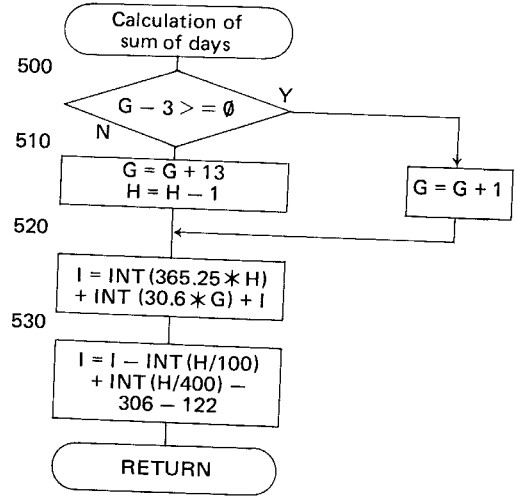
from 1976 year 10 month 5 day
to 1982 year 6 month 4 day : 2068 days
to 1985 year 1 month 1 day : 3010 days

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 = W
70
Calculation of sum of days
J = 1
100
H = F
G = V
I = W
120
Calculation of sum of days
X = I – J
140
Number of days
690
"Z"
END


Calculation of sum of days
500
G – 3 >= ∅
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 correct 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

After the buzzer sounds, 4 to 6 characters will be displayed. You are to type in the same characters within the allotted time.

If the 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.

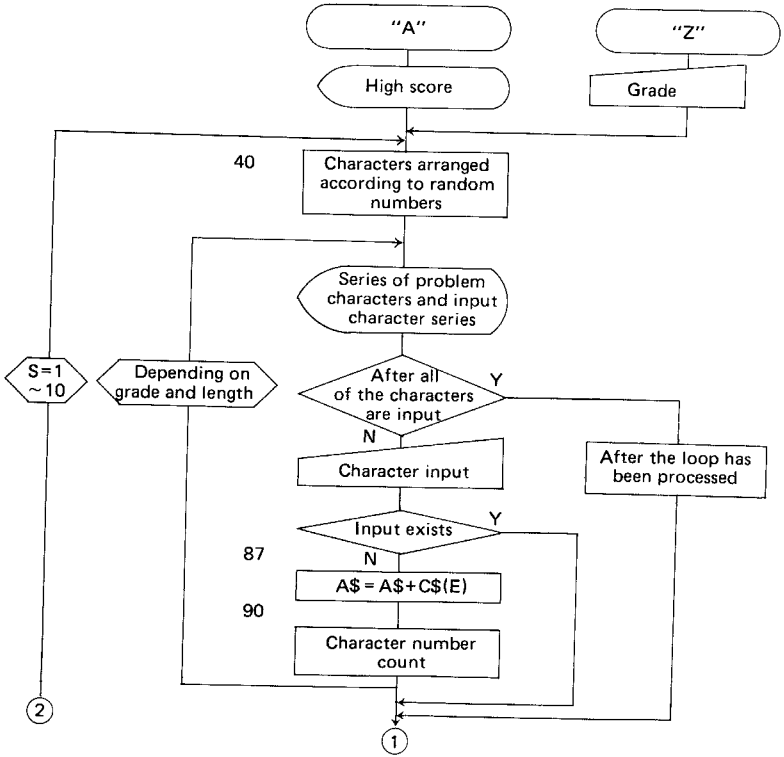
Point competition is done within the same grade category.

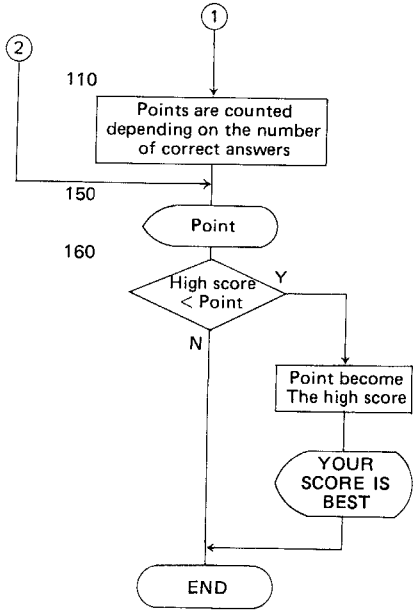
There are 10 problems, making the maximum score 100 points.

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
		YOUR SCORE IS BEST	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)	⋮	
	⋮	⋮	
	⋮	YOUR – SCORE = 60	
		>	

FLOWCHART


"A"
"Z"
High score
Grade
40
Characters arranged according to random numbers
Series of problem characters and input character series
S=1 ~10
Depending on grade and length
After all of the characters are input
Y
N
Character input
After the loop has been processed
Input exists
Y
87
N
A$ = A$+C$(E)
90
Character number count
2
1


1
2
110
Points are counted depending on the number of correct answers
150
Point
160
High score < Point
Y
N
Point become The high score
YOUR SCORE IS BEST
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$=""
70:BEEP 3:E=0: WAIT 30:
   USING "&&&&&&&&"
80:FOR W=1 TO B*10/L:
   PRINT Y$;"   ";A$:
   IF E=B LET W=B*20/L:
   GOTO 100
85:C$(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 : BEEP
   3: PAUSE "YOUR-SCORE
   =";P
160:IF P>X LET X=P: WAIT
   100: PRINT "YOUR SCO
   RE IS BEST"
170:END

475
```

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

$$H = H_0 + V_0 t + \frac{1}{2} a t^2$$

$$V = V_0 + at$$

$$V^2 = {V_0}^2 + 2aH$$

$$H_0 = 500, \quad V_0 = -50, \quad F_0 = 200$$

H : height
V : speed
a : gravitational acceleration
t : time
V_0 : initial speed

H_0 : initial height
V_0 : initial speed
F_0 : initial fuel
F : fuel burned

The initial height, initial fuel level, and the wait time are stored in line 30 as data. By changing these values, the above variables can be changed.

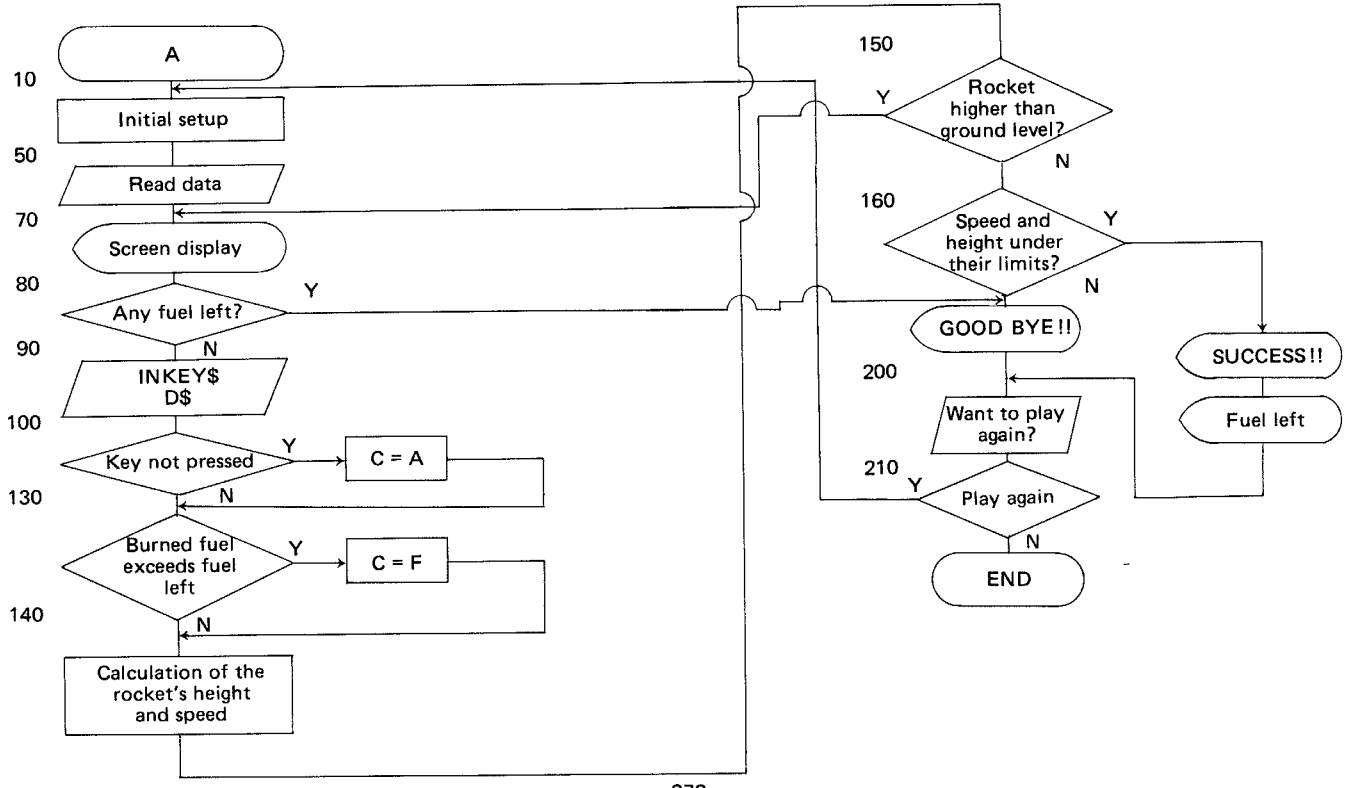
INSTRUCTIONS

1. It is started by pressing (DEF) (A) . Press (0) ~ (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	H: 500 S: −50 F: 200 C: 0	
	9	H: 452 S: −46 F: 191 C: 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
N
90
INKEY$
D$
100
Key not pressed
Y
C = A
130
N
Burned fuel exceeds fuel left
Y
C = F
140
N
Calculation of the rocket's height and speed
272
150
Rocket higher than ground level?
Y
N
160
Speed and height under their limits?
Y
N
GOOD BYE!!
200
Want to play again?
210
Play again
Y
N
END
SUCCESS!!
Fuel left

PROGRAM LIST

```
10:"A": WAIT 50: CLEAR
   : USING :S=-50:A=0:D
   $=""
20:BEEP 3: PRINT "  ***
   START ***"
30:DATA "TIME=",50,"FUE
   L=",200,"HEIGHT=",50
   0
40:RESTORE
50:READ B$,W,B$,F,B$,H
60:WAIT W
70:PRINT USING "####";"
   H:";H;" S:";S;" F:";
   F;" C:"; STR$ C
80:IF F<=0 GOTO 170
90:BEEP 1: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 BEEP 5: PRINT
   "SUCCESS!!": GOTO 18
   0
170:BEEP 3: 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

438
```

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$	√

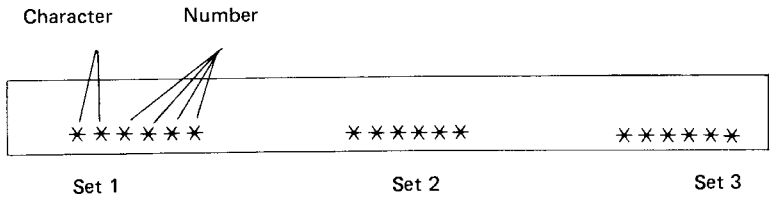
Program Title: MEMORY CHECKER

OVERVIEW

Three lines with a total of 18 characters will be displayed on the screen for approximately 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 approximately 5 seconds. There are 2 characters and 4 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 3 and latter 3 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**

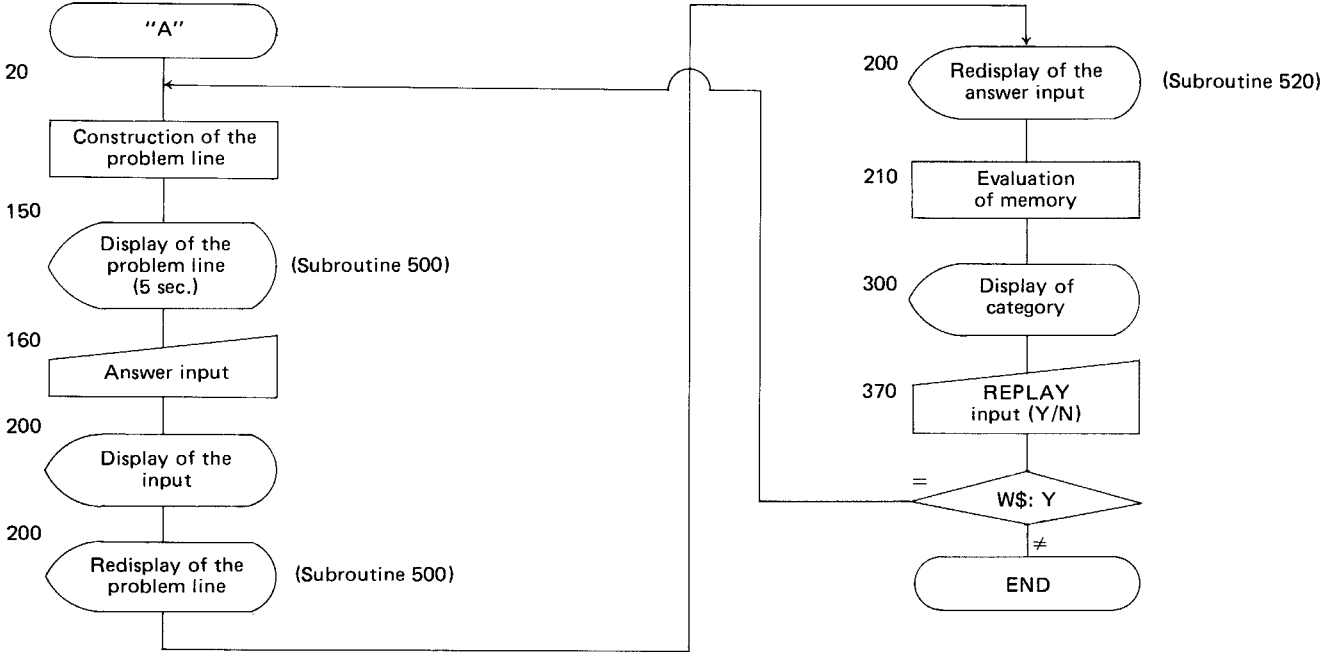
KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	MEMORY CHECK	Title
2		******XXXXXX ******XXXXXX ******XXXXXX	Display of problem line (5 sec.) * . . . character X . . . number
3		ANS. = _	Waiting for the input of set 1
4	(Example) AB1234 ENTER	ANS. = _	Waiting for the input of set 2

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
5	✱✱XXXX (ENTER)	ANS. = _	Waiting for the input of set 3
6	✱✱XXXX (ENTER)	✱✱XXXX ✱✱XXXX ✱✱XXXX	Display of the problem line (1.5 sec.)
7		✱✱XXXX ✱✱XXXX ✱✱XXXX	Display of the answer input
8		IDIOT	display of category
		BAD	
		AVERAGE	
		OK	
		GOOD!	
		✱INTELLIGENT✱	
		✱✱GENIUS✱✱	
9		✱REPLAY (Y/N) ?	Player input request
10	(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
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

PROGRAM LIST

```
10:"A": USING : WAIT 20
   0: PRINT "MEMORY CHE
   CK": CLEAR : RANDOM
20:DIM G$(6)*1,N$(10)*1
   ,V$(3)*3,X$(3)*6,Z$(
   3)*3,Y$(3)*6
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$(I)=" "
100:FOR J=1 TO 3:K= RND
   9
110:Y$(I)=Y$(I)+N$(K):
   NEXT J
120:L= RND 9:J=(I-1)*2+1
130:A$(I)=G$(J)+G$(J+1)+
   N$(L)
140:H$=Y$(I):A$(I+3)=
   RIGHT$ (H$,3): NEXT
   I
150:GOSUB 500
160:FOR I=1 TO 3
170:INPUT "  ANS. = ";X$
   (I):X$(I)= LEFT$ (X$
   (I),6)
180:Z$(I)= LEFT$ (X$(I),
   3)
190:V$(I)= RIGHT$ (X$(I)
   ,3): NEXT I
200:GOSUB 520: GOSUB 500
   : GOSUB 520
210:N=0
220:FOR I=1 TO 3
230:IF A$(I)=Z$(I) LET N
   =N+1
240:IF A$(I+3)=V$(I) LET
   N=N+1
250:NEXT I
260:N=N+1
270:WAIT 150: ON N GOTO
   300,310,320,330,340,
   350,360
300:BEEP 1: PRINT "  IDI
   OT": GOTO 370
310:BEEP 1: PRINT "  BAD
   ": GOTO 370
320:BEEP 2: PRINT "  AVE
   RAGE": GOTO 370
330:BEEP 2: PRINT "   OK
   ": GOTO 370
340:BEEP 3: PRINT "   GO
   OD!": GOTO 370
350:BEEP 4: PRINT "* INT
   ELLIGENT *": GOTO 37
   0
360:BEEP 5: PRINT "**GEN
   IUS**"
370:W$="": BEEP 1: 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: BEEP 2:
   PRINT A$(1);A$(4);"
     ";A$(2);A$(5);"
   ";A$(3);A$(6)
510:RETURN
520:WAIT 80: BEEP 1:
   PRINT USING "&&&&&&"
   ;X$(1); USING ;"   "
   ; USING "&&&&&&";X$(
   2); USING ;"   ";
   USING "&&&&&&";X$(3)
525:USING
530:RETURN
600:END

891
```

MEMORY CONTENTS

A$	3 columns of characters
B$	
C$	
D$	
E$	
F$	
G	
H$	√
I	Index
J	Random number generation
K	
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$(3)*3	3 columns after answering (1 ~ 3)
X$(3)*6	Work (1 ~ 3)
Y$(3)*6	Work (1 ~ 3)
Z$(3)*3	3 columns before answering (1 ~ 3)

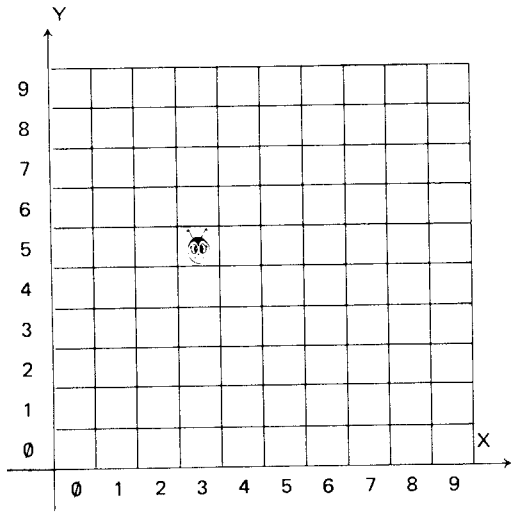
Program Title: BUGHUNT

OVERVIEW

This is a game involving a man chasing after a bug.

CONTENTS

The bug moves according to random numbers.
The man chases the bug and kills it.
The man moves by using the (8) (4) (6) (2) keys. (INKEY$ is used)
Each time the man moves one space, so does the bug. (Sometimes the bug will stay in the same place.)
Initially, the man is in position (Ø, Ø).
The bug is placed at a position that was chosen at random.
Hints are displayed as distance.
The distance is displayed by the ABS(X−a) + ABS(Y−b) equation.
The initial energy level is 100. This decreases by 1 with time.
Each time that a bug is killed, the energy increases by 5, 10, or 15. (The amount is chosen randomly.)
The score is determined by how many bugs were killed when the energy level reaches 0.
(The position of the bug may "warp" when cornered.)
The program can be started by either pressing RUN (ENTER) or (DEF) (A).



Position of the man (X, Y)
Position of the bug (a, b)

Concerning the display (Small characters are actual values)

(x, y)	DISTANCE = ℓ	E = e

Present position (X coordinate, Y coordinate) — Hint (distance) — Remaining energy

- Each time the man moves the display changes

Bug is caught

HIT!	HIT!

BANG!	BANG!

SCORE	t	ENERGY	e

Concerning the BEEP sound

★ Hint: When the distance is 1 the BEEP goes off 3 times
2 " 2
3 " 1

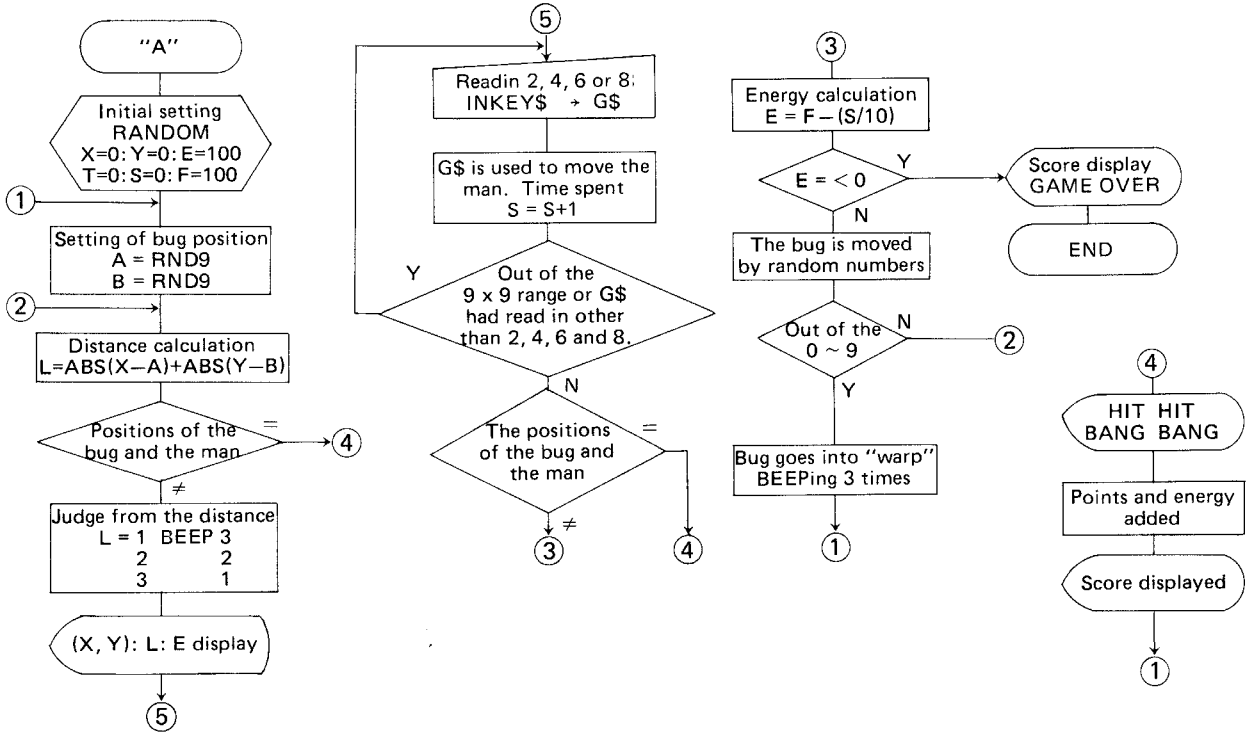
★ If the distance is greater than 3 no BEEP is given.

★ When the bug is caught, the BEEP goes off 5 times.

KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	(DEF) (A)	(0, 0) DISTANCE = 5 E = 100	
	8	(0, 1) DISTANCE = 4 E = 99	
	6	(1, 1) DISTANCE = 2 E = 98	2 BEEPs
	8	HIT! HIT!	5 BEEPs
		BANG! BANG!	
		SCORE 1 ENERGY 108	

FLOWCHART


"A"
Initial setting
RANDOM
X=0: Y=0: E=100
T=0: S=0: F=100
1
Setting of bug position
A = RND9
B = RND9
2
Distance calculation
L=ABS(X−A)+ABS(Y−B)
Positions of the bug and the man
=
4
≠
Judge from the distance
L = 1 BEEP 3
2 2
3 1
(X, Y): L: E display
5
5
Readin 2, 4, 6 or 8:
INKEY$ → G$
G$ is used to move the man. Time spent
S = S+1
Y
Out of the 9 x 9 range or G$ had read in other than 2, 4, 6 and 8.
N
The positions of the bug and the man
=
≠
3
4
3
Energy calculation
E = F − (S/10)
E = < 0
Y
Score display
GAME OVER
END
N
The bug is moved by random numbers
Out of the 0 ~ 9
N
2
Y
Bug goes into "warp"
BEEPing 3 times
1
4
HIT HIT
BANG BANG
Points and energy added
Score displayed
1

PROGRAM LIST

```
10:"A": RANDOM : WAIT 2
   50: PRINT "** BUGHUN
   T GAME **": BEEP 3
20:X=0:Y=0:E=100:F=100:
   T=0:S=0
30:A= RND 9:B= RND 9
40:L= ABS (X-A)+ ABS (Y
   -B)
50:IF X=A AND Y=B GOTO
   400
100:IF L=1 BEEP 3
110:IF L=2 BEEP 2
120:IF L=3 BEEP 1
130:WAIT 50: PRINT "(";
   STR$ (X);",";STR$ (
   Y);") DISTANCE=";
   STR$ (L);" E=";STR$
   (E)
150:S=S+1:E=F- INT (S/2)
153:IF E<=0 THEN 500
155:G$= INKEY$ : IF G$="
   " GOTO 130
157:BEEP 1
160:IF G$="2" LET Y=Y-1:
   GOTO 210
170:IF G$="4" LET X=X-1:
   GOTO 210
180:IF G$="6" LET X=X+1:
   GOTO 210
190:IF G$="8" LET Y=Y+1:
   GOTO 210
200:GOTO 150
210:IF X<0 LET X=0: GOTO
   150
220:IF Y<0 LET Y=0: GOTO
   150
230:IF X>9 LET X=9: GOTO
   150
240:IF Y>9 LET Y=9: GOTO
   150
250:IF X=A AND Y=B GOTO
   400
260:E=F- INT (S/2)
270:IF E<=0 GOTO 500
280:R= RND 5
290:IF R=1 LET B=B-1:
   GOTO 340
300:IF R=2 LET A=A-1:
   GOTO 340
310:IF R=3 LET A=A+1:
   GOTO 340
320:IF R=4 LET B=B+1:
   GOTO 340
340:IF A<0 OR A>9 GOTO 3
   70
350:IF B<0 OR B>9 GOTO 3
   70
360:GOTO 40
370:BEEP 4: PAUSE "*** W
   ARP ***": GOTO 30
400:PAUSE "HIT! HIT!"
410:BEEP 5
420:PAUSE "BANG!  BANG!"
430:T=T+1:C= RND 3*5:F=F
   +C
435:E=F- INT (S/2)
440:WAIT 100: PRINT "SCO
   RE ";T;" ENERGY ";E
450:GOTO 30
500:WAIT : PRINT "SCORE
   "; STR$ (T);" *GAME
   OVER!!*"
510:END

732
```

MEMORY CONTENTS

A	Position of bug X coordinate
B	Position of bug Y coordinate
C	Amount of energy added
D	
E	Remaining energy
F	Energy level
G$	Key read in
H	
I	
J	
K	
L	Distance between bug and man
M	
N	
O	
P	
Q	
R	Size of bug movement
S	Time spent
T	Score

U	
V	
W	
X	Man position X coordinate
Y	Man position Y coordinate
Z	

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.

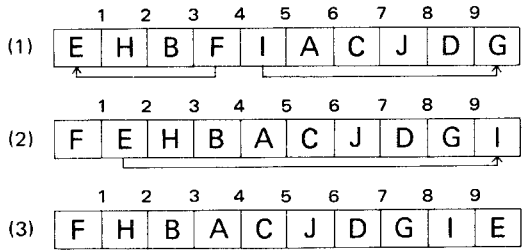
It is fun to race with 2 or 3 of your friends.

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 breakpoints (1 ~ 9) where the numbers are placed. Inputting 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

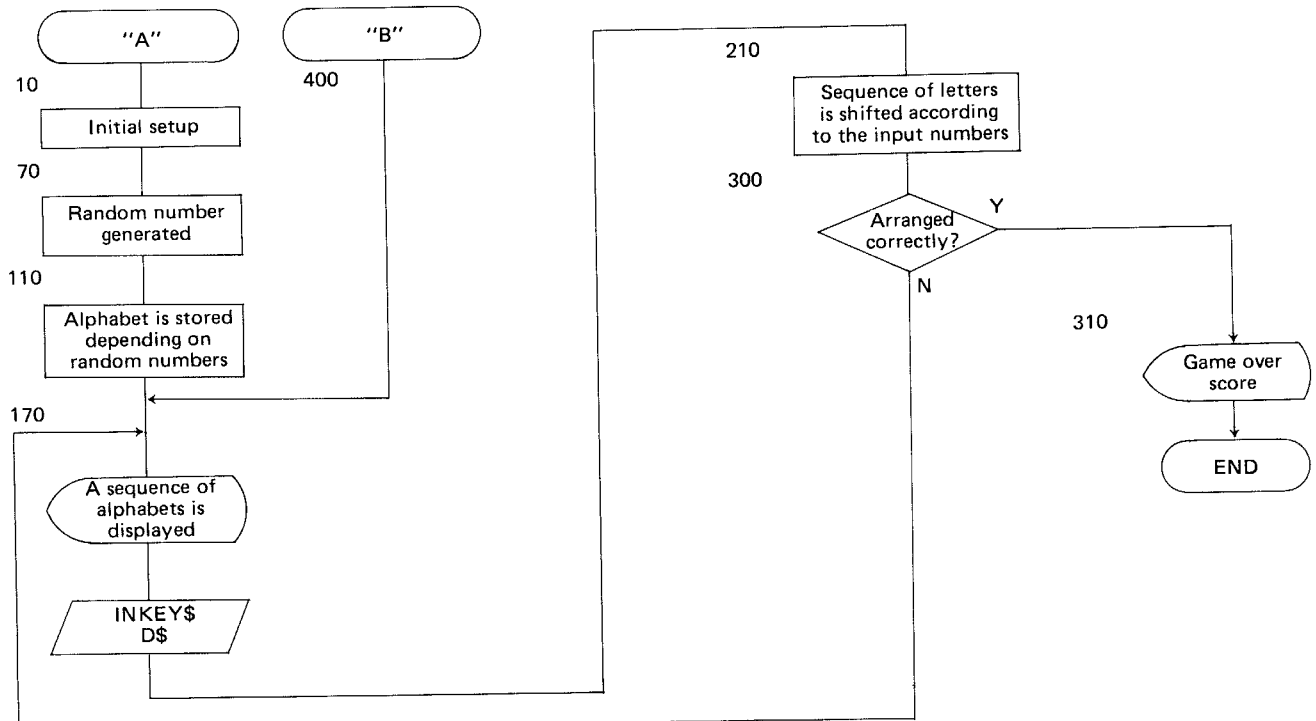
In (1), 4 is input; "F" and "I" move to each side, changing the configuration to (2). If 1 is now input, the "E" moves to the far right; but "F" stays in its place because it is already in the far left position, becoming configuration (3).



KEY OPERATION SEQUENCE

Step No.	Key Input	Display	Remarks
1	DEF A	DOUBLE ROTATION	
		A ~ J	Random sequence 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 alphabets?
1	DEF B	A ~ J	
		Same as DEF A in succession	

FLOWCHART


"A"
"B"
10
400
Initial setup
70
Random number generated
110
Alphabet is stored depending on random numbers
170
A sequence of alphabets is displayed
INKEY$
D$
210
Sequence of letters 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$(1)
130:N=0
150:BEEP 1
170:D$="": 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:BEEP 5: PAUSE "GAME
    END"
320:WAIT 200: PRINT
    USING "####";"YOUR S
    CORE";N
330:END
400:"B": WAIT 50: GOTO 1
    20

471
```

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)	Alphabet sequences

ADDENDUM
Cat. No. 26–3590

1. With the PC-3, you can use an array as a first element for a two dimensional array. An array as a second element will not work.

Example:
B (C (Ø), 5) – – OK
B (5, C (Ø)) – – NO
There is one exception. If the inner array is "A( )", it can be used as the second element of the two dimensional array.

2. When the decimal places as set by the statement "USING" and the number to be displayed or printed becomes Ø, the last Ø is dropped.
Example: If decimal places are set with USING "##.#":
If A = Ø.1, Ø.1 will be displayed
If A = Ø.Ø1, Ø will be displayed instead of Ø.Ø

One way you can get around this result is to use a program such as the following:

```
1Ø USING "##.#"
2Ø IF A<Ø.1 THEN PRINT USING "##.##" ;A/1Ø : GOTO 4Ø
3Ø PRINT A
4Ø ------
```

3. In the Reserve mode, do not press [ENTER] whenever "TO" is used in a character string (GOTO is ok). The contents of the Reserve mode will change if [ENTER] is pressed.

Radio Shack
Fort Worth, TX 76102

IMPORTANT INFORMATION

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

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

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

CUSTOM MANUFACTURED FOR RADIO SHACK, A DIVISION OF TANDY CORPORATION

RADIO SHACK, A DIVISION OF TANDY CORPORATION
U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U.K.
91 KURRAJONG ROAD MOUNT DRUITT, N. S. W. 2770	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

3A3

PRINTED IN JAPAN