Personal Computer

PB-770

OWNER'S MANUAL

CASIO®

(英)

Personal Computer

PB-770

OWNER'S MANUAL

CASIO®

INTRODUCTION

Carrying the compact PB-770 personal computer is equivalent to taking along the capacity of a tabletop model. RAM (Random Access Memory) can be expanded up to 32K bytes, which gives you a broad range of 30,000 character storage.

Writing (storing) data in the RAM is an easy process using the BASIC language programming described in this manual. And once the data you want has been stored, it can be read selectively anytime or changed to suit new applications. Remember, a computer is worthless without a program, so the more you learn about BASIC language programming, the more you will be able to utilize the many advantages of personal computing.

Once you have seen how well the PB-770 can serve you, its functions can be broadened by connecting an optional plotter-printer with cassette interface. This allows the fast preparation of expressive 4-color graphs and tables.

You can be sure that the more you use the PB-770 the closer a companion it will become. The aims of this manual are to first build your familiarity with this clever personal computer so you can start to make full use of it in your daily life.

CONTENTS

OUTLINE

The many features of the PB-770 personal computer present an interesting challenge to the beginner, but there is no instant route to total understanding of its varied uses.

The step-by-step approach is by far the best path to familiarity with each function, and it is usually found that computing skill is gained in direct proportion to time spent on the keys.

So this manual is laid out to introduce the personal computer in an easy way by providing various practice programs in BASIC while training you in actual key operation.

Of primary importance is learning the correct handling of the PB-770. Chapter 1 lays out its features and basic usage procedures. Chapter 2 covers the functions of each key and the display screen preparatory to learning BASIC.

Fundamentally, a computer stores and computes. Chapter 3 sets out to explain how a BASIC program can be prepared to store volumes of data in the PB-770 by an "Array Programming" method for recall whenever required. It also outlines for the beginner the use of graphic programming on the large display of the PB-770 and how to make use of the operational plotter-printer with cassette interface.

Chapter 4 is for the user who has already mastered BASIC. It explains in detail how to use many of the commands and functions of the PB-770. We do not recommend that the beginner learns all commands from this chapter in series. Only 10 BASIC commands are necessary to form a good structural knowledge.

Chapter 5 gives representative programs for use with the PB-770. They are practical, easy to apply, and may be rearranged for specific applications.

CHAPTER 1

GENERAL GUIDE

1-1 PRIOR TO OPERATION

The PB-770 is a product of CASIO's strict testing process, high level electronics technology, and strict quality control.
To ensure long life and trouble-free operation, please observe the following precautions.

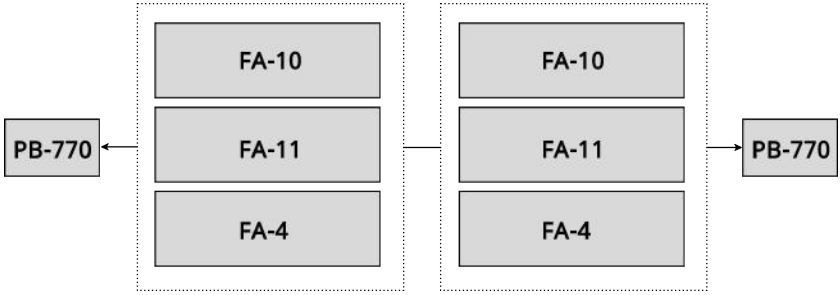
■ **IMPORTANT**

- This computer is constructed of precision electronic components. Never attempt disassembly and/or maintenance. Special care should also be taken to avoid damage by bending or dropping. Do not carry the computer in your hip pocket.
- Use only the optional FA-10 plotter-printer with cassette interface, FA-11 plotter-printer with standard cassette tape recorder and cassette interface, or FA-4 printer interface (CENTRONICS standard) with cassette interface. Never connect any other peripheral equipment to the connector of this unit.
- Avoid temperature extremes. Do not store this unit in an automobile, near a heater, or any other location where it may be exposed to high temperatures. Also avoid use and/or storage in areas subject to high humidity and dust. Extremely low temperatures can slow display response or cause the display to cease operating. Normal operation will return when the temperature is brought back to normal.
- Clean the unit by wiping its surface with a soft, dry cloth or a cloth dampened with a neutral detergent. Never use such liquids as thinner or benzine.
- Before assuming malfunction, first check the power supply and confirm that there are no programming errors.
- Should service be required, contact your nearest dealer.

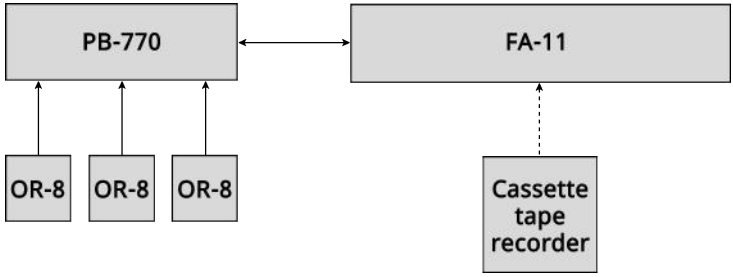
1-2 SYSTEM CONFIGURATION AND CONNECTIONS

FA-11: Plotter-printer with standard cassette tape recorder and cassette interface
FA-10: Plotter-printer with cassette interface
CM-1: Microcassette tape recorder
FA-4: Printer interface (CENTRONICS standard) with cassette interface
OR-8: RAM expansion pack (8K bytes)

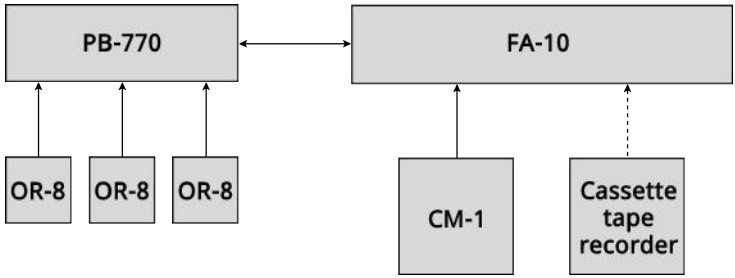
■ Connecting to another PB-770 (Unit-to-unit programs/data transference)



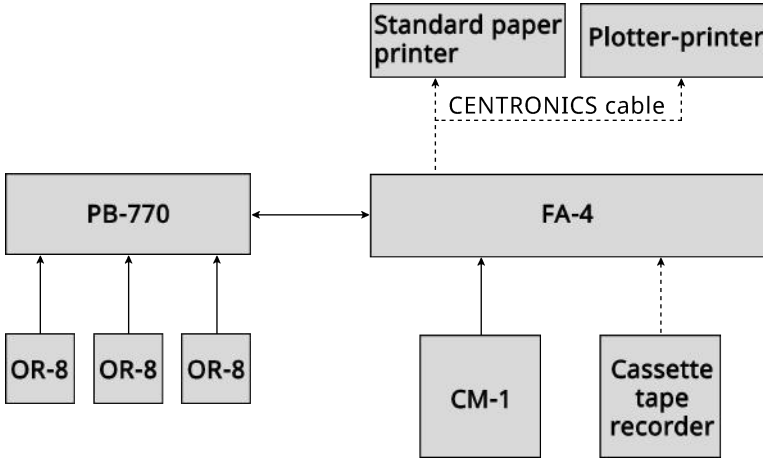
■ Connecting an FA-11



■ Connecting an FA-10



■ Connecting an FA-4



1-3 BATTERY MAINTENANCE

■ Battery Loading

Turn the power of the PB-770 OFF. Turn the computer over and slide open the battery compartment lid (Fig. 1).

Load four AA size batteries. Batteries may burst if not installed correctly. Always ensure that the polarities of the batteries are correct. Load the batteries so that their minus (—) poles are contacting the springs located in the battery compartment (Fig. 2).

Mixing old and new batteries will considerably shorten overall battery life. Therefore, always load or replace a full set of new batteries.

Battery leakage can damage the unit and cause malfunction. Always remove batteries when the unit is not used for extended periods.

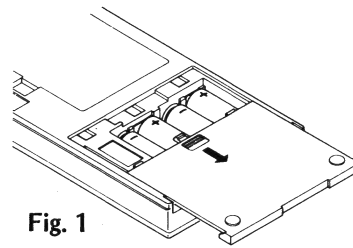


Fig. 1

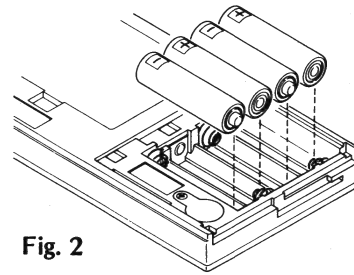


Fig. 2

■ Battery Replacement

When the buzzer sound produced by the BEEP command weakens or when the display becomes blank, replace the batteries with new ones. (For battery specifications, see page 329.)

■ Power Source Configuration

Since the PB-770 power source is divided into a main power source and a sub power source for RAM backup (Fig. 3), programs or data in the mainframe are not lost during main power source or sub power source battery replacement. Programs or data are only lost when the main and sub power sources are removed at the same time.

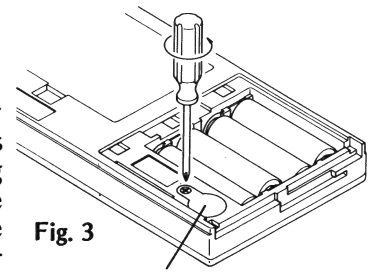


Fig. 3

RAM backup battery

* In order to prevent the chance of malfunction due to battery leakage, be sure to replace the main batteries and RAM backup battery every 2 years regardless of how much they are used.
* With the main batteries removed, the sub battery protects RAM for approx. 4 months when RAM capacity is standard 8K bytes (approx. 1 month when RAM capacity is expanded to 32K bytes).

* When both the main batteries and the RAM backup battery are replaced, be sure to enter NEW ALL [↵] after replacing batteries.
* **Keep batteries away from children. If swallowed, contact a physician immediately.**

■ Auto Power Off

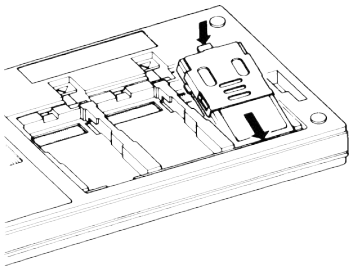
Power is automatically switched off approximately 6 minutes after the last key operation (except during program execution) to conserve power. Power can be restored by pressing the [ON/BRK] key or turning the power switch OFF and then ON again.

■ Low-voltage Detection Feature

The Low-voltage Detection feature of the PB-770 protects RAM contents when the voltage of the main batteries drops below a certain level.
When the voltage decreases, the whole display becomes blank or the display during program execution blinks, and the PB-770 is no longer operable. The batteries must then be immediately replaced. RAM contents will be altered by switching the unit ON with dead batteries installed.

1-4 RAM EXPANSION PACK (OPTIONAL)

Fig. 1 Turn power OFF.



Insert the RAM Expansion Pack as shown above.

Fig. 2 Slide the holder in the direction of the arrow.

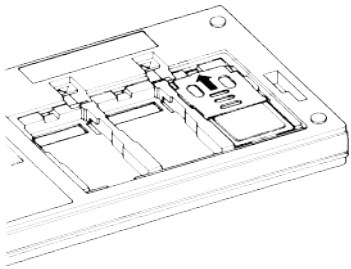
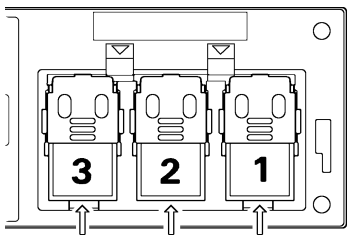


Fig. 3 RAM Expansion Pack insertion sequence.



RAM 3 RAM 2 RAM 1

The basic unexpanded RAM capacity of the PB-770 is 8K bytes. The RAM capacity can be expanded up to a maximum of 32K bytes by installing optional OR-8 RAM expansion packs.

One RAM expansion pack provides 8K bytes. Expansion packs are installed by the following procedure.

(1) Turn off the power of the PB-770.

(2) Turn over the PB-770 and remove the RAM box cover by pressing the two catches.

(3) Holding the sides of the OR-8 RAM expansion pack, insert it into the far right position of the unit (Fig. 1).
Since RAM is very sensitive to static electricity, be careful not to touch the terminals of the OR-8.

(4) Lightly press the holder and slide it into a locked position (Fig. 2).

(5) Install the required number of RAM expansion packs. Then replace the RAM box cover and turn the power of the PB-770 ON.

(6) Enter NEW ALL [↵] . Then confirm the RAM capacity by entering [SHIFT] [SYSTEM S] [↵]

(7) Install the RAM Expansion Packs in the sequence of 1—2—3 (Fig. 3).
If a RAM Expansion Pack is installed in position 2 skipping position 1, correct functions cannot be performed.

1-5 NOMENCLATURE AND OPERATION

- **Display contrast control** . . . Adjust so the display can be easily read.
- **Alphabet keys** . . . . . Used to display upper-case letters. When the alphabet keys are pressed while holding down the [CAPS] key, lower-case letters are displayed.
  Commands or symbols printed above each key can be displayed by pressing the alphabet keys while holding down the [SHIFT] key (one key commands).
  Functions printed below each key can be displayed by pressing the alphabet keys while holding down the [F] key (one key functions).

Example:

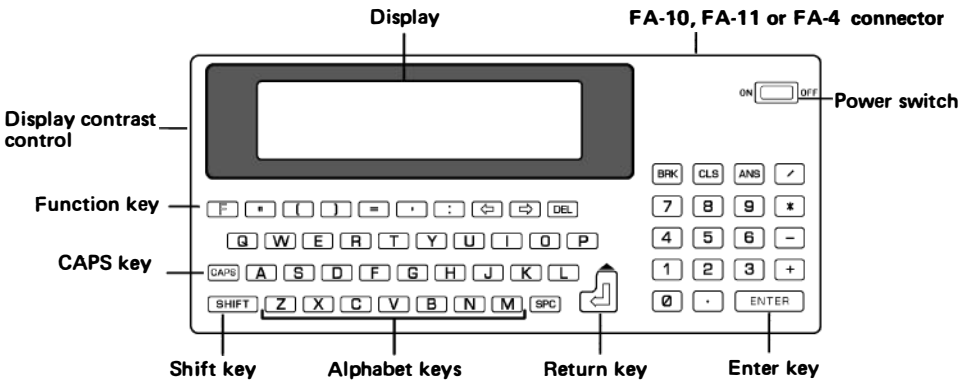
SYSTEM · · · SHIFT mode
[S] · · · DIRECT mode
INKEY$ · · · FUNCTION mode

* Use the CHR$ function for symbols and characters which cannot be input directly by pressing a key. (See page 245.)

- **Return key ([↵])** . . . . . Used to perform BASIC command or program input and output and to execute data input and output.
- **Enter key ([ENTER])** . . . . . Used to execute an instruction during manual calculations.

Example: [1] [2] [/] [4] [ENTER] → 3

- See "2-5 Editing and Special Key Functions" in Chapter 2 for the detailed use of each key.



1-6 TEST OPERATION

This is a demonstration program to show the functions of the PB-770. If you have trouble inputting the program correctly, refer to Chapters 2 and 3.
With this program you will be able to see various functions of the display screen, the BEEP command and the execution speed of graphic commands.

```
 10  CLS
 20 LOCATE 3,1:PRINT "PB-770 TESTING"
 30 FOR A=31 TO 0 STEP -1
 40 DRAW(0,A)-(159,A)
 50 NEXT A
 60 LOCATE 5,1:PRINT "BEEP SOUND"
 70 FOR A=0 TO 9
 80 BEEP 1:BEEP 1:BEEP 0
 90 NEXT A
100  CLS
110 FOR A=33 TO 255
120 PRINT CHR$(A);
130 NEXT A
140 FOR A=0 TO 20
150 NEXT A
160  CLS
170 FOR A=1 TO 16
180 DRAW(A+2,16-A)-(A*3,16-A)-(A*3,15+
    A)-(A+2,15+A)-(A+2,16-A)
190 NEXT A
200 FOR A=0 TO 200
210 NEXT A
```

CHAPTER 2

KEY OPERATION AND DISPLAY

2-1 KEY FUNCTIONS IN DIRECT MODE

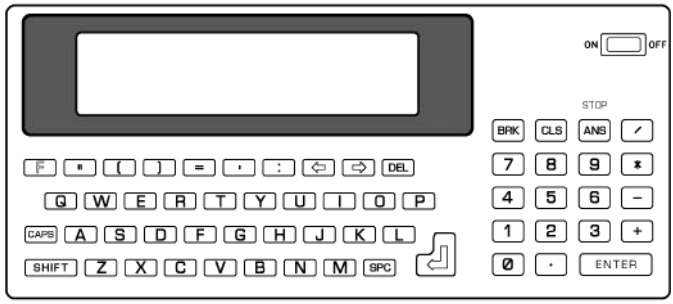
When a key entry is made in direct mode, the character or function inscribed on the key is input.

A — Z	Upper-case alphabetic characters
SPC	Space (blank)
", =, (,), ;, :	Symbols
0 — 9	Numbers
.	Decimal point
+, −, *, /	Arithmetic symbols
CLS	Display clear
DEL	Character delete
BRK	Break (execution halt)
ENTER	Manual calculation execution
↵	Program input and execution
⇦, ⇨	Cursor movement (left, right)
SHIFT	SHIFT mode designation
CAPS	CAPS mode designation
F	FUNCTION mode designation
ANS	Answer key (Most recent calculation result)

■ Number of characters in one statement

A maximum of 79 characters can be entered in a calculation formula during manual calculation or in one line during BASIC programming.

Key Functions In Direct Mode



2-2 KEY FUNCTIONS IN SHIFT MODE

When the SHIFT key is held down and another key is pressed, the brown character, symbol, etc. printed above each key is input.
26 different one-key commands are provided.

&, #, <, >, $, ;, ¥, ?, !, ^, % Symbols

INS Character, symbol insertion

P0 — P9 Program area designation

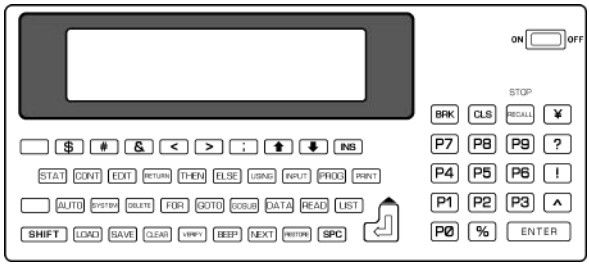
Previous line display in the EDIT mode (see page 134.)

↑, ↓ Cursor movement (up, down)

HOME Cursor movement (to the beginning of a statement)

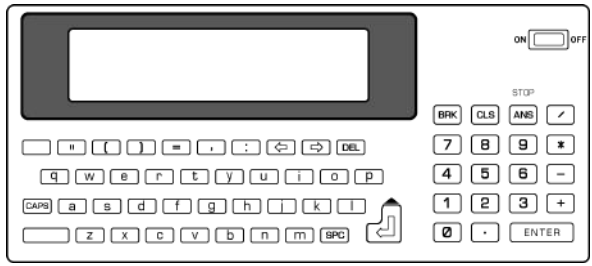
RECALL Recall function

Key Functions In Shift Mode



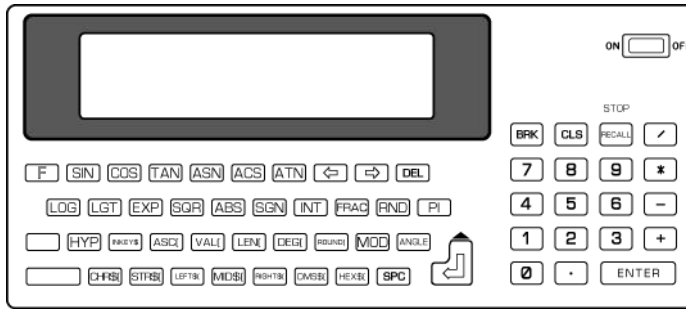
2-3 CAPS MODE

Lower-case letters are input by holding down the CAPS key and pressing another key.



2-4 KEY FUNCTIONS IN FUNCTION MODE

When the [F] key is held down and another key is pressed, the respective function noted below each key is input.



2-5 EDITING AND SPECIAL KEY FUNCTIONS

(1) [SPC] ..... (SPACE) Blank entry in all key modes.

(2) HOME [CLS] ..... (Clear screen/Home) Clears the display and moves the cursor to the top left of the display.
[SHIFT] [HOME] only moves the cursor to the beginning of a statement while the display remains as it is.

(3) INS [DEL] ..... (Delete/Insert) Deletes the character or symbol at the cursor position, and shifts the characters or symbols at the right of the cursor to the left.
[SHIFT] [INS] shifts the characters or symbols at the right of the cursor to the right and inserts a blank.
A repeat function allows the continuous deletion of characters or insertion of spaces when [DEL] or [SHIFT] [INS] is held down.

(4) ON [BRK] ..... (Break) Suspends calculation and program execution. Also used to turn the power ON when auto power off has been activated.

(5) [↑] [↓] [←] [→] ... Moves the cursor left, right, up, down. The repeat function is available only for left, right movement.

(6) [ENTER] ..... (Enter) Executes manual calculation. During key input wait in an INPUT statement, functions as [↵]. Assignment statements, commands, and statements cannot be executed with [ENTER]

(7) STOP RECALL [ANS] . . . . . Stores the result of a previously executed manual calculation. Also stores the numerical value output by a PRINT or LPRINT statement.

Example: 3.4 ∗ 5 [ENTER] → 17

5.8 ∗ 3 − [ANS] [ENTER] →0.4

Pressing [SHIFT] [RECALL] activates a recall function that displays the last calculation formula executed using the [ENTER] key, the last program statement stored using the [↵] key, etc.

Example: 100 ∗ 5 [ENTER] → 500

[SHIFT] [RECALL] →100 ∗ 5

Pressing STOP [ANS] will suspend the execution of a program. Execution can be resumed using the CONT command .

(8) [↵] . . . . . (Return/Line back) Inputs programs and executes commands. Manual calculations cannot be executed using this key. The previous line can be displayed in the EDIT mode (see page 134) by [↑]

(9) [SHIFT] [P0] — [SHIFT] [P9]. . . Specifies a program area and executes the program in the specified area from the first line.

(10) [SHIFT] . . . (Shift) The shift mode is specified by holding down this key. The unit automatically returns to the direct mode when this key is released.

(11) [CAPS] . . . (Capital shift) The CAPS mode is specified by holding down this key. The unit automatically returns to the direct mode when this key is released.

(12) [F] . . . (Function) The function mode is specified by holding down this key. The unit automatically returns to the direct mode when this key is released.

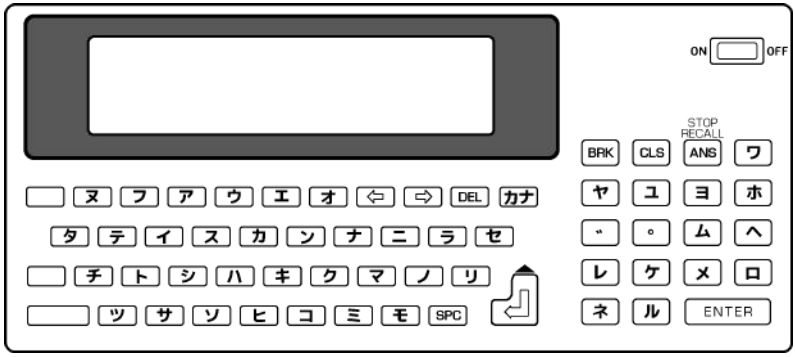
(13) [↵] . . . The line in which an error occurred will be displayed for correction by pressing this key immediately after the error is generated during program execution.

2-5-1 KEY FUNCTIONS IN KANA MODE

Japanese or modded European PB-770s only

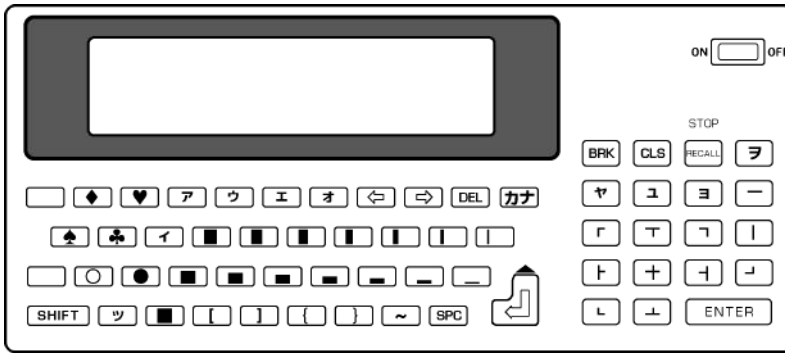
When Kana mode is active after pressing the カナ key, the respective symbol noted below each key is input. Note that Kana mode differs from other modes like CAPS in that it stays active until the カナ key is pressed again.

Key Functions In Kana Mode



When Kana mode is active and the SHIFT key is pressed in combination with one of the keys below, the respective symbol shown in the illustration below is displayed.

Key Functions In Kana+SHIFT Mode



2-6 CALCULATION FUNCTIONS

■ Calculation Precision and Functions

All internal calculations are performed with a 12 digit mantissa (+ 2 digit exponent). Since decimal base operation is used, high precision calculations can be performed.
Manual calculations are executed with the [ENTER] key.
Calculation results are displayed with a 10 digit mantissa (+ 2 digit exponent). In this case, the 11th digit of the mantissa is rounded off.

■ Operator Functions

^	Power
+, −	Addition, Subtraction
∗, /	Multiplication, division
MOD	Remainder calculation (If the numerical value includes a fraction, the fraction is discarded in this operation.)

The calculation range is as follows.

(1) Division by 0 causes an MA error.
(2) When overflow occurs (i.e. when a result exceeds the calculation range), an OV error is generated.
(3) Power range

$0 \wedge 0 \longrightarrow$ MA error
$(\pm x) \wedge 0 \longrightarrow 1$
$0 \wedge y \longrightarrow 0$
$0 \wedge (-y) \longrightarrow$ MA error
$(-x) \wedge (\pm y) \longrightarrow$ Possible only when y is an integer. Otherwise, an MA error is generated.
* Where $x > 0, y > 0$.

■ Calculation Priority

Calculations are executed in the following sequence.
1. Elements in parentheses
2. Functions
3. Powers (^)
4. Plus (+) and minus (−) signs
5. ∗, /
6. MOD
7. +, −

Calculation

Formats

1. $\frac{X+Y}{2}$ ⟶ (X+Y)/2
2. $X^2+2\times Y+Y^2$ ⟶ X^2+2*X*Y+Y^2
3. $-Y^2$ ⟶ -Y^2
4. $(-Y)^2$ ⟶ (-Y)^2
5. $(X^Y)^2$ ⟶ X^Y^2
6. X^{Y^2} ⟶ X^(Y^2)
7. Remainder of $\frac{X}{Y}$ ⟶ X MOD Y

Examples

1. 0.5^0 ⟶ 1
2. -0.5^0 ⟶ -1
3. (-0.5)^0 ⟶ 1
4. 0.5^2 ⟶ 0.25
5. 0.5^-2 ⟶ 4
6. (-0.5)^-2 ⟶ 4
7. 0.5^0.5 ⟶ 0.7071067812
8. (-0.5)^0.5 ⟶ MA error
9. 2^-0.5 ⟶ 0.7071067812
10. (-2)^-0.5 ⟶ MA error
11. 10 MOD 6 ⟶ 4
12. -10 MOD 6 ⟶ -4
13. 10 MOD-6 ⟶ 4
14. -10 MOD-6 ⟶ -4

■ Relational Operators

Relational operators can only be used in an IF statement (see page 186).

=	Equal
<>, ><	Not equal
<	Smaller than
>	Larger than
=>, >=	Either larger than or equal to.
=<, <=	Either smaller than or equal to.

Examples: A + B < > 0. . . The result of A + B does not equal 0.
A$< > "Y" . . . The content of A$ does not equal "Y".
A$ = CHR$(84) + CHR$(79) + CHR$(77) . . . A$ equals "TOM".
CHR$(67) > CHR$(N) . . . CHR$(67), which is C, is larger than CHR$(N) in the Character Code Table (page 327).

■ Operations Using Variables

The contents of variables can be confirmed with the ENTER key.

Example: A ENTER →0

When a numerical value is entered in a variable, the contents of the variable are as follows.

Single-precision: Everyting from the 13th digit of the mantissa to the right is discarded (12 digits). Single-precision is the normal calculation precision.

Half-precision: Everything from the 6th digit of the mantissa to the right is discared (5 digits). Half-precision is 5 digit numerical values realized by specifying an array variable (!). It can only be specified in an array variable. (See page 70).

Half-precision computation

Storage of a half-precision value requires only 4 bytes as opposed to the 8 bytes needed for a single precision value. Except for engineering or scientific applications, 5 digits are usually sufficient for most computations. If computation results concerning such data as test results, percentages, product numbers, prices and quantities can be kept within 5 digits, the amount of RAM area required for data storage is halved and memory space is conserved.

2-7 VARIABLES

■ Kinds of Variables

The PB-770 employs the following types of variables.

(1) Numerical variables

Numerical fixed variables (up to 12 digits).
Numerical registered variables (up to 12 digits).
Numerical array variables (Half-precision numerical array: up to 5 digits. Single-precision numerical array: up to 12 digits).
* The number of digits shown above is the number of internal calculation digits.

(2) Character variables

Character fixed variables (up to 7 characters).
Character registered variables (up to 16 characters).
Character array variables (String length can be specified from 1 to 79 characters. When no length is specified a default value of 16 is automatically used.).

Example:

DIM A$(9, 9) . . . . . .Each character string can be 1 to 16 characters long.
DIM A$(9, 9)*50. . . .Each character string can be 1 to 50 characters long.

■ Fixed Variables (A−Z, A$−Z$)

Memory where numerical values or characters are stored has 26 kinds of Fixed Variables which are A—Z or A$—Z$. Numerical fixed variables and character fixed variables with identical names cannot be used together. If an attempt is made to use identical variable names, a UV error will occur.

Incorrect Usage: 10 PRINT A ; A$ → UV error

■ Registered Variables

In addition to fixed variables, variable names with two characters that consist of either 2 upper-case alphabetical characters or an upper-case alphabetical character and a number, can be used. If a variable name is defined with three characters or more, an SN error will occur during execution.

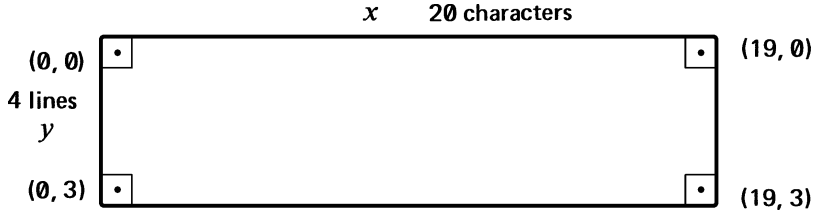
Examples: A B, X 1, Y 1, X 2, Y 2, A Z $, A A $, B 1 $, Z 9 $

- The beginning of a variable name must be an upper-case alphabetical character.
- Reserved words (IF, TO, PI, etc.) cannot be used as variable names.
- A 12-digit mantissa + and 2-digit exponent can be stored in a numerical registered variable (AB, X1, etc.).
- Up to 16 characters can be stored in a character registered variable.
- 40 registered variables including array variable names can be used. If an attempt is made to use more than 40 variables, a VA error will occur that will suspend execution. At this time, the variable names should be cleared using the CLEAR or ERASE command.
- A registered variable name can be recalled by executing LISTV. A numerical registered variable uses 8 bytes and a character registered variable uses 17 bytes.

2-8 DISPLAY SCREEN

■ Character Coordinates

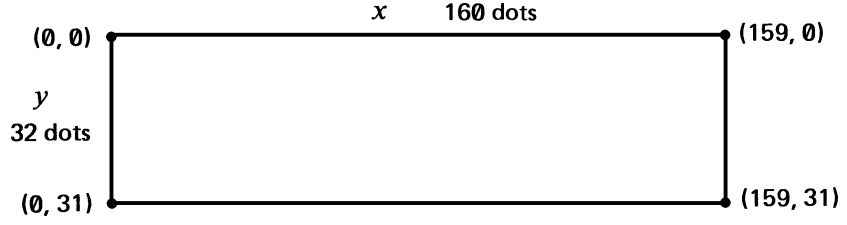
Twenty characters horizontally and four character lines vertically fit in the display window (LCD). Character locations are expressed by a LOCATE statement with the following coordinates.



LOCATE (X, Y) . . . . . See LOCATE.
Based on the coordinates mentioned above, 222 characters in the character code table (see page 327) can be displayed.

■ Graphic Coordinates

Dots can be located on the display based on the coordinates shown below. Dot locations are expressed by a DRAW or DRAWC statement which allows dots and straight lines to be drawn or erased.



DRAW (X, Y) . . . . . See DRAW, DRAWC.

Whether a dot is lit or not can be confirmed using the POINT function.

POINT (X, Y) . . . . . See POINT.

2-9 NUMBER OF BYTES USED FOR VARIABLES

The remaining RAM capacity is reduced each time data are assigned to a variable (except fixed variables) during program execution. The number of bytes used for each type of variable is shown in the table below. Fixed variables from A through Z store data separate from the RAM area so they have no effect on the RAM capacity.

Registered Variables		Array Variables	
Variable	Number of Bytes Used	Variable	Number of Bytes Used
Numerical variable	8	Numerical variable (Half-precision)	4
		Numerical variable (Single-precision)	8
Character variable	17	Character variable (Fixed-length)	17
		Character variable (Defined-length)	2–80 (1–79 characters)

CHAPTER 3

"BASIC" REFERENCE

3-1 INTRODUCTION TO BASIC

No doubt you have probably heard the word BASIC used at one time or another. It stands for "Beginner's All-purpose Symbolic Instruction Code", and it truly is one of the most basic of computer languages.

The beauty of this language is that it allows sophisticated programs to be produced using simple English commands that resemble everyday conversation.

BASIC was developed at a U.S. university in 1964 for use on a large, main frame computer. Since then, however, it has grown in popularity and is now probably one of the most commonly used computer languages.

In this chapter we will learn some of the fundamentals of BASIC that will allow you to eventually develop and write programs of your very own.

3-2 USING THE KEYS

Although the PB-770 has a large data processing capacity and can perform complicated numerical calculations, it can also be used to perform manual calculations without using programs.

To help us get used to the PB-770, let's start with a very simple operation. The following is displayed after power is switched ON.

Ready P0 (This means that the program area is specified to No. 0.)

The numerical keys (ten keys) on the right side of the keyboard are mainly used when the PB-770 is employed as a calculator. [X] , [÷] , and [=] are not included with the ten keys as with a standard calculator. Although [=] is located within the set of main keys it cannot be used as the [=] key on a standard calculator.
The [*] and [/] keys are used for [X] and [÷] respectively, while the [ENTER] key functions as the [=] key.

Let's try 1+2. When you enter [1] [+] [2] , is the following displayed?

```
Ready  P0
1 + 2_
     └ Cursor
```

If you make a mistake, move the Cursor to the location of the mistake using the [←] and [→] keys and input the correct value.
Next, when you press the [ENTER] key, the answer will be displayed as follows.

```
1 + 2
  3
```

Besides the four basic arithmetic functions, the PB-770 is also capable of such operations as powers, trigonometric functions, inverse trigonometric functions and logarithmic functions.

[Numerical Expression]		[Input Format]
$5 \times 6 \div 2$	➡	5 * 6 / 2 ENTER
6.5^2	➡	6.5 ^ 2 ENTER
$\frac{\text{SIN } 30^\circ + \text{COS } 60^\circ}{\text{TAN } 45^\circ}$	➡	(SIN 30 + COS 60) / TAN 45 ENTER

3-3 VARIABLES AND ASSIGNMENT

Now let's try another calculation.

500000 X (1 + 0.07)^{10}, 800000 X (1 + 0.07)^{10}

These expressions compound interest over 10 years and add it to the principals. What is the simplest way to perform these two calculations? Once a calculation expression is input, it is partially available for repeat use.

Therefore,

Enter A = (1 + 0.07)^10 ↵ and input the previous two expressions as

A * 500000 [ENTER], A * 800000 [ENTER]

and the calculation becomes easier to perform.
The value of (1 + 0.07)^{10} is stored in A. This A is called a variable in a program.
To assign numerical value to variable A, the following operation is performed.

A = 176 ↵ (Assign 176 to A)

(Left side) (Right Side)

Assignment is made to store the right side in the left side, so 176 is assigned to A in this example. The assignment instruction is " = ". To confirm that 176 is assigned to A, enter A [ENTER] which should display the contents of variable A. Is 176 displayed?

This point is very important in understanding BASIC.
" = " is the assignment instruction and does not mean equal as used in mathematics (except in a IF statement). For example, enter

A = A + 1 ↵

to assign the value of A + 1 to A. Assuming that 176 is stored in variable A. After the above expression is executed by the computer, 177 is assigned to variable A. Enter A [ENTER] to confirm this.
Is the following displayed?

```
A
 177
```

— (Cursor)

■ [ENTER] and [↵]

It is also important to understand the difference between the [ENTER] and [↵] keys. It should be noted that the [↵] key was used to input A=176 and the [ENTER] key was used to display the value of A in the previous operations.

The [ENTER] key is used the same as the keys on a standard calculator for displaying an answer. This is called manual calculation.

The [↵] (Return) key, on the other hand, is used to execute the commands of a BASIC program. For example, it is used to input a program, to correct a certain part of a program, or to execute a BASIC command.

3-4 USING VARIABLES

An alphabetical character from A to Z or two characters (alphabetical character + one character) such as AA, and N1 are used as variable names. Variables with numerical values assigned as in the previous section can be freely used in calculation expressions.
Now, let's practice. When you enter

A = 36 ↵
B = 12 ↵

36 is assigned to A, and 12 is assigned to B. Next, enter

A + B

since [ENTER] is used "to display an answer" in a manual calculation. If 48 is displayed, perform the next step.

A − B [ENTER]	→ 24
A ∗ B [ENTER]	→ 432
(4 + A) ∗ LOG B [ENTER]	→ 99.39626599

Mastering the use of variables results in a high degree of versatility. Soon, however, you will realize that the simple examples shown above are extremely limited. That is what brings us to programming. Actual programming is not very difficult at all if you can understand how variables are used.

3-5 PROGRAM ENTRY

First of all, let's look at the proper procedure for inputting a program. A look at the keyboard of the PB-770 shows that the alphabetical keys are laid out the same as on a standard typewriter keyboard. Now, perform the following inputs.

[SHIFT] [PROG] [0] [↵] → Specifies program area P0.
[N] [E] [W] [↵] → Erases program stored in P0.
[CLS] → Clears screen.

[Program]	[Key Operation]
10 CLEAR	[1] [0] [SHIFT] [CLEAR] [↵] ⟶ ⌣ indicates pressing two keys simultaneously.
20 A=A+1	[2] [0] [A] [=] [A] [+] [1] [↵]
30 LOCATE 7,2	[3] [0] [L] [O] [C] [A] [T] [E] [7] [,] [2] [↵]
40 PRINT A	[4] [0] [SHIFT] [PRINT] [A] [↵]
50 GOTO 20	[5] [0] [SHIFT] [GOTO] [2] [0] [↵]

Enter [CLS] [R] [U] [N] [↵] or [SHIFT] [P0] . If correct entries were made, numerical characters are displayed at the center of the screen at a high speed:
1, 2, 3 . . . . . .
If "SN error P0 — line No." is displayed to indicate an entry error, correct (debug) the specified line as follows.

[SHIFT] [EDIT] Line No. [↵]

The specified line is displayed when you perform this operation. Move the cursor to the point to be corrected, perform the correct input and press the [↵] key. The next line will be displayed, and, if no correction is required, press the [BRK] key. Then enter [R] [U] [N] [↵] again. It should be noted that the EDIT mode is used for program corrections.

■ Program Areas

The PB-770 contains a total of 10 "program areas" numbered from P0 through P9. Each of the program areas is independent, so up to 10 programs can be individually stored in the unit, recalled when needed and executed without affecting programs in the other pro-gram areas.
When the power of the PB-770 is turned ON, "Ready P0" appears on the display to indicate that the specified program area is P0. The BRK key can be pressed at any time to display the currently specified program area. The following operations are used to change the program area from the one currently specified.

PROG 1 ↵ or SHIFT PROG 0 1 ↵.

3-6 BASIC PROGRAMMING [1]

Now, let's try an actual BASIC program.
A program that obtains the area of a square as the one shown below is prepared using the following sequence.
(1) Request input of the length of one side A.
(2) Multiply the entered numerical value by itself.
(3) Display the result.
(4) Return to (1).


```
10  INPUT  A  ..... ①
20  B = A * A  ..... ②
30  PRINT  B  ..... ③
40  GOTO  10  ..... ④
```

Now, input this program using the following procedure. Be sure to press the keys correctly.

Power ON . . . . . Ready P0 displayed.

NEW[↵] . . . . . [↵] is an input key located on the right below the alphabet keys.

10 [SHIFT] [INPUT T] A [↵] . . . . . Press the [SHIFT] and [INPUT T] keys simultaneously.

20 B=A*A [↵] . . . . . [*] is located in the ten keys.

30 [SHIFT] [PRINT P] B [↵] . . . . . Press the [SHIFT] and [PRINT P] keys simultaneously.

40 [SHIFT] [GOTO G] 10 [↵] . . . . . Press the [SHIFT] and [GOTO G] keys simultaneously.

Next, execute the program using the following procedure.

[CLS] [R] [U] [N] [↵] . . . . . [SHIFT] [P0 Ø] can also be used.

After this entry is made, ? is displayed. _ is called a cursor. Now, enter

8.5 [↵]

and the next display should appear. If it does not appear, check if there is a program input mistake with [SHIFT] [EDIT/E] [↵]. Be careful not to make mistakes concerning the difference between 0 and O, and 1 and I.

```
RUN
? 8.5 . . . . . Value of one side is 8.5.
  72.25 . . . . . Area is 72.25.
? _ (Cursor) . . . . . What is the value of one side ?
```

After confirming the execution of this program, let's analyze it.

```
10 INPUT A
   (Make an entry)  (to A)
```

"10" is a line number which indicates the program execution sequence. It is increased here by 10 for each line (to be explained later). INPUT means to make an entry, or, in other words, " ? " is displayed to indicate that the computer is waiting for an entry. After an entry is made, the command stores it in a numerical variable.

```
20 B = A * A . . . . . Assign the result of A * A to B.
```

Line 20 computes the area. The entered numerical value is multiplied by itself to provide the area of a square, and the result is assigned to B.

```
30 PRINT B
   (Display)  (B)
```

Line 30 displays the area. PRINT is used as a "display" instruction. This line provides the instruction to display the contents of B.

```
40 GOTO 10
   (Go)  (to line 10)
```

GOTO is a command that means "go to" the line with the number that follows the command.

The basics of the commands (INPUT, PRINT, and GOTO) used in this program should be understood after this explanation. However, this program is somewhat imperfect because it does not indicate what the input for "?" should be, or what kind of computation result is provided when this program is executed. Therefore, let's add to this program with the following procedure.

BRK — Ready P0 displayed.

SHIFT EDIT E ↵ — Press SHIFT and EDIT E keys simultaneously.

After the line 10 is displayed, press ← to move the cursor to the location of A, then enter " A = " SHIFT : A ↵ .
The modified result is as follows.

```
10 INPUT "A=" ; A
```

A message with the characters inside " " is displayed.

When a message is inserted, a semicolon (;) is required before the variable. Be careful not to enter a colon (:) by mistake.

Next, line 20 is displayed. Since this line is not to be changed, press ↵ .
Line 30 is displayed next. Move the cursor to the location of B, and enter " A R E A SPC " SHIFT : B ↵ . The modified result is as follows.

```
30 PRINT "AREA ␣" ; B
```

Message is displayed inside " ".

B is displayed following message.

One space.

After this, line 40 is displayed. Since this line is not to be modified, press ↵ .
Now, let's execute the program.

CLS R U N ↵ . . . . . SHIFT P0 can also be used.

Now this program is considerably improved compared with the previous program. This is because it asks "A=?" and displays "AREA 72.25" as the computation result.

3-7 BASIC PROGRAMMING [2]

Now let's look at another program that will help us get better acquainted with BASIC. In this program, the multiples of a specified value are produced within the range of 0 to 200.

```
10 REM MULTIPLE ↵ ···· Press the ↵ key at the end of each line.
20 A=0 ↵
30 INPUT "NUMBER";N ↵
40 A=A+1 ↵
50 B=N*A ↵
60 IF B>200 THEN 100 ↵
70 PRINT B; ↵
80 INPUT " OK";C$ ↵
90 GOTO 40 ↵
100 END ↵
```

After you finish inputting the program, press and hold down the SHIFT key followed by the EDIT (E) key. Then, when you press the ↵ key, the first line of the program will be displayed. If there is a mistake in the first line, move the cursor into position and correct the error.

[1] 10 REM MULTIPLE See page 209.

Line number — Command — Label

A BASIC program consists of line numbers, instructions (program instructions or part of an instruction), and variables or expressions that use variables.
Since line 10 is a REM statement, a label is provided to indicate this is a multiple program. Anything following REM is not executed. Now, press the ↵ key.

[2] 20 A = 0

Line number — Numerical variable (Fixed variable)

In line 20, a value of 0 is assigned to variable A as the first step of the program. This is called variable initialization.

Note that until now all lines have been numbered in multiples of 10. Actually, any line number between 1 and 9999 can be used in the PB-770. Numbering the lines of a program in multiples of ten makes the program easier to read and modify.
Line 20 can also be expressed as follows.

```
20 LET A = 0
```

See page 195.

LET is an optional assignment command.
Now, press [EXE].

[3]

```
30    INPUT    "NUMBER" ; N
```

Line number — Input command — Message statement — Variable

See page 189.

Since INPUT is an input command statement, execution is not shifted to the next line unless the input of a numeral or character is performed by a key entry. A numeral or character that is entered is assigned to the variable following the message statement, and execution proceeds to the next line.
Although the message statement can be omitted, it is used to tell the operator what kind of data should be input.
When " ; " is placed after the message statement of INPUT, " ? " is displayed after the message. If " , " is used, " ? " is not displayed.
Now, press [EXE].

[4]

```
40 A=A + 1
```

Variable A is a counter that keeps track of how many times execution is performed for each entered value. Variable A is initialized to 0 in line 20. The first execution of line 40 performs A=A+1 (A=0+1), so the value of A is set to 1.
When line 40 is executed at this time (A=1), A=A+1 (A=1+1) is performed and A takes on a value of 2. The value of A will increase by 1 each time line 40 is executed. No matter what the value of A at the end of the program, however, it is always set to 0 (in line 20) when the program is executed from the beginning.

SEE

Line 40 execution:	1st time	A = 0 + 1	(A = 1)
	2nd time	A = 1 + 1	(A = 2)
	3rd time	A = 2 + 1	(A = 3)
		⋮	

Press the [EXE] key.

[5] 50 B = N * A

This is an assignment statement (the same as line 40) which assigns the value of N * A to numerical variable B. When this line is first executed, 1 has been assigned to A by the execution of line 40. An optional numerical value has been assigned to N by the execution of INPUT in line 30.
Therefore, if N is 17,

B = 17 * 1 ⟶ 17 is assigned to B.

SEE In the following execution of line 50, a value of 2 is assigned to A the second time,

2nd time B = 17 * 2
3rd time B = 17 * 3
⋮

} The multiple of N is continuously assigned to the numerical variable B, cancelling any previous value of B.

Now let's look at the next line. Press ↵

[6] 60 IF B>200 THEN 100

Line number to which a jump is made.
Conditional expression
Decision statement

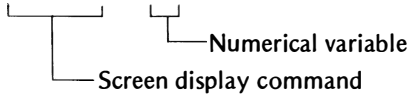
This is a conditional statement that says "If (IF) the value of B is larger than 200 (B>200), jump to line 100." In other words, if the value of B is equal to 200 or less, execution proceeds to the next line without a jump. When line 50 is repeatedly executed, the value of B becomes 204 after 12 times (17x12), B>200 is realized and a jump (branch) is made to line 100.

IF expression THEN line number See page 186.

If this expression is realized, the program jumps to a specified line number.

Now press ↵ .

[7] 70 PRINT B ; See page 198.



PRINT is a display command. In this program, this command displays the contents of numerical variable B on the screen. The semicolon after B is used to keep everything displayed continuously. Because of this, the display called for in line 80 will occur directly after B without line change.

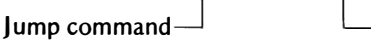
[8] 80 INPUT " OK" ; C$ See page 189.

The INPUT statement you learned in the section of line 30 is used again in this line to perform character key input. When line 80 is executed, "OK?" is displayed as a message statement by which key input of up to 7 characters can be assigned to character variable C$. If numerical variable C is used here, a numerical value is only accepted as key input, and a character or ↵ entry results in an SN error.

The function of this line is to temporarily stop the display of the computation result of line 70 using the INPUT statement which waits for key input. If this line is not provided, many results are repeatedly displayed at one time.

Press the ↵ key.

[9] 90 GOTO 40 See page 184.



This is a command for an unconditional jump to line 40. Press the ↵ key.

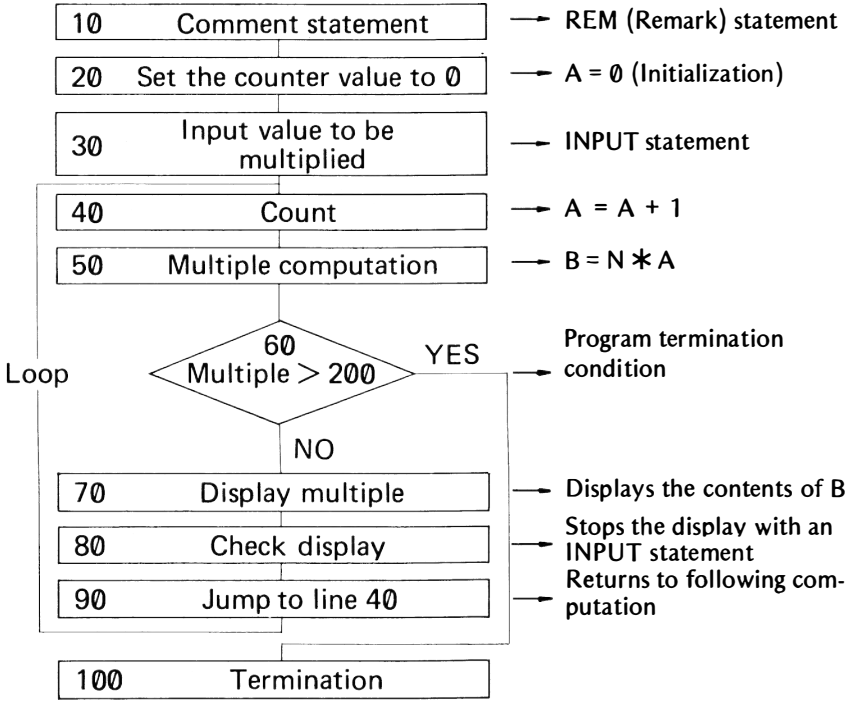
[10] 100 END See page 170.

END is a command that terminates the program. END is an essential command for a program because if a subroutine follows, the subroutine is also executed. (See page 180 for subroutines.)

END can be inserted in line 60 as follows. If this is done, line 100 is not required.

```
60 IF B>200 THEN END
```

Let's observe the program flow as shown in the flow chart below.



Line 90 causes an unconditional jump (jump is always performed) back to line 40, so the program is continuously executed between these two lines. This execution is exited to line 100 by the conditional statement in line 60 when the product of the multiple times the input value exceeds 200.

EXERCISE

■ **Prepare a program to compute the accumulated sum of a series of input numerical values.**

Hint

(1) Clear all variables to make them 0.
(2) Request input of a numerical value.
(3) Add the numerical value to the sum of the previous numerical values.
(4) Display the result.

Answer

```
10 CLEAR
20 INPUT "DATA =";A
30 B = B + A
40 PRINT "TOTAL =";B
50 GOTO 20
```

Explanation

CLEAR in line 10 is a command that clears all numerical and character variables. In this case, only variable B would be cleared by B=0. If the GOTO command in line 50 causes a jump to line 10, the variable becomes 0 and accumulation cannot be performed.

3-8 PROGRAM EXECUTION

Now the program that was input has been checked.
Let's execute this program.
Press the BRK key and the CLS key first. Then enter the program execution command R U N ↵ . If a program is written in the P0 area, SHIFT P0 functions the same as RUN ↵ .
If program input has been correctly performed, the following will be displayed.

```
RUN
NUMBER ?_
```

The input of a value is requested. Perform the following key operation.

17 ↵

Then the following will be displayed.

```
RUN
NUMBER ? 17
 17 OK ?_
```

The display shows that the minimum multiple of 17 is 17 and confirmation is requested. Press the ↵ key to display the next multiple.

```
RUN
NUMBER ? 17
 17 OK ?
 34 OK ?_
```

Press the ↵ key for the next multiple. After this, multiples up to 187 will be displayed by repeating this operation. If you press the ↵ key again and the next multiple does not appear, the conditional expression in line 60 has been fulfilled. Since the multiple exceeds 200, program execution terminates. To execute the program again, press R U N ↵ again.

Program Modification

Further program modification will help to learn more about the use of various commands.
First, multiples up to 300 can be obtained by changing B>200 in line 60 to B>300. Line 60 can be displayed by SHIFT EDIT(E) 60 ↵. Move the cursor to the position and change 200 to 300. Next, press the ↵ key and BRK key. Then enter R U N ↵.
Let's check how the display is changed by changing the PRINT statement in line 70.
Line 70 reads "PRINT B;". The semicolon after B functions to continue the display as you already know. Now, let's delete the semicolon.

SHIFT EDIT(E) 70 ↵

Program Execution Sequence

```
NUMBER? 17
 17  OK?
 34  OK?
 51  OK?
 68  OK?
 85  OK?
 102  OK?
 119  OK?
 136  OK?
 153  OK?
 170  OK?
 187  OK?
Ready  P0
```

Move the cursor to the position at " ; " and press the DEL key. Note that " ; " disappears. Press the ↵ and BRK keys and run the program.
The message statement "OK?" will now be displayed under the multiple. In other words, since " ; " is gone, the next character is not displayed in the same line, it is displayed in the next line.

3-9 DISPLAY SCREEN CONFIGURATION

Now, let's learn some techniques of screen display control by changing line 70 of the program that was prepared in the previous section.

BRK SHIFT EDIT(E) 70 ↵

The following display should appear.

```
70 PRINT B
```

See page 198.

Let's display variable B together with variable A which is used as the program repeat counter.

[1] 70 PRINT A ; B ;

Move the cursor to the position of variable B and enter SHIFT INS(DEL) INS(DEL)
Now a 2 character space is provided before B. Enter A SHIFT : → SHIFT :
Don't forget to finally press ↵. Press BRK , and then execute the program to check the display. Next, make the following modification.

[2] 70 PRINT A, B ;

To make the modification, press the BRK key, display line 70 in the edit mode and use the cursor keys as outlined above. Finally, press the ↵ key and then run the program again.
The line change is performed by "A, B;".
Let's make another modification. Press BRK .

[3] 70 PRINT A ; TAB (8) ; B;

After the modification, execute the program.
The TAB (8) function moves the cursor to the position which is specified by the number inside ( ). Confirm there is a space between the display of variables A and B. Notice that when 3 numerals are displayed for variable B, the left side is aligned and the last character is shifted as shown below.

```
 3          51  OK?
 4          68  OK?
 5          85  OK?
 6          102  OK?
```

Since this is a display of a multiple, there is no problem here. However, when a quantity or price is displayed, the right side should be aligned. To produce this display, the program should be rewritten using the USING function.

[4] 70 PRINT A; TAB (8); USING "###" ; B;

See page 263 for details on USING. Now, execute the program to check the display.
The LOCATE command also controls the screen display. Let's rewrite line 70 using this command.

[5] 70 CLS : LOCATE 5,2 : PRINT A; B; . . .
Command Command Command

A line in which two or more commands are connected by " : " is called a multistatement.

Don't forget to press ↵ after any modification. The display as shown below.

```
 1  17
OK?
```

The display appears at the center of the screen.
The LOCATE command is used as follows.

LOCATE X, Y

X indicates the column and Y the line where a character is to be displayed.
See page 196 for details.

3-10 REPEAT PROGRAM EXECUTION

A "routine" is a task within a program that needs to be repeated a specified number of times. For example, we may wish to check a large volume of data to find a specific character or value. Or maybe we need to arrange data in some kind of order. Whatever out requirements, we can have the computer go through all of the present data to compare it against another value, to compare it with neighboring data, or to rearrange everything. The commands included in this section are essential for such applications.
Let's start with a simple program. Check which program area is empty before inputting the program.

See page 151.

The following display appears after this entry is made.

```
P ♥♥23456789  ANGLE  0
   8KB  4720B
Ready P0
_
```

The numbers of program areas that have already been used to store programs or data are replaced with ♥'s. The example display above shows that program areas 0 and 1 contain programs. The 8KB indicates the total RAM capacity. The 4720B indicates that there are 4720 bytes of remaining RAM capacity available for use. Of course, this number would be higher if RAM expansion packs were being used.
A number from 0 through 2 appears after ANGLE to indicate the angle unit (see page 154). This value does not influence ordinary computations, and is always 0 (DEGREE) when the power of the unit is switched ON.
Ready P0 indicates the presently designated program area. In this case, a program can be written in program area 0. Since program areas 0 and 1 are already occupied in this example, we would enter PROG, followed by a value from 2 through 9 and then ↵ .
Now, input the following program.

```
10 CLS
20 FOR A=1 TO 20
30 PRINT CHR$(254);
40 NEXT A
```

Enter the following to display the first line of the program and confirm correct input.

SHIFT EDIT (E) ⏎ ——————→ Specifies the EDIT mode

Advance to the next line by line by pressing ⏎.
After "Ready P0" appears on the display, run the program.
This program displays ▒ 20 times continuously.
Now, let's learn the new commands included in this program.

[1] 10 CLS

CLS is a command that clears the screen and moves the cursor to the upper left corner. It is used to prepare the screen for the next display.

[2]

```
20 FOR A = 1 TO 20
30 PRINT CHR$(254);
40 NEXT A
```

Line 20 and line 40 are actually a single command.

FOR A = 1 TO 20 ..... Sequentially assigns a numerical value from 1 to 20 to variable A.

NEXT A ..... If A < 20, 1 is added to the value of A and a return is made to FOR.

This is called a FOR-NEXT loop. Let's follow the execution procedure.
(1) 1 is assigned to A.
(2) Execute line 30.
3) NEXT A in line 40 checks if A<20.
4) Since A=1, execution returns to line 20 and 2 is assigned to A.
5) Execute line 30.
6) Check if A<20 in line 40.
(7) Since A=2, execution returns to line 20 and 3 is assigned to A.
(8) When the value of A finally reaches 21, the line following the NEXT statement is executed. In this example program there is no line after NEXT, so program execution terminates and "Ready P0" is displayed.

Now let's look at line 30 which is repeated 20 times in this program.

[3] 30 PRINT CHR$(254); See page 245.

Display — CHR$ (254) — same line

All characters and keys are assigned code numbers ranging from 0 through 255, and CHR$ (254) is the function that specifies character code number 254 (▒). (See table on page 327.)
Since ▒ cannot be entered directly by pressing a key, it is specified by the function CHR$ (254). This function is essential for specifying such symbols and graphics. "CHR$ (" is entered using [F] [Z] CHR$(

EXERCISE

- **Prepare a program in which the integers from 1 to a specified number are continuous displayed using FOR-NEXT.**

Hint

```
FOR A = 1 TO N
      ⟆
NEXT A
```

Answer

```
10 CLS
20 INPUT "NUMBER"; N
30 FOR A = 1 TO N
40 PRINT A;
50 NEXT A
60 END
```

Explanation

To provide for the input of an optional numeral in line 20, "NUMBER?" is displayed as a message statement. The entered value is assigned to variable N and the number of repeats is specified in line 30. For example, if the value 15 is entered, line 40 is executed 15 times.
The variable used by FOR-NEXT is displayed in line 40. This program shows that 1 is added to the variable of the FOR-NEXT command each time the loop is repeated.

3-11 SUM TOTAL PROGRAM

The FOR-NEXT loop introduced in the previous section requires some time getting used to, so let's try another program.
This time, let's use program area P4. Use the procedure already outlined to designate the program area. Enter NEW, then ⏎, and we're ready to go.
This program computes the cumulative cost of a series of articles with different unit prices. Subtotals for each article are also provided. The total number of different articles is input at the very beginning of the program.

```
 10 CLEAR
 20 INPUT "NUMBER OF ARTICLES";N
 30 FOR A=1 TO N
 40 INPUT "UNIT PRICE";B
 50 INPUT "QUANTITY";C
 60 PRINT "SUBTOTAL";B*C
 70 D=D+B*C
 80 NEXT A
 90 PRINT "TOTAL";TAB(10);"$";D
100 END
```

The processing in each line of this program is as follows.

10 Clears all variables (assigns 0 to all variables).
20 Requests input of the number of articles (Assigns the number of articles to N).
30 (Unit price and quantity are requested for the number of times specified by N. After each input, the subtotal is displayed and is added to the total.
80)
90 Displays the total amount.
100 Termination

Let's analyze the FOR-NEXT loop from line 30 to line 80 in detail. The task to be performed in the loop from FOR to NEXT is as follows.

(1) The unit price is input and assigned to variable B.
(2) The quantity is input and assigned to variable C.
(3) The unit price is multiplied by the quantity and the subtotal is displayed.
(4) The subtotal is added to the cumulative total.

In line 90 of this program, an easy-to-read display can be obtained.

```
90 PRINT "TOTAL" ; TAB (10) ; " $ " ; D
```

- PRINT "TOTAL" — Displays total.
- TAB (10) — Takes 10 spaces.
- " $ " — Displays $.
- D — Displays total amount.

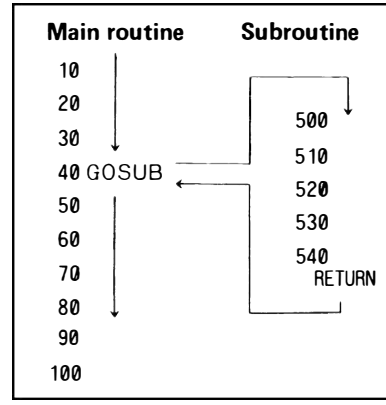
Let's rearrange the program based on a subroutine concept. The fundamentals of the subroutine are shown below.
The command used for this procedure is GOSUB-RETURN.
Line 40 to 70 in the previous program are changed to lines 500 to 540, and GOSUB 500 is inserted in line 40.

```
GOSUB 500
```

Go to the subroutine at line 500.

```
RETURN
```

Return to the command following GOSUB.



Lines 30 to 80 are modified according to the above as follows.

```
30 FOR A = 1 TO N
40 GOSUB 500
50 NEXT A
```

And now our program looks like this:

```
 10 CLEAR
 20 INPUT "NUMBER OF ARTICLES";N
 30 FOR A=1 TO N
 40 GOSUB 500
 50 NEXT A
 60 PRINT "TOTAL";TAB(10);"$";D
 70 END
500 PRINT A;TAB(5);"UNIT PRICE";:INPUT
    B
510 PRINT A;TAB(5);"QUANTITY";:INPUT C
520 PRINT "SUBTOTAL";B*C:BEEP 1
530 D=D+B*C
540 RETURN
```

FOR-NEXT loop. Repeats N times.

Subroutine

Input this program in a new program area. Confirm the difference in execution between this and the previous program.

3-12 CHARACTER VARIABLES

Before getting into the actual storage of large volumes of data, let's first have a look at how character data is handled. As has already been mentioned, variables are roughly divided into two categories: numerical variables (such as A, B, C, A1) to which only numerical values can be assigned, and character variables (such as A$, B$, A1$) to which characters and symbols can be assigned.
Since numbers as well as symbols and alphabetical characters can be assigned to character variables, the difference between the two types of variables may not be clear. It should always be remembered, however, that a numerical variable expresses a quantity, whereas a numerical value in a character variable expresses the character only.

For example:

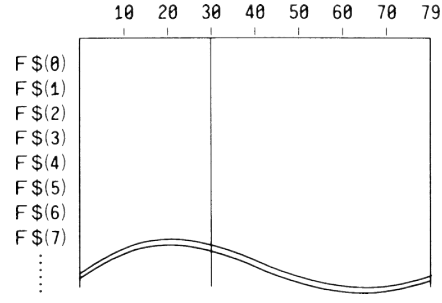
(Numerical value)	**(Character)**
4 + 3 [ENTER] → 7	"4" + "3" [ENTER] → 43

As can be seen on the right, the assigned characters are enclosed in quotation marks the same as messages with the INPUT and PRINT commands. The result of adding two characters is a string that contains the two characters. A numeral treated as a character can be converted to a numerical value using the VAL function (see page 247).
In the following section we will be discussing the use of character variables in arrays. Each element of a character array usually holds up to 16 characters, but can be specified to hold up to 79 characters.

Character arrays are specified as follows.

```
DIM F$(50)*30
    |      |  |
    |      |  +-- Maximum number of characters per variable
    |      +----- Number of data
    +------------ Character variable
```

Memory area is conserved by specifying the maximum number of characters per variable. When 30 is specified as above, the required memory is computed as follows.

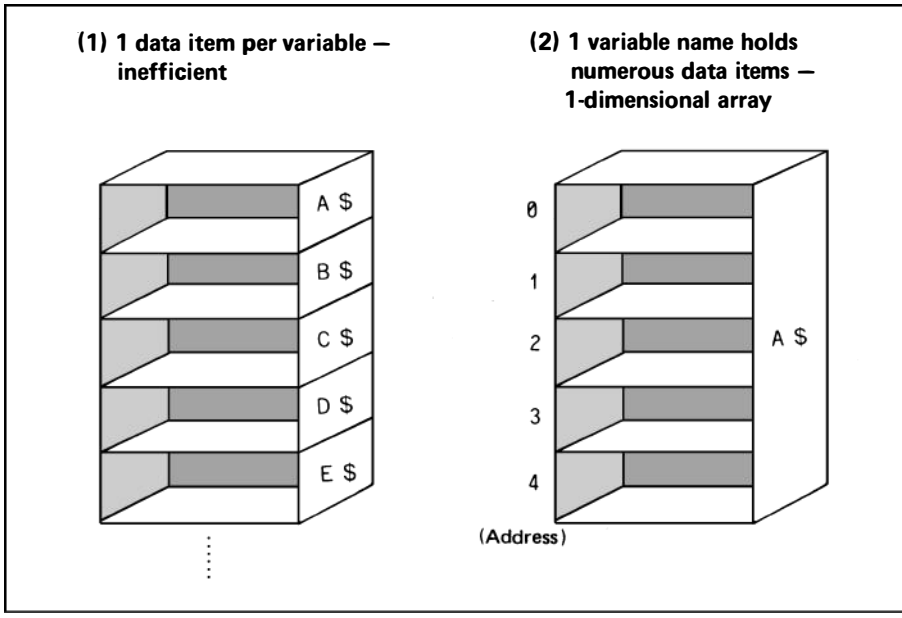


$$50 \times \underbrace{(30+1)}_{\text{Number of characters + 1}} = 1550 \text{ (Bytes)}$$

The less the memory used per variable, the more overall memory space available. With numerical variables, the selection is between single-precision and half-precision variables. A half-precision numerical array requires only half the memory as a single-precision array. This will be explained in the following section.

3-13 WHAT IS A DIMENSION?

The word "dimension" can often be heard when talking about computer programs. A dimension can be thought of as a kind of container or shelves as shown in figure (2) below. Data (such as numerical values or characters) can be stored on these shelves for later retrieval.
In this example, 5 shelves are prepared under 1 variable name. If we store "Smith" in A$(0), "Johnson" in A$(1) and "Foster" in A$(2), we can then recall these data by specifying the variable name and control number. Different data can be included under A$. Without arrays, each piece of data would require its own variable name as shown in figure (1) below. This would make it difficult to keep track of data and would result in inefficient programs.

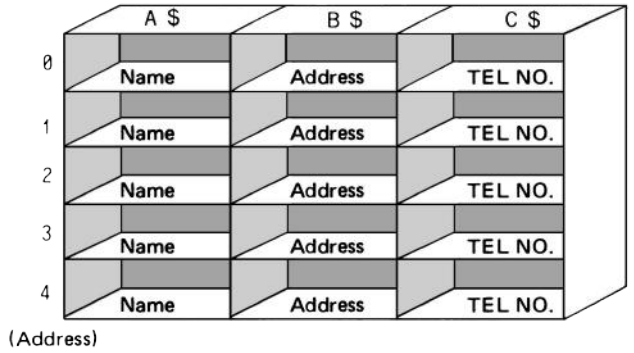


■ 1-Dimensional Arrays

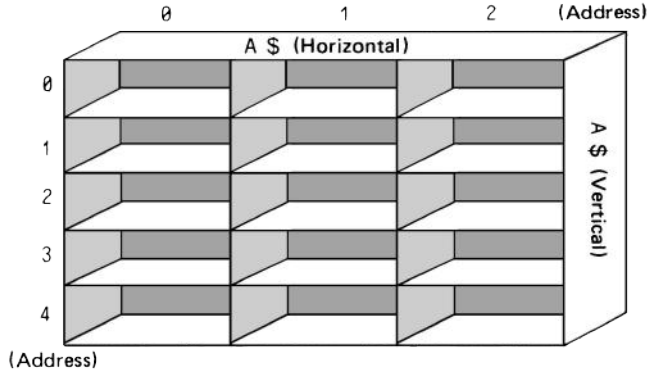
The method shown in figure (2) is known as a 1-dimensional array, and is very handy for inputting large volumes of data. Although only 5 shelves were used for the example, up to 256 (0 — 255) shelves can be reserved per variable name.

Before an array is used, shelf space must be reserved. Attempting to place data on a shelf (in memory) without this preparation will result in an error (UV error).

(3) Directory using 1-dimensional arrays



(4) 2-dimensional array



■ 2-Dimensional Arrays

Based on what we already know about 1-dimensional arrays, we could construct a directory using a series of variables. In figure (3), we see that names are assigned to A$, addresses to B$ and telephone numbers to C$. To find out a person's telephone number, for example, we would input the name and then use the resulting A$ shelf number to find the right shelves in B$ and C$. A specific shelf can be specified by indicating the horizontal variable name and the vertical control number.

Another means to accomplish the same result would be to use what is known as a 2-dimensional array as shown in figure (4). This type of arrangement allows the vertical and horizontal arrangement of data under a single variable name. Up to 256 rows and columns (Ø – 255) can be specified, but, due to memory limitations, a 256 x 256 array is impossible because the memory required for a 2-dimensional array is:

Number of vertical addresses x number of horizontal addresses
x number of required memories per address

A 256 x 256 array that allows 16 characters per variable would require:

256 x 256 x (16 + 1) = 1114112 bytes

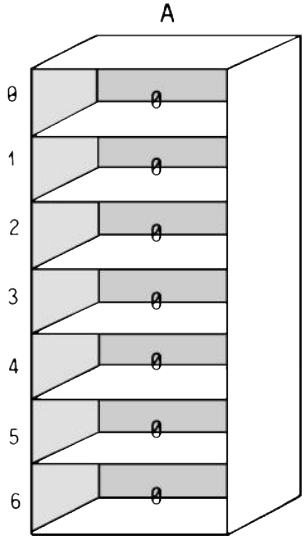
Of course, in this case a memory overflow error would be generated.

3-14 NUMERICAL ARRAY VARIABLES

A numerical array is used when storing numerical values on the shelves described in the preceding section. A variable name (in the case of arrays called an "array variable") such as the A$, B$ and C$ of the previous section is used. The array variables for numerical and string arrays are basically the same for control purposes, but data handling and expression within a program are different.

First let's prepare a program using a numerical array variable to set up a 1-dimensional array like we saw in the previous section. Since this set of shelves will hold numerical values, the array variable must be an alphabetical character. Let's call this array "A", and create a total of 7 shelves from 0 through 6.

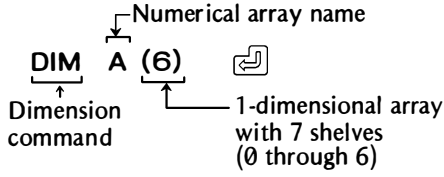
(1) Array variable A configuration and expression



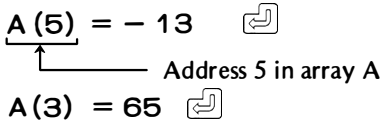
DIM A(6) used to set up shelves

NOTE!
Seven data items are numbered from 0 through 6.

The following is used to prepare shelves 0 through 6 under array variable A.



Specifying the above sets up array A and initializes all arrays in A to 0 (empties all of the shelves). If a DD error is generated when a DIM command is entered, enter CLEAR ↵ and repeat.
Now let's store some data on the shelves.



Now let's confirm that the data are assigned to the specified shelves.

A (5) ENTER ➡ –13 displayed

A (3) ENTER ➡ 65 displayed

The operation outlined above can be included in a program as follows.

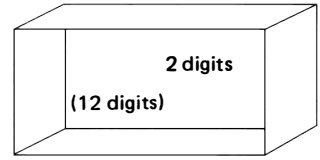
```
 5  CLS
10  CLEAR
20  DIM  A(6)
30  INPUT  "A(5) = "; A(5)
40  INPUT  "A(3) = "; A(3)
50  PRINT  A(5); A(3)
60  END
```

■ **Numerical values assigned to array variables**

Certain restrictions exist concerning the numerical values that are assigned to array variables. The first important thing to remember is that up to a 12-digit mantissa and 2-digit exponent can be assigned to each array variable. Values are displayed, however, up to 10 digits (rounded). Internal computations are conducted with 12 digits.

A numerical array as described above is said to be "single-precision". Often, however, 12 significant digits are not required for computations and 10 digits are not necessary for the display. Therefore, "half-precision" processing can be used in which values are stored only up to 5 digits. As the names imply, the memory space required for a half-precision data item is one half that necessary for single-precision data. The difference between single-precision and half-precision is shown in figure (3).

(2) Array numerical values



Everything past the 12th digit discarded.

(Display)

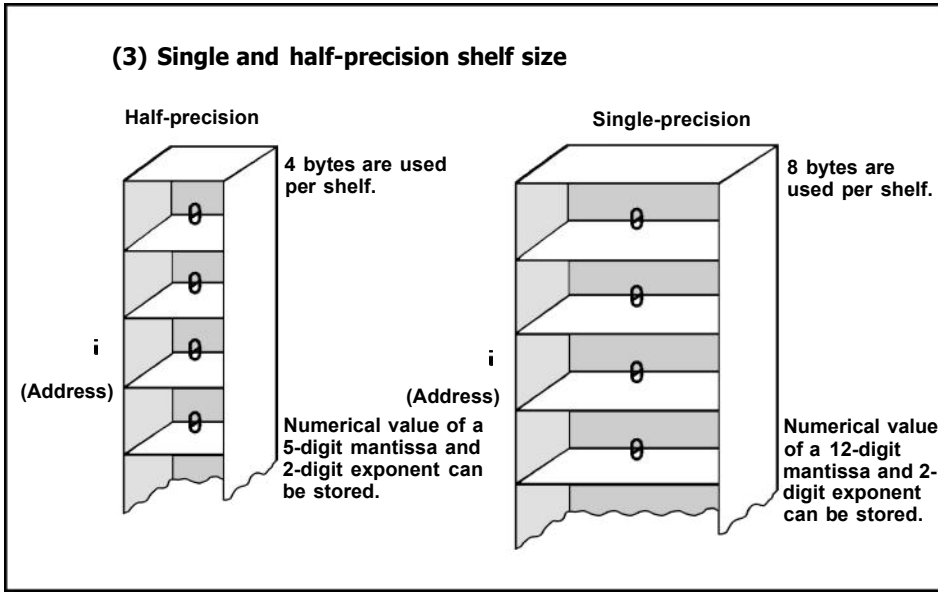
Numerical values up to 100 digits long can be expressed by the PB-770. Even larger digits can be displayed depending upon the program. Once 10 digits are exceeded, however, values appear on the display in exponential form.

12345678909 [ENTER] ➡ 1.234567891 E 10

Mantissa — Exponent

Rounded

Exponential form represents: a number x 10^{10}. In the above example, the display means: $1.234567891 \times 10^{10}$



Single-precision dimension is expressed as follows.

A(i) ..... Single-precision numerical array

A half-precision dimension is expressed using an exclamation mark after the variable.

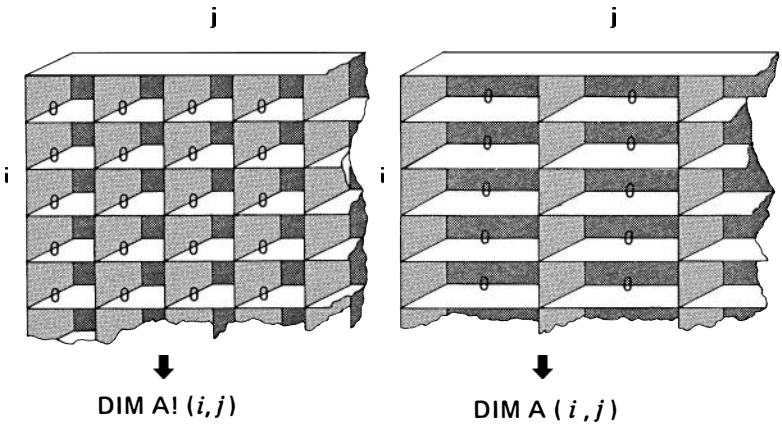
A!(i) .....half-precision numerical array

■ Single-precision and half-precision in a 2-dimensional array

The precisions of 2-dimensional arrays can be regarded the same as those for 1-dimensional arrays. An example is shown below in figure (4).

DIM A! (*i* , *j*) . . . . . Half-precision
DIM A (*i* , *j*) . . . . . Single-precision

(4) Half and single-precision 2-dimensional arrays

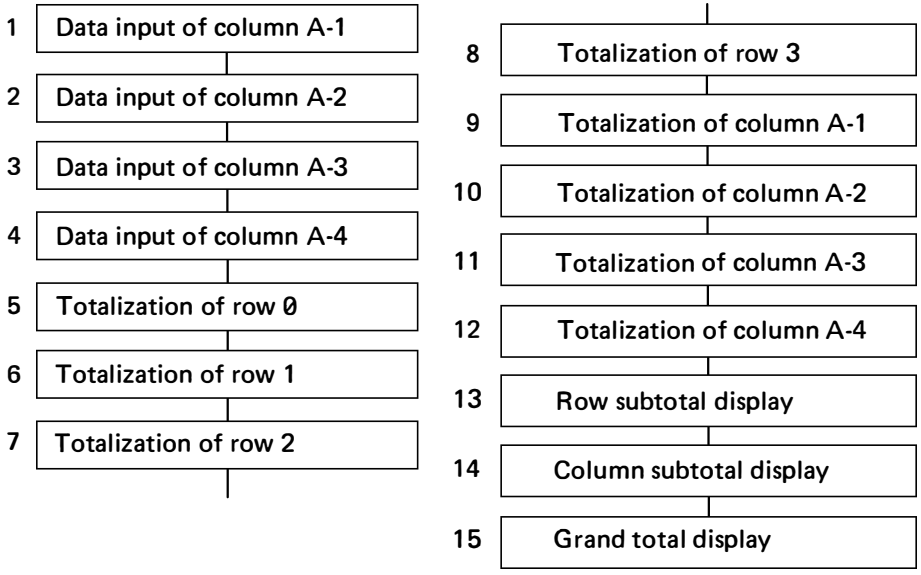


3-15 NUMERICAL ARRAY PROGRAMMING

Let's prepare a program for a totalization table. Column and row subtotals are to be computed, and a grand total is obtained at the end. The table is 4 x 4, so 16 pieces of data are used.

	A-1	A-2	A-3	A-4	Subtotal
0	25	17	3	67	
1	19	20	11	58	
2	32	15	26	55	
3	36	28	29	40	
Subtotal					

Since numerical data are used, let's prepare a program with a 1-dimensional numerical array. Let's select A as the variable name. Now, let's prepare the program according to the following flow chart.



Let's prepare a program to input each column of data as shown in 1 to 4 of the flow chart.

```
10  CLEAR
20  DIM  A!(15)
30  FOR  I=0  TO  3
40  INPUT  A!(I)
50  NEXT  I
```

FOR-NEXT loop

In this program data is entered from A!(0) through A!(3) (column A-1), but how do you enter data to columns A-2 through A-4 without changing the variable name? The following shows what we are trying to accomplish.

	→ J				Subtotal
	A!(0)	A!(4)	A!(8)	A!(12)	B!(0)
↓	A!(1)	A!(5)	A!(9)	A!(13)	B!(1)
I	A!(2)	A!(6)	A!(10)	A!(14)	B!(2)
	A!(3)	A!(7)	A!(11)	A!(15)	B!(3)
Subtotal	C!(0)	C!(1)	C!(2)	C!(3)	D

If we look at the relationship from column to column, we can find the following pattern.

A!(x) A!(x+4) A!(x+8) A!(x+12)

Now all we need is a routine that will add 4 to the loop counter each time we want to move to the next row. This can be accomplished using the nested loop shown in lines 40 through 90 below.

```
10  REM  INPUT
20  CLEAR
30  DIM  A!(15)
40  FOR  J=0  TO  3
50  FOR  I=0  TO  3
60  PRINT  "A-";J+1;"(";I;")";
70  INPUT  A!(J*4+I)
80  NEXT  I
90  NEXT  J
```

Column data input

Data input by shifting sequentially from column A-1.

Let's see how this works. The first value of J and I is 0. J will retain the value of 0 for the next three passes of I. Line 60 results in a display of "A–1 (0)" because both J and I equal 0. Line 70 waits for an input for A! (0) since 0 * 4 + 0 = 0.
On the second pass of I, J still equals 0 but I now equals 1. Therefore, line 60 displays "A–1 (1)" and line 70 waits for an input for A!(1) since 0 * 4 + 1 = 1.
Let's take a look farther down at A!(14). In this case, J=3 and I=2. Line 60 displays "A–4 (2)" and line 70 waits for an input for A!(14) because 3 * 4 + 2 = 14.

This completes the data input program. Data is sequentially entered with the display of "A–Column No. (Vertical No.)? " Data is stored in array variable A! ( ) in line 70.
Now, let's prepare a program for the row subtotal.

```
 95  DIM B!(3)
100  FOR  I=0  TO 3
110  FOR  J=0  TO 3
120  B!(I)=B!(I)+A!(J*4+I)
130  NEXT J
140  PRINT  "B-"; I; TAB(5); B!(I)
150  INPUT "OK"; F$
160  NEXT  I
```

Row computation and display.

Computes and displays subtotal by shifting vertically 4 times.

Data is stored to array variables B!(0)–B!(3) in line 120, and is displayed in line 140. To prevent display of the subtotal of B!(0)–B!(3) from scrolling, the display stops in line 150, and the next display is made by entering ⏎.
Now, prepare a program for the column subtotal.

```
170  DIM C!(3)
180  FOR  J=0  TO 3
190  FOR  I=0  TO 3
200  C!(J)=C!(J)+A!(J*4+I)
210  NEXT  I
220  PRINT "C-"; J; TAB(5); C!(J)
230  INPUT "OK"; F$
240  NEXT J
```

Column computation and display

Computes and displays subtotal by shifting horizontally four times

This is almost the same as the row subtotal method using nested FOR-NEXT loop with a different sequence of variables (loop control variables) I and J.
Now, let's compute the total.

```
250 REM TOTAL
260 FOR  I=0 TO 3
270 D = D+C!(I)
280 NEXT  I
290 PRINT "TOTAL=";D
300 END
```

Programming With A 2-Dimensional Array

The handling of each shelf where data is to be stored is complicated and difficult to understand when a 1-dimensional array is used as previously explained. Let's prepare a program using a 2-dimensional array which is much more convenient for handling column and row data.

1 Initialization

```
10 ERASE A!          .... Clears the array of variable A.
20  CLS              .... Clears the screen.
30 N=4               .... The number of column or row items.
40 DIM A!(N,N)       .... Dimension declaration (The declaration of half-
                          precision numerical array).
```

The number of column and row items can be changed by changing the value of N in line 30.

2 Data Input

```
 50 FOR I=0 TO N-1
 60 FOR J=0 TO N-1
 70 PRINT I;"-";J;
 80 INPUT A!(I,J)
 90 NEXT J
100 NEXT I
```

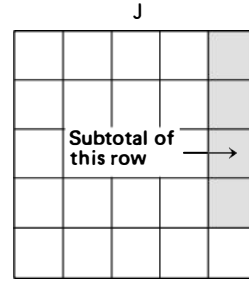
→ J

I ↓					Row total
	28	39	12	54	133
	53	29	55	30	167
	28	17	80	53	178
	60	31	70	44	205
Column total	169	116	217	181	683

N−1 is used in lines 50 and 60 because row and column totals are not required for data input. Note that data are entered in rows and not vertically as with the 1-dimensional array.

3 Row Subtotal

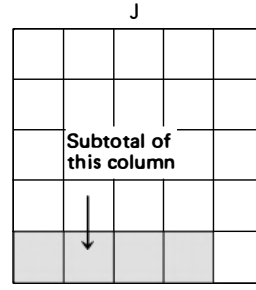
```
110 FOR I=0 TO N-1
120 FOR J=0 TO N-1
130 A!(I,N)=A!(I,N)+A!(I,J)
140 NEXT J
150 NEXT I
```



Row subtotals are computed four times (when N=4). Manually go through this routine to confirm that all values of A!( ) are accounted for.

4 Column Subtotal

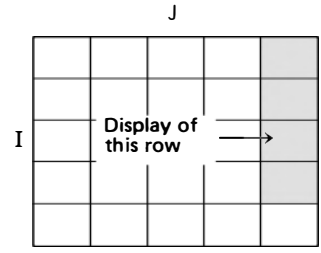
```
160 FOR J=0 TO N
170 FOR I=0 TO N-1
180 A!(N,J)=A!(N,J)+A!(I,J)
190 NEXT I
200 NEXT J
```



Column subtotals are computed four times (when N=4). Again, manually go through this routine to confirm that all values of A!( ) are accounted for.

5 Row Subtotal Display

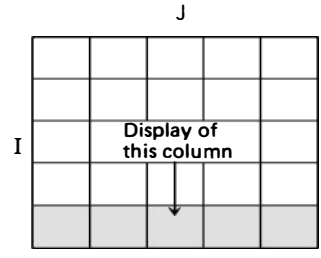
```
210 FOR I=0 TO N-1
220 PRINT A!(I,N);
230 NEXT I
240 STOP
```



The program stops once and confirms each row subtotal. Then, execution proceeds to the next line after SHIFT CONT W ↵ is entered.

6 Column Subtotal Display

```
250 FOR J=0 TO N
260 PRINT A!(N,J);
270 NEXT J
280 END
```



This routine displays the sum of all of the column subtotals. Since the subtotal of the column at the far right is actually the total of the row subtotals, the result is a display of the sum of the entire table.

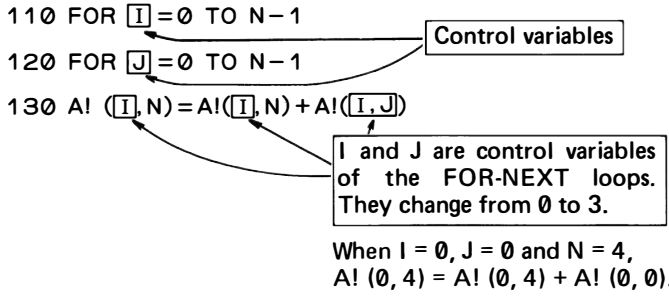
The most difficult point of this program was the double FOR-NEXT loops (commonly called "nesting"). But after using this technique a few times, its value will soon become evident. Just remember that the FOR-NEXT loop automatically increments the value of control variable with each execution.

7 Program Execution

When you run the program, "0–0? " is displayed which requests data entry. Enter Data ↵, then data entry for the next row is requested by "0–1?".
When all data input has been performed the row subtotals and "STOP P0–240" are displayed and the program stops.
Next, the column subtotals and total are displayed by entering SHIFT CONT W ↵ and the program is terminated. Try several different display formats.

■ **Control variables**

In the routine for the row subtotals, the values of control variables change as follows.



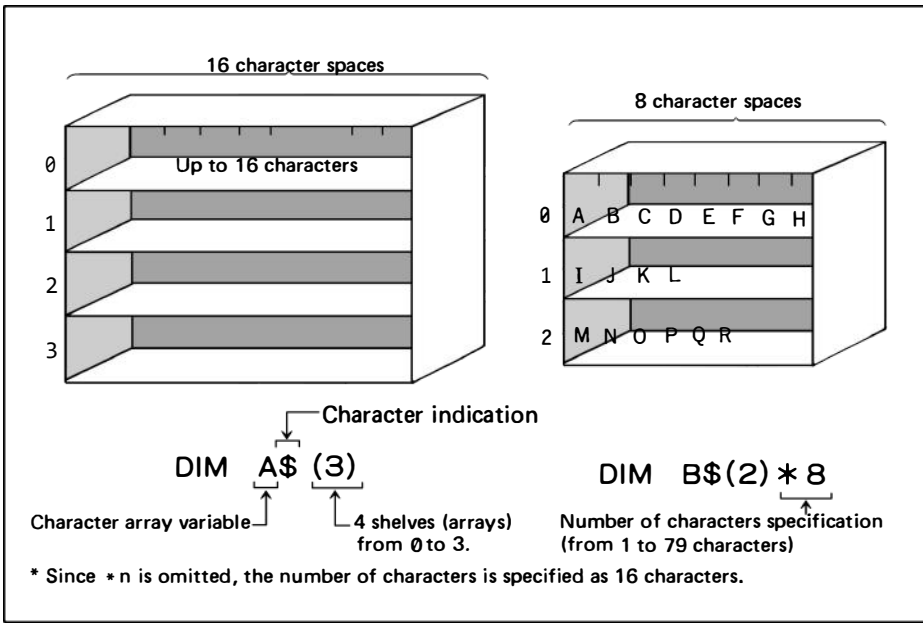
When I = 0, J = 0 and N = 4,
A! (0, 4) = A! (0, 4) + A! (0, 0).

3-16 CHARACTER ARRAY VARIABLES

Characters can also be assigned to 1-dimensional and 2-dimensional arrays. With numerical array variables, single and half-precision can be specified for efficient use of memory, as described before. Character array variables can also be used with the string length specified.
Storing a string that is shorter than 6 characters in an array specified for more wastes memory. Attempting to store a string that is longer than the specified array capacity will generate an error. Therefore, it is important to determine the string length of the array being defined.
The number of characters per character array variable is specified as follows.

DIM A$(i) ∗ n (1 ≦n< 79)

* If ∗n is omitted, the number of characters is specified to 16 characters.



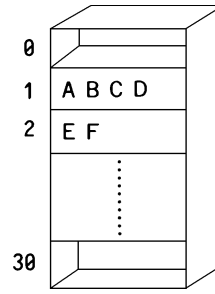
* Since ∗n is omitted, the number of characters is specified as 16 characters.

Let's enter characters in an array variable. Perform the following operations.

```
CLEAR ↵
DIM  H$(30)*5 ↵
H$(1) = "ABCD" ↵
H$(2) = "EF" ↵
```

When you execute these entries, the characters are stored in the array variable as shown in the figure on the right.
To recall these, perform the following operations.

```
H$(1) ENTER  ➡  ABCD
H$(2) ENTER  ➡  EF
```



■ **DIM statement error**
Attempting to enter a string that is longer than the size specified by a DIM statement results in an error.

ST error

At this time, reenter the string keeping it within the defined range.
Attempting to define an array that has the same name as a presently defined array will also result in an error (DD error). For example, this error would be generated if the statement

DIM A$(20) were input followed by

DIM A$(30)

When this happens, erase A$( ) using the ERASE command.

ERASE A$ ↵

It should be noted, however, that the ERASE and CLEAR commands erase data.

■ **String Array Programming**

Exercise

Problem

Prepare an array program with 3 arrays in which up to 10 characters can be stored respectively. Characters are to be entered by character codes. To stop input in the middle, 0 is to be entered. When the entry for 3 arrays has been completed, all characters are displayed.

Hint

Since initialization is with up to 10 characters and 3 arrays,

```
10 CLEAR
20 DIM N$(2)*10
```

The data input routine comes next. Since there are three arrays, prepare three FOR-NEXT loops and use another loop specified by a GOTO statement to read 10 characters. Include an INPUT statement in the loop to read character codes.
The basic configuration of the data input program is as follows.

(1) Set the counter to 0 for the number of characters per variable . . . . . B = 0
(2) Input a character code . . . . . INPUT N
(3) Add 1 to the counter for the number of characters . . . . . B = B + 1
(4) If the counter exceeds 10, input character codes to next array. . . . . IF B = 10 THEN ~
(5) Convert a code number to a character and store it to an array variable . . . . . N$(I) = N$(I) + CHR$(N)
(6) Return to (2) . . . . . GOTO ~

The routine for displaying the result comes next. Provide a display of N$(0)—N$(2) on the same line.

```
100 FOR I=0 TO 2
110 PRINT N$(I);"  ";
120 NEXT I
```

0 to 2 is sequentially entered to I.

Answer

```
 10 CLEAR :CLS
 20 DIM N$(2)*10
 30 FOR I=0 TO 2:B=0
 40 PRINT "N$(";I;")";:INPUT "  No.";N
 45 IF N>255 THEN 40
 50 IF N=0 THEN 90
 60 B=B+1:IF B=10 THEN BEEP :GOTO 90
 70 N$(I)=N$(I)+CHR$(N)
 80 GOTO 40
 90 NEXT I
100 FOR I=0 TO 2
110 PRINT N$(I);" ";
120 NEXT I
130 END
```

When you execute this program, "N$(␣0)␣␣No.?␣" is displayed. Now enter the character code. The numeral enclosed by parentheses of "N$(I)" is changed from 0 to 1 to 2 by entering 0 ↵.
Now, enter the following codes and see what appears.

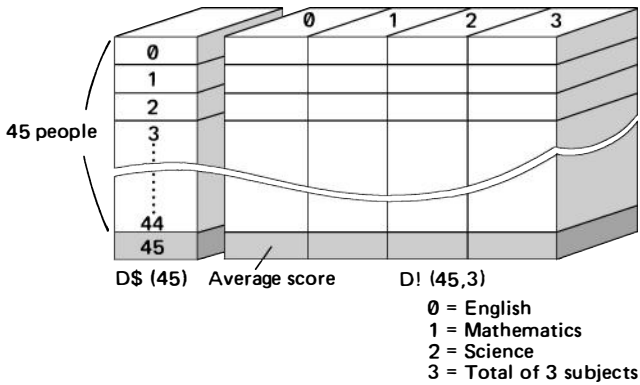
67 ↵ 65 ↵ 83 ↵ 73 ↵ 79 ↵ 0 ↵ 67 ↵
79 ↵ 77 ↵ 80 ↵ 85 ↵ 84 ↵ 69 ↵
82 ↵ 0 ↵ 80 ↵ 66 ↵ 55 ↵ 48 ↵ 48 ↵ 0 ↵

3-17 COMBINATION OF STRING ARRAYS AND NUMERICAL ARRAYS

In most cases when data is processed by preparing a table, a combination of string and numerical arrays is used. In this case, characters and numerical values must be handled at the same time as one data group.

For example, in regard to a name and score, or the names of articles, number of articles and an amount, characters and numerical values must be recalled at the same time.

Let's prepare a result processing program as an example. First, a 1-dimensional character array to store names is required. Then a 2-dimensional numerical array is required in which the total score for three subjects (English, mathematics, and science) corresponding to the name is stored. The following model can be assumed based on the items mentioned above.



The data to be entered is as follows.

Name	English	Mathematics	Science	Total score
A. Y	50	60	75	
K. K	83	71	70	
S. O	60	63	40	
Average score				

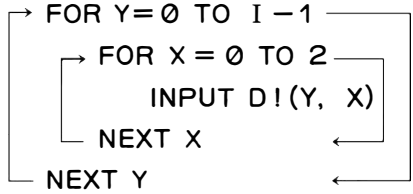
1 Name Input

```
10 ERASE D$,D!: CLS
20 DIM D$(45)*10,D!(45,3)
30 I=0
40 INPUT "NAME ";D$(I)
50 IF D$(I)="END" THEN 80
60 I=I+1:IF I=45 THEN 80
70 GOTO 40
```

Names are stored in D$(0) to D$(44) and the average score is stored in D$(45). The execution is exited from the loop by inputting "END". Of course, lines 40 to 70 provide the loop for name input.

2 Numerical Value Input

The following is a routine that enters numerical value into a 2-dimensional array. Line 90 displays a student's name in D$(Y), and line 120 inputs the English score in D!(0,0) (when X=0). Then line 130 adds the score to D!(0,3). D!(Y,3) is the subtotal of the row since only three subjects are being handled here. The number of elements can be expanded by changing the 3 of D!(Y,3) to "1 + the number of elements". X,Y of D!(Y,X) are determined by the frequency of the FOR-NEXT loop.



The basic input format is shown below.

```
 80 FOR Y=0 TO I-1
 90 PRINT D$(Y)
100 FOR X=0 TO 2
110 PRINT X
120 INPUT " ";D!(Y,X)
130 D!(Y,3)=D!(Y,3)+D!(Y,X)
140 NEXT X
150 NEXT Y
```

→ X

Score table

↓ Y	English (0)	Mathematics (1)	Science (2)	Total

3 Average Score For Each Subject

No matter how many names are input (counted by I), D!(45,X) is specified to store the average score for each subject. In line 180 the cumulative points for each subject are assigned to D!(45,X). In line 200 the average is assigned to D, and in line 210 the decimal part of D is rounded off to one place and then reassigned to D!(45,X). INT is the integer function in which values below the decimal point are discarded.
This function is used because of characteristics of test result processing. If used for other purposes, line 210 can be changed.

```
160 FOR X=0 TO 3
170 FOR Y=0 TO I-1
180 D!(45,X)=D!(45,X)+D!(Y,X)
190 NEXT Y
200 D=D!(45,X)/I
210 D!(45,X )=INT(D*10+0.5)/10
220 NEXT X
```

4 Total Score Display For Each Name

The total score for each name is displayed. After you press ↵ in line 260, the program proceeds to the next name. The total score for a name is processed by loop control variable Y. Therefore, the total always corresponds to the proper name.

Name ..... D$(Y)
Total score ..... D!(Y,3)

```
230 FOR Y=0 TO I-1
240 PRINT D$(Y);" T=";
250 PRINT D!(Y,3)
260 K$=INKEY$:IF K$="" THEN 260
270 NEXT Y
```

5 Average Score Display

The display of the average score for each subject is performed by sequentially displaying data D!(45,X) from X=0 to X=3.

```
280 FOR X=0 TO 3
290 PRINT "AVE.=";
300 PRINT D!(45,X)
310 K$=INKEY$:IF K$="" THEN 310
320 NEXT X
330 END
```

Execute this program and enter data.

The display screen of execution result is as follows.

```
A.Y T=  185
K.K T=  224
S.O T=  163
```

Total score for a name is displayed sequentially each time the ↵ key is pressed.

```
↵ AVE. =  64.3
↵ AVE. =  64.7
↵ AVE. =  61.7
↵ AVE. = 190.7
```

— Average score of English(0)
— Average score of Mathematics(1)
— Average score of Science (2)
— Cumulative average of each person's total score

3-18 STATISTICAL FUNCTIONS

Statistical computation capabilities are essential to business and engineering for analyzing data and making projections. The PB-770 features all of the essential statistical functions listed below, so troublesome computations are simplified while correlation coefficients and estimated values, etc. can be quickly determined.

* STAT LIST [↵] displays the names and values of the basic statistics (indicated by ★ below). To suspend display press [↵]. Pressing [↵] again will resume the display. Entering STAT LLIST [↵] will output the basic statistics to the printer.

Format		Function
CNT★	n	Number of data items processed
SUMX★	Σx	Sum of x data
SUMY★	Σy	Sum of y data
SUMXY★	Σxy	Sum of products of x data and y data
SUMX2★	Σx^2	Sum of squares of x data
SUMY2★	Σy^2	Sum of squares of y data
MEANX	$\overline{x}$	Mean of x data
MEANY	$\overline{y}$	Mean of y data
SDX	$x\sigma_{n-1}$	Sample standard deviation of x data $\sqrt{\frac{n\Sigma x^2-(\Sigma x)^2}{n(n-1)}}$
SDY	$y\sigma_{n-1}$	Sample standard deviation of y data $\sqrt{\frac{n\Sigma y^2-(\Sigma y)^2}{n(n-1)}}$
SDXN	$x\sigma_n$	Population standard deviation of x data $\sqrt{\frac{n\Sigma x^2-(\Sigma x)^2}{n^2}}$
SDYN	$y\sigma_n$	Population standard deviation of y data $\sqrt{\frac{n\Sigma y^2-(\Sigma y)^2}{n^2}}$
LRA	A	Linear regression constant term $\frac{\Sigma y-\mathrm{LRB}\cdot\Sigma x}{n}$
LRB	B	Linear regression coefficient $\frac{n\cdot\Sigma xy-\Sigma x\cdot\Sigma y}{n\cdot\Sigma x^2-(\Sigma x)^2}$
COR	r	Correlation coefficient $\frac{n\Sigma xy-\Sigma x\cdot\Sigma y}{\sqrt{\{n\Sigma x^2-(\Sigma x)^2\}\{n\Sigma y^2-(\Sigma y)^2\}}}$
EOX (numerical expression)	$\hat{x}$	Estimated value (value of x estimated from y) $\mathrm{EOX}(y_n)=\frac{y_n-\mathrm{LRA}}{\mathrm{LRB}}$
EOY (numerical expression)	$\hat{y}$	Estimated value (value of y estimated from x) $\mathrm{EOY}(x_n)=\mathrm{LRA}+x_n\cdot\mathrm{LRB}$

STAT CLEAR [=] should be entered to clear the statistical memory area before new data is entered.

■ Statistical Data Input

- Single variable
  Individual data input . . . . . STAT data [=]
  Multiple input of same data . . . . . STAT data [SHIFT] [,] frequency [=]

- Paired variable
  Individual data input . . . . . STAT x data [,] y data [=]
  Multiple input of same data . . . . . STAT x data [,] y data [SHIFT] [,] frequency [=]

Data is input using the [=] key, but such results as standard deviation are obtained using the [ENTER] key. Incorrect operation will result in an SN error being displayed.

Example

The table represents the shipments of articles x and y over a period of 5 days. Calculate the standard deviation and determine the variance in the shipments.

Article \ Date	4	5	6	7	8
x	2	2	5	8	8
y	1	5	5	5	9

Operation:

STAT CLEAR [=]

STAT 2 [,] 1 [=] STAT 2 [,] 5 [=] STAT 5 [,] 5 [=]

STAT 8 [,] 5 [=] STAT 8 [,] 9 [=]

STAT LIST [=] (Basic statistics will be displayed automatically.)

CNT 5 ····· Number of data items
SUMX 25 ····· Sum of x data
SUMY 25 ····· Sum of y data
SUMXY 149 ····· Sum of products of x data and y data
SUMX2 161 ····· Sum of squares of x data
SUMY2 157 ····· Sum of squares of y data

Ready P0

MEANX [ENTER]	5	Mean of x data
MEANY [ENTER]	5	Mean of y data
SDXN [ENTER]	2.683281573	Population standard deviation of x data
SDYN [ENTER]	2.529822128	Population standard deviation of y data

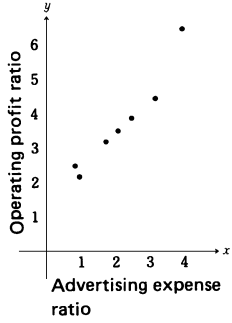
According to these statistical results, the standard deviation of article x is larger than that of y though the sums and means are the same. Therefore, it can be said that the variance in the shipment is greater for x.

Use the following data for regression computation and determine the correlation coefficient and estimated value.

Example

The following table shows last year's advertising expense ratio (advertising expenses/operating expenses x 100) and operating profit ratio (operating profits/sales x 100) for chain of 7 supermarkets.

	1	2	3	4	5	6	7
Advertising expense ratio (x)	0.8	2.1	2.5	1.8	3.1	4.0	1.0
Operating profit ratio (y)	2.5	3.4	3.7	3.2	4.3	6.3	2.3



Use the table to produce a scatter diagram. Looking at the scatter diagram, it can be said that profit was directly proportional to the amount spent for advertising. A line connecting the plots (dots) in the diagram is called a regression curve, and, since it is almost a straight line in this example, it is called linear regression. Regression curves are logarithmic, exponential and power curves, and the selection of the curve depends upon the relationship between the x and y data. It should be noted that the range of the correlation coefficient (r) is $-1 \leqq r \leqq 1$. The correlation is positive when $0 \leqq r \leqq 1$, negative when $-1 \leqq r < 0$, and no correlation exists when r=0.

Now let's input the data for the 7 stores.

Operation: STAT CLEAR ⏎

STAT0.8▶2.5⏎STAT2.1▶3.4⏎STAT2.5▶3.7⏎
STAT1.8▶3.2⏎STAT3.1▶4.3⏎STAT4.0▶6.3⏎
STAT1.0▶2.3⏎

LRA ENTER	1.174221646	..... Linear regression constant term (A)
LRB ENTER	1.142512973	..... Linear regression coefficient (B)
COR ENTER	0.9628252383	..... Correlation coefficient (r)

The correlation coefficient (r) indicates that x and y have a positive correlation. Now let's calculate how much advertising expense ratio is required for an operating profit ratio of 5.7% and how much operating profit ratio can be expected from an advertising expense ratio of 4.5%.

EOX5.7 ENTER	3.961248986
EOY4.5 ENTER	6.315530022

These results tell us that an advertising expense ratio of 3.96% is required for an operating profit ratio of 5.7%, while an advertising expense ratio of 4.5% can be expected to produce an operating profit ratio of 6.32%.

• Logarithmic, Exponential and Power Regression

Let's apply the various types of regression to the data in the table below.

Year (x)	Results (y)
'79	5,810
'80	5,637
'81	6,736
'82	7,938
'83	8,169

• Logarithmic regression

The regression formula is $y = A + B \cdot \ln x$

The logarithm of x is input for x data, and y data is input as it is.

$\Sigma \ln x$, $\Sigma (\ln x)^2$ and $\Sigma \ln xy$ are obtained for Σx, Σx^2 and Σxy respectively.

Operation:

STAT CLEAR [=]

STAT LOG 54 [,] 5810 [=] STAT LOG 55 [,] 5637 [=]

STAT LOG 56 [,] 6736 [=] STAT LOG 57 [,] 7938 [=]

STAT LOG 58 [,] 8169 [=]

LRA [ENTER]	−151086.8602 Regression constant term (A)
LRB [ENTER]	39240.6409 Regression coefficient (B)
COR [ENTER]	0.9461867989 Correlation coefficient (r)
COR [SHIFT] [⊕] 2 [ENTER]	0.8952694585 Decision coefficient (r^2)

• Exponential regression

The regression formula is $y = A \cdot e^{B \cdot x}$ ($\ln y = \ln A + B \cdot x$)

The logarithm of y is input for y data, and x data is input as it is.

$\ln A$, $\Sigma \ln y$ and $\Sigma x \cdot \ln y$ are obtained for A, SUMY and SUMY2 respectively.

Operation: STAT CLEAR ↵
STAT 54 , LOG 5810 ↵ STAT 55 , LOG 5637 ↵
STAT 56 , LOG 6736 ↵ STAT 57 , LOG 7938 ↵
STAT 58 , LOG 8169 ↵

EXP LRA	ENTER	21.93154256 Regression constant term (A)
LRB	ENTER	0.102384121 Regression coefficient (B)
COR	ENTER	0.9442661562 Correlation coefficient (r)

● Power regression

The regression formula is $y = A \cdot x^B$ ($\ln y = \ln A + B \cdot \ln x$).
The logarithms of x and y are input for data x and y respectively.
$\ln A$, $\Sigma \ln x$, $\Sigma (\ln x)^2$, $\Sigma \ln y$, $\Sigma (\ln y)^2$ and $\Sigma (\ln x \cdot \ln y)$ are obtained for A, Σx, Σx^2, Σy, Σy^2 and Σxy respectively.

Operation STAT CLEAR ↵
STAT LOG 54 , LOG 5810 ↵ STAT LOG 55 , LOG 5637 ↵
STAT LOG 56 , LOG 6736 ↵ STAT LOG 57 , LOG 7938 ↵
STAT LOG 58 , LOG 8169 ↵

EXP LRA	ENTER	6.651154824E−07
LRB	ENTER	5.725355325
COR	ENTER	0.9433168782

3-19 USING GRAPHIC CHARACTERS

The characters shown on the Character Code Table on page 327 can be used by the PB-770.

(1) CHR$ function

The word "NAME" can be displayed on the screen using the following command.

```
PRINT "NAME" ↵
```

The CHR$ function can also be used to produce the same display.

```
PRINT CHR$(78);CHR$(65);CHR$(77);CHR$(69) ↵
```

As can be seen, specifying a character code for the respective letters caused them to be displayed on the screen.
The other characters can be used in the same manner.
The following table illustrates some of the codes.

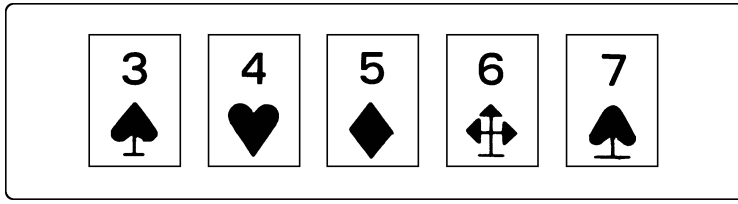
Code Number	Display	Code Number	Display
32, 160		164	､
161	｡	165	･
162	｢	176	ｰ
163	｣		

(2) Graphic symbols

The following program will produce a display of 5 playing cards.

```
10  CLS
20 FOR I=0 TO 4
30 LOCATE I*3,1:PRINT I+3
40 LOCATE I*3+1,2:PRINT CHR$(232+I MO
   D 4)
50 DRAW(I*24+4,4)-(I*24+20,4)-(I*24+2
   0,26)-(I*24+4,26)-(I*24+4,4)
60 NEXT I
70 IF INKEY$="" THEN 70
80 END
```

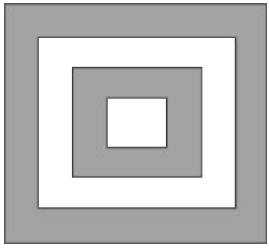
Execution Result



3-20 DISPLAYING PATTERNS

PB-770 Only

The PB-770 is capable of displaying patterns which cannot be found in the Character Code Table. Let's try to display the pattern shown below.



Display format

```
AB$="FF18DB5A5ADB18FF" ↵
PRINT $AB$ ↵
```

After the above noted key operation, a pattern should appear on the screen.

(1) Display process

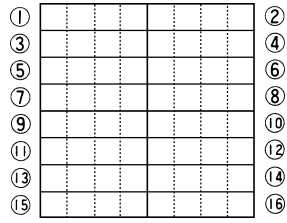
Displays are produced by attaching a $ before a character variable (in this case AB$). This same process can also be used in a program.

The string that is assigned to AB$ is made up of hexadecimal values (the hexadecimal number system uses numbers 0 through 9 plus alphabetical characters from A through F). This same configuration can be used to produce a variety of user generated graphics.

(2) Pattern configuration

Graphics are produced using the hexadecimal pattern shown below. The pattern is an 8 x 8 grid, and each of the 64 positions of the grid represent a dot on the screen.

As can also be noted, dots are arranged into 16 groups of 4 dots each. These 16 groups are numbered according to the hexadecimal system mentioned above.



(3) Dot designation

Dots are specified using the hexadecimal values. The following shows the binary equivalents of the 16 hexadecimal values.

■ **Hexadecimal→Binary**

0→0 0 0 0	4→0 1 0 0	8→1 0 0 0	C→1 1 0 0
1→0 0 0 1	5→0 1 0 1	9→1 0 0 1	D→1 1 0 1
2→0 0 1 0	6→0 1 1 0	A→1 0 1 0	E→1 1 1 0
3→0 0 1 1	7→0 1 1 1	B→1 0 1 1	F→1 1 1 1

The above table shows that the hexadecimal values can be expressed as 4-digit binary values. On the screen a binary "1" indicates that a dot is displayed, while a "0" indicates that the dot is not displayed.
It is important to note, however, that the PB-770 counts dots from left to right, from 0 through 3. It should also be noted that this counting system is the opposite that of the binary number system.
The following table illustrates the 16 patterns possible with the 4-dot groups. Specifying the proper hexadecimal value will produce the corresponding pattern on the screen.

Pattern	Hexadecimal value
□□□□	0
■□□□	1
□■□□	2
■■□□	3
□□■□	4
■□■□	5
□■■□	6
■■■□	7

Pattern	Hexadecimal value
□□□■	8
■□□■	9
□■□■	A
■■□■	B
□□■■	C
■□■■	D
□■■■	E
■■■■	F

(4) Cautions

The following points should be noted when defining patterns.

1. Failing to include a "$" directly before the character variable will cause the string in the variable to be produced on the screen.
2. Pattern display cannot be accomplished if the assigned string is longer than 16 characters, if it is shorter than 16 characters, or when the LPRINT command is used.

(5) Using character definition to display " αβγ "

The following program is included to help you get some practice with character definition. When executed, the program displays " αβγ ".

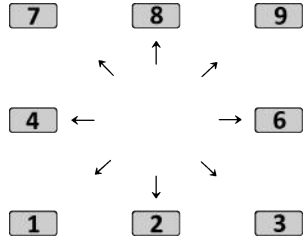
```
10 A1$="000442A2111 9E600"
20 A2$="C12222A12222E320"
30 A3$="002241C040606040"
40 PRINT $A1$;$A2$;$A3$;
```

(6) Pattern utility

It is often difficult to convert a hand-drawn pattern to hexadecimal format for input to the computer. The following program makes it possible to develop a pattern on the screen of the PB-770 itself and then automatically convert it to hexadecimal.

Usage

First, input the utility program on pages 98 and 99 into the PB-770. After confirming that input was performed correctly, enter RUN [↵].
An 8 x 8 grid will appear on the left side of the screen. The blinking dot at the upper left position of the grid represents the pattern cursor. The word "ON" or "OFF" on the right of the screen indicates the status of the pattern cursor. ON indicates that a dot is present at the pattern cursor position, while OFF indicates that a dot is not present. The [·] key is used to switch the status of the pattern cursor. Movement of the pattern cursor is controlled by the following keys.



Movement of the cursor is confirmed by a beep each time one of the movement keys is pressed. Try moving the pattern cursor around the grid and turning some dots ON and OFF. Try making a pattern of your choice.

Once a pattern is formed, press the CLS key. Shortly, the hexadecimal value that represents the pattern formed above will be displayed. Write down this value and assign it to a registered character variable in the PRINT$ statement to display the original pattern that you created above.

```
 10  CLS
 20 X=2:Y=2:F=0
 30 FOR I=0 TO 32 STEP 4
 40 DRAW(I,0)-(I,31)
 50 IF I=0 THEN 70
 60 DRAW(0,I)-(31,I)
 70 NEXT I
 80 LOCATE 11,3:PRINT "OFF";
 90 BEEP
100 K$=INKEY$:L=ASC(K$):N=VAL(K$)
110 IF L=12 THEN 210
120 DRAW(X,Y)
130 DRAWC(X,Y)
140 IF F=1 THEN DRAW(X,Y):DRAW((X-2)/4
    +100,(Y-2)/4):GOTO 160
150 DRAWC((X-2)/4+100,(Y-2)/4)
160 IF K$="" THEN 100 ELSE BEEP
170 LOCATE 11,3
180 IF L=46 THEN IF F=0 THEN F=1:PRINT
     "ON "; ELSE F=0:PRINT "OFF";
190 IF N>0 THEN IF N<10 THEN IF N<>5 T
    HEN GOSUB (N+4)*100
200 GOTO 100
210 AA$="":BEEP
220 FOR K=0 TO 7
230 Z=0
240 FOR I=0 TO 3
250 IF POINT(I+100,K)<>0 THEN Z=Z+2^I
260 NEXT I
270 AA$=AA$+RIGHT$(HEX$(Z),1)
```

```
280 Z=0
290 FOR I=4 TO 7
300 IF POINT(I+100,K)<>0 THEN Z=Z+2^(I
    -4)
310 NEXT I
320 AA$=AA$+RIGHT$(HEX$(Z),1)
330 NEXT K
340 CLS :BEEP :LOCATE 10,0:PRINT $AA$
350 PRINT " $";AA$;"$"
360 PRINT " OK?(Push any key)";
370 K$=INKEY$:IF K$="" THEN 370 ELSE 1
    0
380 END
500 X=X-4:Y=Y+4:IF X<0 THEN X=30
510 IF Y>30 THEN Y=2
520 RETURN
600 Y=Y+4:IF Y>30 THEN Y=2
610 RETURN
700 X=X+4:Y=Y+4:IF X>30 THEN X=2
710 IF Y>30 THEN Y=2
720 RETURN
800 X=X-4:IF X<0 THEN X=30
810 RETURN
1000 X=X+4:IF X>30 THEN X=2
1010 RETURN
1100 X=X-4:Y=Y-4:IF X<0 THEN X=30
1110 IF Y<0 THEN Y=30
1120 RETURN
1200 Y=Y-4:IF Y<0 THEN Y=30
1210 RETURN
1300 X=X+4:Y=Y-4:IF X>30 THEN X=2
1310 IF Y<0 THEN Y=30
1320 RETURN
```

3-21 PB-770 GRAPHIC FUNCTIONS

The PB-770 has a large liquid crystal display (LCD) which displays 20 characters x 4 lines, and also has 160 x 32 dots which allow graphic displays.
PB-770 graphics can draw a precision graph and patterns using simple commands.
Also, a 4 color 114 mm wide plotter-printer with cassette interface (FA-10 or FA-11) can be connected to the PB-770.
Since up to 80 characters can be printed on 114 mm wide paper, this printer can be used in almost the same way as a full-scale plotter printer.
The compact PB-770 is provided with sophisticated graphic functions that should be mastered to fully utilize its capabilities.
Although you may feel them troublesome at first, you will soon become accustomed to the graphic functions with experience.

Plotter-printer with standard cassette tape recorder and cassette interface (FA-11)

3-22 GRAPHIC COMMANDS AND SCREEN COORDINATES

Graphics are drawn on the screen by connecting a series of dots, so the only thing required for graphics is providing dots at the proper locations. The PB-770 has two commands which are used to draw and erase dots at specified locations on the screen.

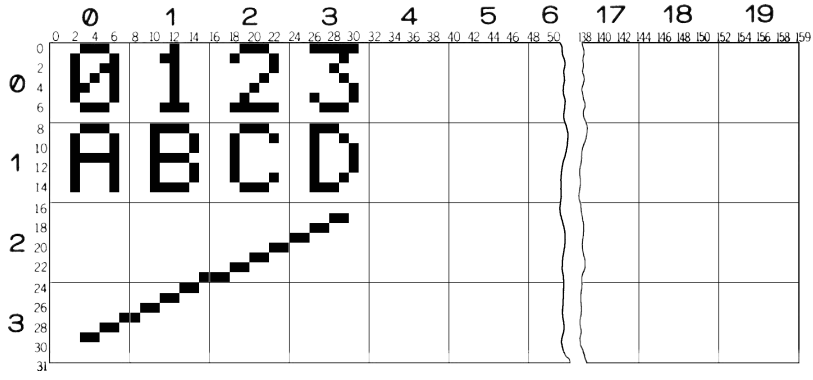
DRAW . . . . Draws dots and straight lines.
DRAWC . . . Erases dots and straight lines.

Also, the following convenient function is provided for graphics.

POINT . . . . Shows whether a dot is drawn or not at a specified location.

It is necessary to understand dot locations (coordinates) on the screen before explaining the use of the above commands and function.
The small numerals (horizontal: 0 to 159, vertical: 0 to 31) in the figure below indicate dot locations (graphic coordinates), while the large numerals (horizontal: 0 to 19, vertical: 0 to 3) indicate character coordinates.
A character is drawn by 8 x 8 dots, and its display location has to be determined.
Conversely, dots and lines can be drawn at any location using graphic coordinates.
The straight line from (3, 29) to (29, 17) was drawn by a graphic command which allows graphics to be freely drawn anywhere.

Screen coordinates



Graphic coordinates consist of 5120 dots with 160 dots in the X direction and 32 dots in the Y direction. The top left corner of the screen is (0, 0), and the bottom right corner is (159, 31).
Dot locations on the screen can be specified using these coordinates.
For example, to draw a dot at the (X, Y) location, use

```
DRAW (X, Y)
```

and to erase a dot at (X, Y), use

```
DRAWC (X, Y)
```

A straight line can be drawn with the same command by specifying the coordinates at both ends $(X_1, Y_1) - (X_2, Y_2)$ of the line as follows.

```
DRAW (X1, Y1) – (X2, Y2)
```

A line that connects three dots $(X_1, Y_1) - (X_2, Y_2) - (X_3, Y_3)$ can be drawn by specifying the following.

```
DRAW (X1, Y1) – (X2, Y2) – (X3, Y3)
```

Any number of lines can be drawn as a single line by linking coordinates with "–".
A straight line can be erased by specifying the following.

```
DRAWC (X1, Y1) – (X2, Y2)
```

When a dot at a specified location is lit (drawn), the POINT function produces 1, and when not lit 0. For example, the point at (X, Y) can be checked as follows.

```
POINT (X, Y)
```

The values 1 which indicates a dot is drawn, and 0 which indicates a dot is not drawn can be used in a program by assigning the value to a variable as follows.

```
A = POINT (X, Y)
```

Let's look at programs for drawing polygons using the method for drawing a straight line that was previously explained.
To draw a polygon, link the coordinates at the vertexes with the DRAW command.

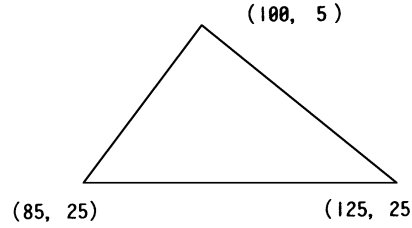
■ Triangle

A program that draws a triangle with vertexes (100, 5), (85, 25), (125, 25) is as follows.

```
10 REM --- TRIANGLE ---
20  CLS
30 DRAW(100,5)-(85,25)-(125,25)-(100,
   5)
40 END
```

Coordinate specification by the DRAW command can also be performed with numerical expressions.
For example, the program above can be rewritten using numerical expressions as follows.

Triangle drawn on the screen




```
10 REM --- TRIANGLE ---
20  CLS
30 X=100:Y=5 ..... Dot coordinates where drawing starts.
40 DRAW(X,Y)-(X-15,Y+20)-(X+25,Y+20)-
   (X,Y)
50 END
```

■ Rectangle

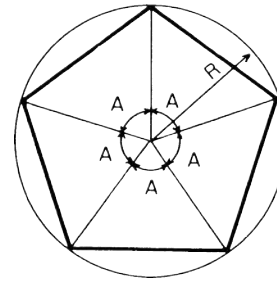
The following program draws a rectangle with the straight line (80, 5) – (150, 28) as its diagonal.

```
10 REM --- RECTANGLE ---
20  CLS
30 DRAW(80,5)-(150,5)-(150,28)-(80,28
   )-(80,5)
40 END
```

■ Equilateral Pentagon

An equilateral polygon is one that is inscribed inside a circle with each vertex having equal spacing.
The figure on the right shows an equilateral pentagon inscribed inside a circle with radius R. The lines that connect the 5 vertexes with the center of the circle all cross at the same angle, A (in this case A=72°).
A program for a pentagon inscribed inside a circle with a center (100, 18), and a radius 15 can be drawn as follows.

Equilateral Pentagon




```
10 REM --- PENTAGON ---
20  CLS
30 R=15            ····· Radius
40 X=100:Y=18      ····· Center of circle
50 A=360/5         ····· Angle
60 FOR I=0 TO 360 STEP A
70 DRAW(X-SIN(I)*R,Y-COS(I)*R)-(X-SIN
   (I+A)*R,Y-COS(I+A)*R)
80 NEXT I
90 END
```

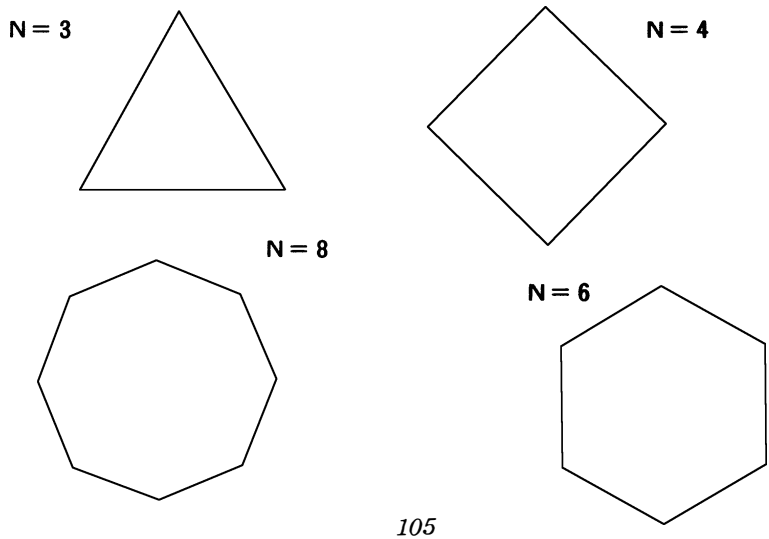
■ Equilateral Polygon N

Equilateral polygon N can be freely drawn by changing angle A in the program for the pentagon.

Pentagon A = 360/5
Polygon N A = 360/N

The following is a general purpose program that draws an equilateral polygon N after N is entered.
The center of the circle is (100, 15).

```
 10 REM --- POLYGON
 20  CLS
 30 R=15              ····· Radius
 40 X=100:Y=15        ····· Center of circle
 50 INPUT "N=";N      ····· Equilateral polygon N
 60 A=360/N           ····· Angle of equilateral polygon N
 70 FOR I=0 TO 360 STEP A
 80 DRAW(X-SIN(I)*R,Y-COS(I)*R)-(X-SIN
    (I+A)*R,Y-COS(I+A)*R)
 90 NEXT I
100 END
```



3-23 DRAWING A CURVE

A curve can be drawn by specifying dot coordinates on the screen. The problem, however, is how to specify the coordinates.
Many different curves can be drawn using mathematical formulas. Let's draw a circle and SIN curve as follows.

■ Circle

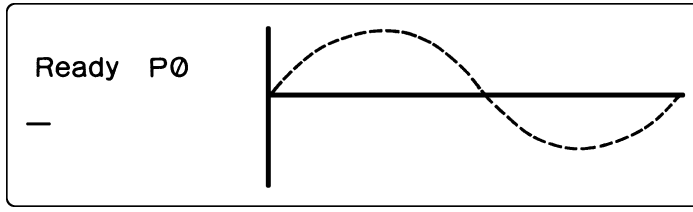
As the value N in the equilateral polygon program of the last section increases, the resulting figure comes closer to being a circle. Although, from a strictly geometrical point of view, a polygon can never be a circle, curves are actually produced on the screen by linking a series of straight lines.
The following program produces a circle with a radius of 15 originating at coordinates (100, 15).

```
10 REM --- CIRCLE
20  CLS
30 FOR I=0 TO 360 STEP 5
40 X=100+COS(I)*15 ..... Horizontal dot position
                         on the circular arc.
50 Y=15-SIN(I)*15  ..... Vertical dot position on
                         the circular arc.
60 DRAW(X,Y)
70 NEXT I
80 END
```

■ SIN Curve

The value of the SIN(I) function changes from 0 to 1 to 0 to −1 to 0 as the value of I progresses from 0° to 360°.
Therefore, a sine curve can be produced by drawing dots (that have been magnified to a suitable size) along a vertical axis. The following program draws a sine curve together with the X and Y axes.

```
 10 REM --- SINE ---
 20  CLS
 30 A=65:B=15 ····· SIN curve origin (origin of the X, Y axes).
 40 M=12      ····· Enlargement
 50 FOR I=0 TO 360 STEP 4
 60 X=A+I/4   ····· X coordinate dot
 70 Y=B-SIN(I)*M ····· Y coordinate dot
 80 DRAW(X,Y)
 90 NEXT I
100 DRAW(A,2)-(A,28)      ····· Y axis
110 DRAW(A,B)-(A+92,B)    ····· X axis
120 END
```



3-24 DRAWING A LINE GRAPH

Line graphs are often used when time variations are checked, such as a temperature changes or price changes over a period of time.
Therefore, it is recommended that the screen be fully utilized so that changes can be clearly seen.

■ **Monthly Average Temperature**

It is assumed that the monthly average temperature in a certain city is as shown in the following table. Data is entered in the program using a DATA statement. Then the monthly average temperature is drawn by the line program while data are supplied using a READ statement.

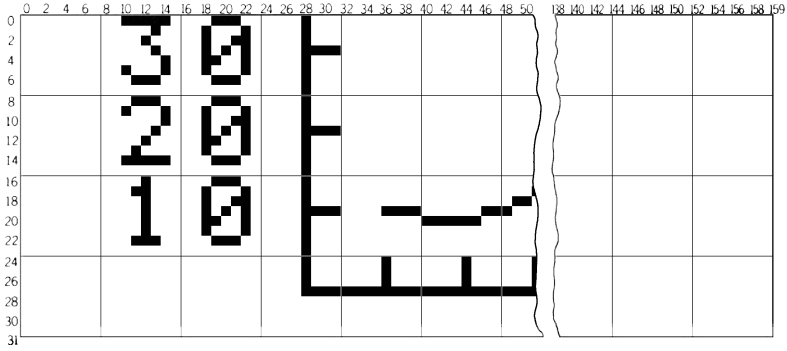
Month	Temp. (°C)	Month	Temp. (°C)	Month	Temp. (°C)
Jan.	11.5	May	19.8	Sep.	22.3
Feb.	9.8	June	23.4	Oct.	18.5
Mar.	13.7	July	26.6	Nov.	15.9
Apr.	18.3	Aug.	28.2	Dec.	14.7

```
 10 REM --- LINE GRAPH ---
 20  CLS
 30 FOR I=1 TO 3
 40 PRINT 30-(I-1)*10;CHR$(147)·····CHR$(147)
 50 NEXT I                                    =├
 60 PRINT TAB(3);CHR$(154); ·····CHR$(154)=└
 70 FOR I=1 TO 12
 80 PRINT CHR$(144); ·····CHR$(144)=┴
 90 NEXT I
100 FOR I=1 TO 12
110 X=36+(I-1)*8
120 READ A
130 Y=4+(30-A)*0.8
140 IF I=1 THEN 160
```

```
150 DRAW(P,Q)-(X,Y)
160 P=X:Q=Y
170 NEXT I
180 DATA11.5,9.8,13.7,18.3,19.8,23.4,2
    6.6,28.2,22.3,18.5,15.9,14.7
200 IF INKEY$="" THEN 200      Prevents the screen from
                               scrolling.
210 END
```

A portion of the dot pattern is magnified as shown below to help visualize the locational relationship of the graph's values, vertical axis (Y), horizontal axis (X), and lines. When you prepare a program for drawing a pattern or graph, it is recommended that such a picture be drawn to determine the locational relationship.

Dot Pattern



If line 200 were omitted, a command wait occurs soon after program execution has terminated which causes the following display and destroys the graph.

```
Ready P0
```

Line 200 preserves the graph on the screen until a keys is pressed.

3-25 PREPARATION FOR DRAWING A BAR GRAPH

■ Drawing A Bar Graph Using Characters

When you execute the following program and enter the numerals from 1 to 20 to N, N number of _ marks are displayed continuously.

```
10  CLS
20 INPUT "N=";N
30 FOR I=1 TO N
40 PRINT CHR$(131); ·····CHR$ (131) is the _ mark.
50 NEXT I
60 END
```

The length of the bar is proportional to the value of N. Since this unit is capable of displaying 20 characters in one line, however, it is necessary to scale N to keep values that exceed 20 within the limit of the screen. For example, to display a value of 100, the program above is modified as follows.

```
30  FOR  I=1  TO  N/6
          or
30  FOR  I=1  TO  N  STEP  6
```

It should be noted that with this program values from 90 through 95 would produce bars of the same length, so this technique is only capable of rough approximations. Graphics can be used to overcome this problem.

■ Using Graphics In A Bar Graph

The above program can be rewritten to include graphics as follows. By employing graphics, a total of 160 display units are available for resolution that is 8 times greater than the method used above.

```
10  CLS
20 INPUT "N=";N
30 FOR I=1 TO N*1.5
40 DRAW(I-1,8)-(I-1,14)
50 NEXT I
60 END
```

While the bars used for this program are plain, easy-to-read graphs can be drawn by changing the pattern of the bars.
Bar graphs can be drawn with the following patterns by adding or substituting the following routines (a) to (d) in the above program.

ⓐ ⓑ ⓒ ⓓ

```
ⓐ 35 IF I=N*1.5 THEN 40
  36 IF I MOD 2=0 THEN 50

ⓑ 30 FOR I=8 TO 14 STEP 2
  40 DRAW(0,I)-(N*1.5-1,I)

ⓒ 35 FOR J=8 TO 14 STEP 2
  40 DRAW(I-1,J+(I+1) MOD 2)
  45 NEXT J

ⓓ 25 FOR J=8 TO 14
  30 FOR I=1 TO N*1.5 STEP 3
  35 X=(J-8) MOD 3+I-1
  36 IF X>N*1.5-1 THEN X=N*1.5-1
  40 DRAW(X,J)
  50 NEXT I:NEXT J
```

3-26 TWO EXAMPLES OF BAR GRAPH PROGRAMS

Bar graphs are often used to provide representation of relative relationships among academic scores, sales, production amounts, etc. Besides this, the graph must be scaled to accurately show the magnitude of each item while keeping within the physical restrictions of the screen.
Basically, bar graphs can be classified into two general categories. The first type shows the relationship among a number of quantities by assigning each quantity to its own bar. (i.e. production or sales totals for individual products.) The second type shows the relationship of individual quantities to the whole. (i.e. production or sales totals for individual products compared with overall totals.)
Now let's prepare programs for both of these two basic types.

Data example: Production amounts for vehicles A and B are shown in the following table.

	1980	1981	1982
Vehicle A	48,200	57,200	67,200
Vehicle B	39,200	31,100	27,500

 : Pattern of the bar graph for vehicle A.

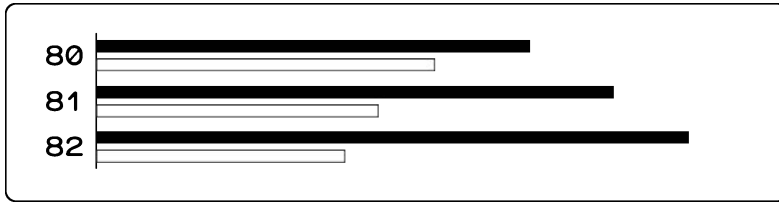
 : Pattern of the bar graph for vehicle B.

■ Assigning Each Item To Its Own Bar

```
10 REM --- BAR GRAPH ---
20  CLS
30 FOR I=0 TO 2
40 LOCATE 0,I
50 PRINT 80+I;CHR$(136) ·····CHR$(136)=I
60 FOR J=1 TO 2
70 READ A
80 FOR K=0 TO 2
```

```
 90 Y=I*8+(J-1)*4+K
100 IF J=2 THEN IF K=1 THEN DRAW(24+A/
    500,Y):GOTO 120
110 DRAW(25,Y)-(24+A/500,Y)
120 NEXT K:NEXT J:NEXT I
130 IF INKEY$="" THEN 130
140 END
150 DATA48200,39200,57200,31100,67200,
    27500
```

Display Example



■ Comparing Individual Totals with Overall Total

```
 10 REM ---BAR GRAPH ---
 20  CLS
 30 FOR I=0 TO 2
 40 LOCATE 0,I
 50 PRINT 80+I;CHR$(136)
 60 READ A,B
 70 FOR J=0 TO 6
 80 Y=I*8+J
 90 DRAW(25,Y)-(24+A/1000,Y)
100 NEXT J
110 X=24+A/1000:Y=I*8
120 DRAW(X+1,Y)-(X+B/1000,Y)-(X+B/1000
    ,Y+6)-(X+1,Y+6)
```

```
130 NEXT I
140 IF INKEY$="" THEN 140
150 END
160 DATA48200,39200,57200,31100,67200,
    27500
```

Display Example

3-27 ANIMATION DRAWING

When you execute the following program, the * mark moves to the left and right.

```
10  CLS
20 FOR I=5 TO 15
30 LOCATE I,1
40 PRINT " *"
50 NEXT I
60 FOR I=15 TO 5 STEP -1
70 LOCATE I,1
80 PRINT "* "
90 NEXT I
100 GOTO 20
```

Moves from left to right.

Moves from right to left.

As can be seen above, the control variable (I) increments from 5 to 15. This means that the coordinates of the LOCATE command in line 30 are sequentially changed from (5,1) through (15,1). It should be noted that the PRINT command in line 40 is written with a space inserted in front of the "*".
This space is essential because it "erases" the previously drawn "*" by replacing it with an empty space. This sequential displaying and erasing along successive positions produces an illusion of movement from left to right. Lines 60 through 90 decrement the control variable to reverse the procedure and cause the "*" to move from right to left on the screen. This short program represents the basic principle of graphic animation.
Vertical movement is performed in a similar manner, but, since the "*" is displayed in successive lines of the screen, a statement must be added to return one line to erase the previous "*". The following program illustrates vertical animation.

```
10  CLS
20 FOR I=0 TO 3
30 LOCATE 10,I:PRINT "*";
40 IF I=0 THEN 60
50 LOCATE 10,I-1:PRINT " "
60 NEXT I
```

Moves from up to down.

```
 70 FOR I=3 TO 0 STEP -1
 80 LOCATE 10,I:PRINT "*";
 90 IF I=3 THEN 110
100 LOCATE 10,I+1:PRINT " "
110 NEXT I
120 GOTO 20
```

Moves from down to up.

■ Adjusting Speed

If the * mark moves too fast in the above program, the speed is controlled using a FOR-NEXT statement.
Execute the program after adding the following line to the horizontal movement program.

```
45 FOR J=1 TO 50:NEXT J
```

When this statement is used, the speed of the movement from left to right becomes slightly slower. The speed is increased by reducing the final value of the FOR-NEXT statement, and is reduced by increasing the value.

■ Moving A Dot Like A Curved Line

The movement of a dot to form a curved line is accomplished using graphic coordinates.
In the following program, a dot repeatedly moves along a dotted line enclosed within a frame. (See Execution Example.)

```
10 CLS
20 DRAW(13,0)-(13,31)-(137,31)-(137,0)
```

..... Draws a frame.

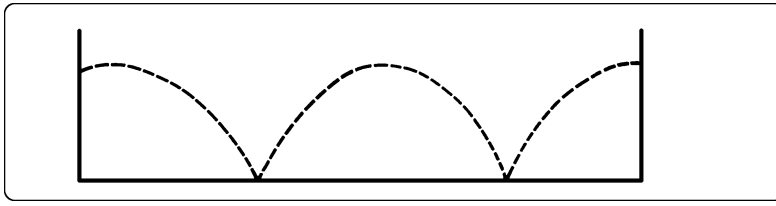
```
30 DRAW(14,0)-(14,30)-(136,30)-(136,0)
```

..... Draws a frame.

```
 40 N=1:P=14
 50 FOR I=0 TO 360 STEP 3
 60 X=P+N:Y=29-25*ABS(COS(I))
 70 DRAW(X,Y)
 80 IF P=14 THEN 100
 90 DRAWC(P,Q)
100 P=X:Q=Y
110 NEXT I
120 N=-N
130 GOTO 50
```

..... Computation of dot coordinates of the curved line.

Execution Example



■ POINT Function

The POINT function checks to see whether a dot is drawn or not at a specified location of the graphic coordinates.
If a dot is positioned at horizontal coordinate (X) and vertical coordinate (Y).

POINT (X, Y) · · · · 1

If a dot is not at that position,

POINT (X, Y) · · · · 0

The following program provides an example of the POINT function. At first, "CASIO PB-770" is displayed on the first line, then the POINT function checks whether each dot is drawn or not, and copies it to the third line depending on the value of (X,Y).

```
 10  CLS
 20 PRINT "*** CASIO PB-770 ***"
 30 FOR X=0 TO 159
 40 FOR Y=0 TO 7
 50 IF POINT(X,Y)=0 THEN 70
 60 DRAW(X,Y+16)
 70 NEXT Y
 80 NEXT X
 90 IF INKEY$="" THEN 90
100 END
```

Lines 30 through 80 confirm whether or not dots are drawn in the first line of the character coordinates. If dots are not present, execution jumps from line 50 to line 70. If dots are detected, they are reproduced at the same location of the character coordinates in the third line.
A hardcopy of the display can be produced using the optional FA-10 or FA-11 plotter-printer. Though details on the use of printers are included in another section, the following program is included as reference.

```
 10  CLS
 20 PRINT "CASIO PB-770"
 25 LPRINT CHR$(28);CHR$(37)
 30 FOR X=0 TO 159
 40 FOR Y=0 TO 7
 50 IF POINT(X,Y)=0 THEN 70
 55 U=X*0.59:W=Y*0.59
 60 LPRINT "D";U;",";-1*W;",";U+0.4;",
    ";-1*(W+0.4)
 70 NEXT Y
 80 NEXT X
 90 IF INKEY$="" THEN 90
100 END
```

3-28 GAME APPLICATIONS

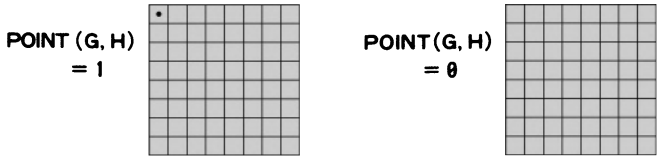
The animation learned in the previous sections is often used for games, and the following program illustrates this. Though requiring no particularly difficult technique, games require routines that realistically produce graphic animation.
The length of the program may make it look rather imposing, but it is suggested that the user try some modifications to become familiar with the function of each line. This is one of the best ways to add new programming techniques to your repertoire.
First, input the following program.

```
 10 REM ---BLOCK DESTROY---
 20  CLS
 30 FOR I=0 TO 3
 40 LOCATE 2,I:PRINT CHR$(137);
 50 NEXT I
100 FOR I=3 TO 6
110 FOR J=0 TO 3
120 LOCATE I,J
130 PRINT CHR$(141);
140 NEXT J:NEXT I
200 LOCATE 17,1
210 PRINT CHR$(147)
220 R=1:S=0
300 FOR N=3 TO 1 STEP -1
310 LOCATE 0,0:PRINT N
320 X=10:Y=INT(RND*2+1):A=1:B=1
330 LOCATE X,Y:PRINT CHR$(236);
340 Q$=INKEY$
350 IF Q$="1" THEN GOSUB 500 ELSE IF Q
    $="0" THEN GOSUB 600
360 IF X=3 THEN A=-A
370 IF Y=0 THEN B=-B ELSE IF Y=3 THEN
    B=-B
```

```
380 IF X=16 THEN GOSUB 800
390 LOCATE X,Y:PRINT " ";
400 IF X>16 THEN 440
410 G=(X+A)*8:H=(Y+B)*8
420 P=POINT(G,H)
430 IF P=1 THEN GOSUB 700
440 X=X+A:Y=Y+B
450 IF X>18 THEN BEEP :BEEP :BEEP  ELS
    E 330
460 NEXT N
470  CLS :LOCATE 2,1:PRINT "SCORE ";S
480 FOR I=1 TO 6:BEEP 1:NEXT I
490 IF INKEY$="" THEN 490 ELSE END
500 LOCATE 17,R:PRINT " ";
510 R=R-1
520 IF R<0 THEN R=0
530 LOCATE 17,R:PRINT CHR$(147);
540 RETURN
600 LOCATE 17,R:PRINT " ";
610 R=R+1:IF R>3 THEN R=3
620 LOCATE 17,R:PRINT CHR$(147);
630 RETURN
700 S=S+8-X:BEEP 1
710 LOCATE X+A,Y+B:PRINT " ";
720 X=X+A:Y=Y+B:A=-A
730 IF Y=0 THEN B=-B ELSE IF Y=3 THEN
    B=-B
740 RETURN
800 IF Y=R THEN A=-A:BEEP  ELSE IF Y+B
    =R THEN BEEP :A=-A ELSE 830
```

```
810 IF RND<0.7 THEN 830
820 LOCATE X,Y:PRINT " ";:X=X-1
830 RETURN
```

Line 420 uses a POINT function to check whether or not a dot is present at a certain location of a block. It does this by checking only the coordinate of the top left corner as shown below.



A block drawn with CHR$ (141).

A block erased with CHR$ (32).

■ How To Play

When this program is executed, blocks are drawn on the left and a racket is drawn on the right. A ball moves across the screen and the player must move the racket into place to keep the ball in play. Pressing the [1] key raises the racket and pressing the [0] key lowers the racket. If the player misses the ball with the racket, a miss is registered and the next ball is served. Missing three times ends the game. The final score is displayed at the end of the game.

■ Variable Table Of Block Destroying Program

A,B	Ball direction
G, H	Coordinates for checking whether a block is drawn or not
I, J	Variables for drawing blocks and fence
N	Number of remaining balls
P	Check whether a block is drawn or not
Q$	Racket direction
R	Racket position
S	Score
X, Y	Ball position

3-29 DRAWING A PATTERN WITH THE PLOTTER-PRINTER

Hardcopy can be produced with the connection of the optional FA-10 or FA-11 plotter-printer with cassette interface. Though a plotter-printer can be used for printing of text, it is especially suited for graphics. Both types of output are produced by drawing a number of dots in series to form the final figure.
This is similar to the method used for the production of figures on the screen, but the mechanical operation of the printer necessitates special commands to increase speed.
Though there are large number of plotter commands, their use makes operation much faster and contributes much to making programming easier.

■ Commands

All commands for making graphics with the plotter-printer start with LPRINT. Many functions can be performed when the commands shown in the command table are used together with LPRINT.
However, it is necessary to specify the graphic mode with the following statement before using these commands.

Graphic mode specification: LPRINT CHR$(28); CHR$(37)

The following statement is used to cancel the graphic mode (i.e., to specify the character mode).

Character mode specification: LPRINT CHR$(28); CHR$(46)

Plotter Command Table

	Command	Name	Description
Plotting	O	ORIGIN	Origin specification of ORG coordinates
	D	DRAW	Links a dot with a dot specified by ORG coordinates.
		RELATIVE DRAW	Draws a line up to a specified dot.
	M	MOVE	Moves to position indicated by ORG coordinates without plotting.
	R	RELATIVE MOVE	Moves to specified position without plotting.
	A	QUAD	Draws a parallelogram on X and Y axes with the diagonal of 2 dots indicated by ORG coordinates.
	C	CIRCLE	Draws a circle and circular arc with a dot specified by ORG coordinates as the center.
	X	AXIS	Draws a coordinate axis from the origin of ORG coordinates in direction of +Y, +X, −Y, and −X.
	G	GRID	Draws horizontal or vertical stripes in a specified square.
	L	LINE TYPE	Draws a solid line, broken line, one-dot chained line, or two-dot chained line.
	B	LINE SCALE	Specifies the pitch of a broken line, one-dot chained line, or a two-dot chained line.
Characters and symbols	S	ALPHA SCALE	Specifies the size of characters and symbols.
	Q	ALPHA ROTATE	Specifies the rotary direction of characters and symbols.

	Command	Name	Description
Characters and symbols	Z	SPACE	Specifies character spacing for a following digit and line.
	Y	VERTICAL or HORIZONTAL	Specifies horizontal or vertical writing.
	P	PRINT	Prints a character string.
	N	MARK	Draws a mark with a pen position as its center.
Control	J	NEW PEN	Pen color selection
	F	LINE FEED	Paper feed and paper return by one line unit.
	H	HOME	Modifies absolute coordinates, or shifts the pattern to a location that is easy to see.
	CHR$(64)	TEST	Checks pen use and pen ink. Execution Example: LPRINT CHR$(28) ; CHR$(37); CHR$(64) ↵
Character control	T	TAB	Tabulator
	?	FORMAT	Program list output

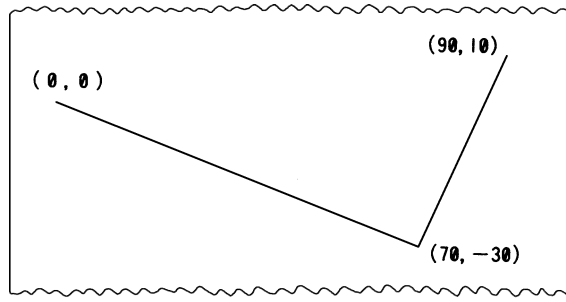
3-30 USING THE PLOTTER-PRINTER

Now let's produce a simple pattern using the plotter-printer. The following program connects the points at (0,0), (70,–30) and (90,10) with two straight lines. Therefore, the D command should be used.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "D";0;",";0;",";70;",";-30;
   ",";90;",";10
```

(90,10) (0,0) (70,–30)

Execution Example



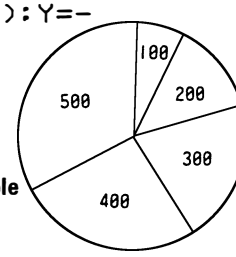
The same technique can be used to connect any number of points desired (as long as one programming line stays within the maximum of 79 characters).

The next program produces a circle graph. The C command is usually used to produce a circle with the plotter-printer, but lines must also be included to divide the circle to reflect the size of various data.
This particular program will use the data 100, 200, 300, 400 and 500 as the data.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "C48,-50,30"
30 S=0:T=0
40 FOR I=1 TO 5:READ A:T=T+A:NEXT I:R
   ESTORE :A=0
50 FOR I=1 TO 5:A1=A
60 READ A:S=S+A
```

```
 70 X=48+INT(30*SIN(360*S/T)):Y=-50+IN
    T(30*COS(360*S/T))
 80 LPRINT "D48,-50,";X;",";Y
 90 A2=(A1+A)/2:GOSUB 200:NEXT I
 95 END
100 DATA100,200,300,400,500
200 X=50+INT(20*SIN(360*S/(T+A2))):Y=-
    48+INT(20*COS(360*S/(T+A2)))
210 LPRINT "M";X-10;",";Y
220 B$=STR$(A)
230 LPRINT "P";B$
240 RETURN
```

Execution Example



Line 40 contains a READ command that reads the data in line 100. Lines 60 and 70 compute locations X and Y on the circumference of the circle for all the data. Then line 80 connects each point (X, Y) with the center of the circle.

3-31 USING PB-700 PROGRAMS

Though the PB-770 has more functions than the CASIO PB-700, programs written for the PB-700 can be used with the PB-770 without modification.

It should be noted, however, that compatibility is only upward, and certain programs written for the PB-770 cannot be executed with the PB-700 without modification. Programs for the PB-770 must be rewritten using only commands that can be executed with the PB-700.
Programs or data stored on cassette tape from the PB-700 can also be used with the PB-770 without modification, but programs or data files (DF) stored on cassette tape from the PB-770 cannot be read with the PB-700.
The difference between the PB-770 and PB-700 is as follows.

- Additional command

  POKE

- Additional functions

  DEG, HYPSIN, HYPCOS, HYPTAN, HYPASN, HYPACS, HYPATN, PEEK, DMS$, HEX$, &H

- Modified commands

  PRINT, CLEAR

- Modified function

  CHR$

CAUTION: When using programs written for the PB-700, if DATA statements are included in lines 2200 through 2299 or in lines with 22 in the last two digits of the line numbers (e.g. 22, 122, 622), change the line numbers. (See page 208.)

Preface to Chapter 4

The following terms are used in the "Format" section of each command or function described in Chapter 4.

- **"Numerical expression"**
  Numerical values, variables and computation expressions. Numerical computations are performed according to the precedence of the operators.
- **"Character expression"**
  Character constants, character variables and character strings.
  Character strings can be concatenated using the symbol "+". That is, addition of character strings makes a character expression.
  (Example) "ABC" + "DE" + "F" = "ABCDEF"
  Note that the number of characters of a character expression concatenated by + should be 79 or less.
- **"Variable"**
  Variables are used to store data. Since there are two types of data (numerical values and characters or symbols), there are numerical variables (such as A or B) and character variables (such as A$ or B$). Refer to Chapter 3 for details.
- **"Variable name"**
  A variable is an uppercase letter (A to Z) or it can be followed by $, number, or uppercase letter.
  (Example) A, A$, AB, A1, XY$, X1$ . . . etc.
  "Variable name" means this format of variable.
- **"Conditional expression"**
  A relational expression which compares the left side value and right side value using relational operators (such as =, =<, >, <>, etc.)
- **"Line number"**
  A number which is attached to the first position of each program line. Numbers 1 to 9999 can be used.
- **"Comment**
  Comments are written at appropriate parts of a program in order to explain program contents.
  They do not affect the program execution at all.
- **"Message"**
  Messages are displayed on the screen in order to let you perform proper in-put or to make the output easy to understand. They are used by enclosing them with " ".
- **"File name"**
  A file name identifies programs or data transferred between cassette tape and the computer.

CHAPTER 4

COMMAND REFERENCE

AUTO

Function	Automatically numbers program lines.
Format	AUTO [first line number] [,increment] (1≦ first line number ≦ 9999) (1 ≦ increment ≦ 9999)

The AUTO command greatly facilitates programming by generating sequential line numbers at preset increments.

With each press of the ↵ key, line numbers are continuously generated from the line number specified by the first argument and incremented by the step specified by the second argument. The default value for both the first line number and the increment is 10. This command cannot be executed with a password being specified.
This command can be released by the following operations.

1) Pressing the ↵ key without input after the line number is displayed.
2) Pressing the BRK or CLS key.
3) Attempting to exceed line number 9999.
4) When an automatically generated line number equals an existing line numb (♥ displayed after the lin number).

Usage

AUTO 100

The above input causes sequential generation of line numbers from line 100 at increments of 10 (i.e. 100, 110, 120, etc.).

AUTO 50, 50

The above input causes sequential generation of line numbers from line 50 at increments of 50 (i.e. 50, 100, 150, etc.).

CONT

Function	To restart the execution of a program that was suspended by a STOP statement or STOP (ANS) key entry.
Format	CONT

The CONT command is used to restart program execution stopped by the STOP command in a program or by operating the STOP (ANS) key. Program execution restarts from the statement following the STOP command.
Once program execution is suspended, numerical variables can be checked or changed before restarting, making this command a valuable debugging tool.

Usage

Insert the STOP command between sections during program preparation to provide easy debugging.
The following program was prepared as an example.

```
10 READ R,H
20 S=PI*R^2
30 STOP
40 V=S*H
50 PRINT R,H,S,V
60 DATA 10,20
70 END
```

When you execute this program with the RUN command, program execution is suspended at line 30 by the STOP command.
Check the execution content of line 10 and line 20. To display the values of the variables, press R [ENTER], H [ENTER] and S [ENTER]. Then 10, 20 and 314.1592654 appear which proves that execution has been performed correctly.
After checking has been completed, resume program execution using the CONT command.

```
CONT ↵
```

SEE STOP

DELETE

Function	Provides partial program deletion by line units.
Formats	DELETE *ln* DELETE *ln*– DELETE –*lm* DELETE *ln*–*lm* *ln*: First line to be deleted. *lm*: Last line to be deleted. (1 ≦ *ln* ≦ *lm* ≦ 9999)

This command is used to delete a specific line in a program or lines in a specified range.
When DELETE is used, it has the following basic formats.

(1) DELETE line number . . . . . Deletes a specified line.
(2) DELETE line number – . . . . Deletes lines from a specified line to the last line of a program.
(3) DELETE – line number . . . Deletes all lines up to a specified line.
(4) DELETE line number *n*
– line number *m* . . . . . Deletes the lines from line number *n* to line number *m*.

ln and *lm* have the following restriction.

1 ≦ *ln* ≦ *lm* ≦ 9999

This command cannot be used in a program.

Usage

A detailed explanation is provided using the following program.

```
10 REM DELETE SAMPLE
20 PRINT "20:A"
30 PRINT "30:B"
40 PRINT "40:C"
50 PRINT "50:D"
```

[1] DELETE line number [↵]

For example, when line 30 in the above program is to be deleted, enter the following.

DELETE 30 [↵]

Also, the same deletion can be performed by the following operation.

30 [↵]

[2] DELETE line number – [↵] (Use the [−] key to enter a "–".)

When the lines from line 30 to the last line are to be deleted in the above program, enter the following.

DELETE 30 [−] [↵]

Also, line 30 and after are deleted by:

DELETE 25 [−] [↵]

If there is no line 25, line 30, which is closest to that number, and all of the following lines are deleted.

[3] DELETE – line number [↵]

This command deletes the lines from the beginning of the program to a specified line number. As an example, enter the following.

DELETE [−] 40 [↵]

Check the listing with [SHIFT] [LIST] [↵], to determine that lines 10–40 were deleted. The same result can be realized by entering the following.

DELETE [−] 47 [↵]

Again, though there is no line 47, the lines from the beginning of the program to the nearest line inside of 47 are deleted.

[4] DELETE line number *n* – line number *m* [↵]

To delete the lines from line 20 to line 40 from the previous program, enter the following.

DELETE 20 [−] 40 [↵]

- Attempting to use the DELETE command in a program protected by the PASS command will result in a PR error.
- Including a fractional part in the line number will result in an SN error.
- If *n* is greater than *m* in item [4] above, or if a specified line number does not exist, nothing will be deleted.

SEE NEW, NEW ALL, PASS

EDIT

Function	Allows a program to be modified.
Format	EDIT EDIT *ln* ($1 \leqq ln \leqq 9999$)

The EDIT command is used to edit a program for deletions, additions, corrections, etc.
The first line of a program is displayed by EDIT [↵] , and the cursor appears at the end. Move this cursor to the position where a correction is to be made with [→], [←], [SHIFT] [↑] or [SHIFT] [↓], and perform corrections with [DEL] or [SHIFT] [INS DEL] .

■ Using The Edit Keys

(1) [→], [←] ..... Moves the cursor to the right and left. Holding down either key will move the cursor in the respective direction until the end (beginning) of the line is reached.

(2) [SHIFT] [↑] [SHIFT] [↓] ..... The cursor can be moved up and down by pressing the [SHIFT] key and the Cursor key at the same time.

(3) [DEL] ..... If the key is held down, characters are continuously deleted until the key is released.

(4) [SHIFT] [INS DEL] .. If the key are held down, spaces are continuously generated until the keys are released.

Usage

Now let's edit a program by inputting the following program.

```
10 PRINT "AVERAGE"
20 INPUT A,B,C
30 H=(A+B+C)/3
40 PRINT H
50 GOTO 10
```

1 Checking A Program List From The Beginning

To display a program from the beginning and correct 1 line at a time, the following is used.

EDIT ⏎

Next, each time the ⏎ key is pressed, the next line will be displayed.

2 Checking A Program List From The End (IN REVERSE)

To display line 10 of a program after the program has been listed from the beginning to line 50, perform the following operation.

Program list displayed up to line 50.

```
20  INPUT  A, B, C
30  H= (A+B+C) /3
40  PRINT  H
50  GOTO  10_
```

⇩

Operate SHIFT ▲ four times until line 10 is displayed as shown on the right.

```
40  PRINT  H
30  H= (A+B+C) /3
20  INPUT  A, B, C
10  PRINT "AVERAGE"_
```

3 Checking A Program From A Middle Line Number

To check a line number in the middle of a program, perform the following operation.

EDIT line number ⏎

After this, the subsequent lines are displayed every time the ⏎ key is pressed.

4 **Using ⇦ , ⇨ , INS DEL**

Line 20 and line 30 in the program are corrected as follows.

```
20  INPUT  A, B, C, D
30  H=(A+B+C+D)/4
```

When you use EDIT ↵ to list line 20, the display is as follows.

SHIFT EDIT(E) 20 ↵

```
Ready  P0
EDIT  20
   20  INPUT  A, B, C_
```

In line 20, after entering " , " and then "D", the correction is completed by pressing the ↵ key and line 30 will be displayed.

, D ↵

```
Ready  P0
EDIT  20
   20  INPUT  A, B, C, D
   30  H= (A+B+C)/3_
```

To insert "+D" in line 30, provide space for two characters with:

⇦⇦⇦ SHIFT INS DEL INS DEL

Pressed together

and press +D⇨⇨4 ↵

- If the EDIT command is used for a program with a password, a PR error will occur.
- Edit Mode Release
  The Edit Mode is released as follows.
  (1) When the BRK key is pressed.
  (2) When the CLS key is pressed.
  (3) When incorrect operation is performed.
  (4) When there is no program to be displayed.
  (5) When the power is turned off.

LIST/LLIST

Functions	LIST: Displays the contents of a program. LLIST: Prints out the contents of a program.
Formats	LIST LLIST LIST LLIST ALL LIST LLIST V LIST LLIST *ln* LIST LLIST *ln*– LIST LLIST –*ln* LIST(LLIST *ln*–*lm* *ln*, *lm* : line numbers ($1 \leqq ln \leqq lm \leqq 9999$)

The LIST and LLIST commands are used to view programs stored in the PB-770. LIST produces the program on the display while LLIST prints out the program on roll paper.
Not only programs, but registered variables can be listed.

Usage

LIST and LLIST commands are utilized as follows.

1 LIST (LLIST) ↵
This command sequentially displays (prints) all of the lines of a program in the current program area beginning from the first line.

2 LIST (LLIST) ALL ↵
The PB-770 has a total of 10 program areas from P0 through P9, each of which can be used for input of independent programs. This command is used to display (print) all programs in all of the program areas.

3 LIST (LLIST) V ↵
This command is used to display (print) all of the presently used registered variables. Besides registered variables, the array names for arrays declared using the DIM command can also be displayed (printed).

4 LIST (LLIST) line number ↵
When a line number is specified after LIST (LLIST), that line of the program in the present program area will be displayed (printed). To display line 30 of a program located in program area P5, for example, perform the following key operation:

PROG 5 ↵ LIST 30 ↵

[5] LIST (LLIST) line number – [EXE] (Use the [(-)] key to enter a "–".)
When the line number is followed by a "–", the program in the present program area will be displayed (printed) from the specified line number to the end. The following key operation will display the program in the present program area from line 70 to the end.

LIST 70 [(-)] [EXE]

[6] LIST (LLIST) – line number [EXE]
When the line number is preceded by a "–", the program in the present program area will be displayed (printed) from the beginning to the specified line number. The following key operation will display the program in the present program area from the beginning to line 100.

LIST [(-)] 100 [EXE]

[7] LIST (LLIST) line number n – line number m [EXE]
In this case, the program is displayed (printed) from line number n to line number m. Since the program is displayed from the smaller line number to the higher line number, n must be less than or equal to m. The following key operation will display from line 50 to line 70 of the program in the present program area.

LIST 50 [(-)] 70 [EXE]

Technique

[1] Suspending LIST, LIST ALL Execution

Because the lines of a program are displayed continuously when a program is listed with the LIST command, it is difficult to confirm the program content. To overcome this, press the [EXE] key to momentarily stop the display. To resume the program listing, press the [EXE] key again.
LIST command execution cannot be stopped with [SHIFT] [STOP (ANS)].
When a program listing check is not required, press the [BRK] key to cancel LIST or LIST ALL command execution.

[2] Leaving LLIST, LLIST ALL

When a program list is being printed out with the LLIST or LLIST ALL commands, temporary suspension cannot be performed with the [EXE] key. You can only cancel LLIST command execution by pressing the [BRK] key.

LOAD

Function	Loads a program or data stored on a cassette tape into the main frame memory.
Formats	LOAD LOAD ALL LOAD, A LOAD, M LOAD, D, address

The LOAD command is used to read back into the PB-770 a program or data that was stored on a cassette tape by a SAVE command. The file name that was used during SAVE is now useful. Enter the file name together with the LOAD command, then the PB-770 automatically searches for the file name and reads the program from the tape.
If a file name is not specified, the first program or data found is read in.

Explanation

The LOAD command has the following formats.

(1) LOAD
Reads the first program found among those stored by SAVE or SAVE "file name" into the presently specified program area.
There is no problem even if the program area during "SAVE" and the presently specified program area are different.

(2) LOAD "file name"
Reads the program that was stored with the same file name into the presently specified program area. In this case, the program area during "SAVE" and the program area during "LOAD" can be different.

(3) LOAD ALL
Reads programs that were stored by SAVE ALL to the same program areas from which they were stored. Since there is no file name, the programs found first among those stored by SAVE ALL are read in.

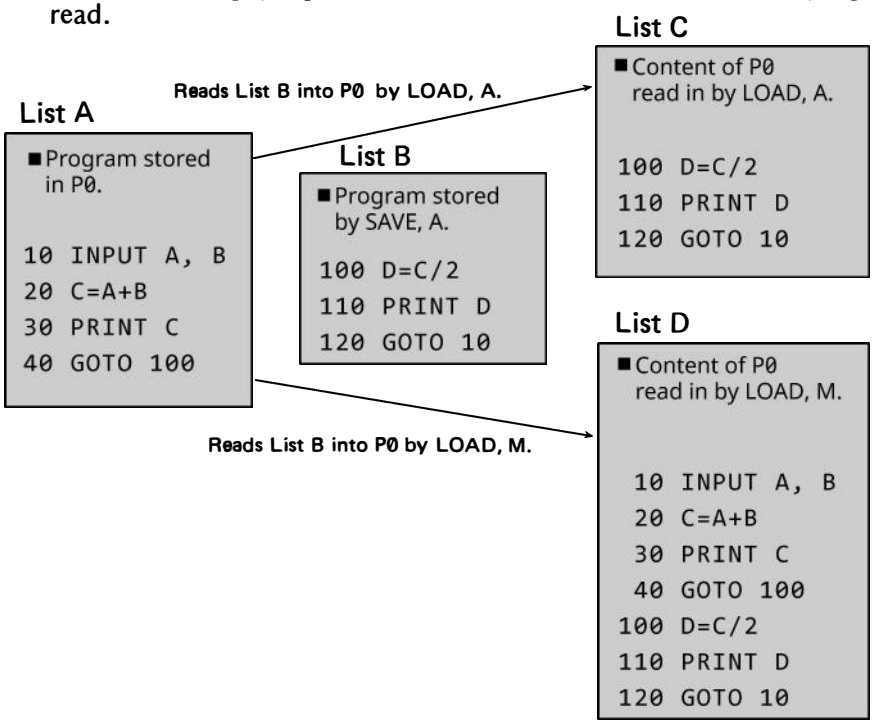
(4) LOAD ALL "file name"
Reads the programs into the PB-770 with the same file name among those stored by SAVE ALL "file name".

(5) LOAD, A/LOAD "file name", A

Reads a program into the PB-770 that was stored in ASCII code (SAVE, A or SAVE "file name", A). If there is no file name, the first program found is read in, and if there is a file name, the program with the same file name is read in.

(6) LOAD, M/LOAD "file name", M

Reads a program that was stored by SAVE, A or SAVE "file name", A. The difference between this format and LOAD A, LOAD "file name",A is that reading a program in this format will not erase a program already in the computer, provided that the line numbers of the existing program are different from those of the program read.



(7) LOAD, D, address

Reads data that was stored in internal code (SAVE,D,address 1,address 2 or SAVE "file name",D,address 1,address 2). Addresses are restricted to the range of −32769 < address < 65536. See CLEAR for details concerning addresses.

Usage

1 SAVE and LOAD formats must match.

The format of the LOAD command must correspond to that used for the SAVE command. Similarly, a program stored with SAVE ALL must be read with LOAD ALL. The following table shows the relationship between the LOAD command and SAVE command.

	LOAD	LOAD "file name"	LOAD ALL	LOAD ALL "file name"	LOAD,A	LOAD "file name", A	LOAD,M	LOAD "file name", M	LOAD,D, address	LOAD "file name", D,address
SAVE	0	X	X	X	X	X	X	X	X	X
SAVE "file name"	0	0	X	X	X	X	X	X	X	X
SAVE ALL	X	X	0	X	X	X	X	X	X	X
SAVE ALL "file name"	X	X	0	0	X	X	X	X	X	X
SAVE,A	X	X	X	X	0	X	0	X	X	X
SAVE "file name",A	X	X	X	X	0	0	0	0	X	X
SAVE,D,address 1, address 2	X	X	X	X	X	X	X	X	0	X
SAVE"file name",D, address 1,address 2	X	X	X	X	X	X	X	X	0	0

2 Password

If a program with a password is stored, the password is also stored. A reference for when programs with a password attached are loaded is as follows.

(1) LOAD can be performed at any time there is no password stored in the PB-770. In this case, a stored password exists as a PB-770 password.

(2) When a password exists in the PB-770, LOAD can only be performed when the program being loaded has the same password. If the passwords are different, a PR error will be generated.

3 Error during LOAD

(1) RW error
This error occurs when a parity error is generated during LOAD. In this case, clear the program which has been loaded by entering NEW ↵ , and perform loading from the beginning.

(2) OM error
This error occurs when the memory capacity is insufficient. In this case, clear unnecessary programs, make the first address of the data area larger, or expand RAM capacity.

SEE SAVE, VERIFY, CHAIN, PUT, GET, PASS

NEW/NEW ALL

Function	Program erase.
Formats	NEW NEW ALL

When new program input is performed, it is necessary to erase the previous program. The commands that erase the previous program are the NEW/NEW ALL commands.

1 NEW Command Functions

The NEW command erases a program in a specified program area with the PROG command used to specify the program area. An attempt to use the NEW command to erase a program protected by a password (see page 144, PASS) will result in a PR error.
Also, variables cannot be cleared (see page 158, CLEAR).

2 NEW ALL Command Functions

The NEW ALL command erases the programs in all of the program areas at one time. Since this command is effective when the PASS command has been executed, careful confirmation should be performed when it is used.
Not only programs, but all variables are cleared, the program area is set to P0, ANGLE is set to 0 (DEG), and the data area is released (see CLEAR).

Usage

The operation is as follows.

NEW ⏎ or NEW ALL ⏎

PASS

Function	Protects a program by assigning a password.
Format	PASS "password"

Often a program that was prepared with great effort is erased by mistake, or is destroyed by writing another program on top of it.
Therefore, important programs are protected with this command to prevent program corrections and erasures from being performed.

1 PASS Command

The PASS command is utilized by entering:

PASS "Password with up to 8 characters" ↵

When this command is used, program corrections and erasures cannot be performed unless the password is cancelled. (This command cannot be used in a program.) The following commands cannot be executed.

(1) AUTO
(2) DELETE
(3) EDIT
(4) LIST, LLIST
(5) NEW

Also, a program can not be newly written. The functions of the PASS command are effective in all of the program areas. Conversely, the PASS command cannot be assigned to only a single program area. An attempt to execute any of the above listed commands when the PASS command has been used will result in a PR error being displayed.

2. PASS Command Release

Characters, numerals and symbols can be used for a password with up to 8 characters. To release a password, make an entry using exactly the same password as follows.

PASS "Password with up to 8 characters" ↵

Therefore, if a password is entered and forgotten, the PASS command function can never be released. Since a password cannot be observed, it is recommended that something which cannot be forgotten, such as your name, be used for a password. Also, before password entry is performed, it is important to confirm that it is correct. If passwords are forgotten, the only solution is to execute the NEW ALL command.
However, since the programs in all the program areas will disappear, they should first be stored on a cassette tape using the SAVE command.

SEE SAVE, LOAD, NEW

PROG

Function	Specifies a program area.
Format	PROG numerical value (or numerical expression) 0 ≦ numerical value (numerical expression) < 10

The PB-770 is provided with 10 program areas (P0—P9) where independent programs can be written.
The PROG command is used to specify a program area.

When the power of the PB-770 is turned on, the display will be as follows.

Power ON

```
Ready  P0
_
```

Usage

When the power is turned on, the program area P0 is automatically specified as shown above. Next, let's specify P9 as the next program area.

Power ON

SHIFT PROG 0 9 ↵

```
Ready  P0
PROG  9
Ready  P9
_
```

Though a numerical expression can be entered after PROG, a computation result (X) within the range of 0 ≦ X < 10 can also be used.
If it is outside this range, a BS error is displayed.

Examples

PROG (100/20) → Specifies P5 with PROG5.
PROG (100/10) → Displays a BS error.
PROG (100/15) → Specifies P6 (The fractional part is discarded.)

SEE GOTO, GOSUB

RUN

Function	Executes a program.
Formats	RUN RUN *ln* ($1 \leqq ln \leqq 9999$)

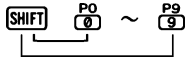
The RUN command is used to execute a program in the presently specified program area.

The RUN command has the following two formats.

(1) RUN [↵]
Starts program execution from the first line of the specified program area. Execution is performed in a sequence with the smallest line number first.

(2) RUN line number [↵]
Starts execution from the line number after RUN. If the specified line number does not exist, execution starts from the nearest line number larger than the specified line number.

Execution can be started from the beginning of any program in a program area without using the RUN command as follows.



Programs P0—P9 can be executed by pressing the [SHIFT] key and any key from [0] ~ [9] at the same time.

SAVE

Function	Stores a program on cassette tape.
Formats	SAVE SAVE ALL SAVE, A SAVE,D, address 1, address 2 SAVE "file name" SAVE ALL "file name" SAVE "file name",A SAVE "file name",D, address 1, address 2

The SAVE command is used when a program is stored on cassette tape.
This command has the following formats.

(1) SAVE . . . . . Stores in binary code format a program from a presently specified program area.
(2) SAVE ALL . . . . . Stores in binary code format the programs of all program areas.
(3) SAVE,A . . . . . Stores in ASCII code format a program in a presently specified program area.
(4) SAVE, D, address 1, address 2
Stores in binary code format the contents from address 1 to address 2 of the memory. The range of the addresses is $-32769 <$ address 1 $\leqq$ address $2 < 65536$. See CLEAR for information on addresses.

- SAVE in ASCII code format requires more time and uses more tape than SAVE in binary code format (internal code format).
  However, it is necessary to use SAVE in ASCII code format to execute LOAD, M (see LOAD command).

Usage

Assigning file names to each program stored on cassette tape makes program handling easier.

1 File Names

SAVE "file name"
SAVE ALL "file name"
SAVE "file name", A
SAVE "file name", D, address 1, address 2

- Although any character or symbol can be used as a file name, 8 characters or less must be used.
- For a program with a password, the actual password is also output as data.

2 Recording Tape Counter Reading

When you press the ↵ key with the cassette tape recorder in the record mode, recording starts. The automatic remote function stops the tape when it has terminated. Make a record of the tape counter reading for recording start and end.

3 Confirmation

Confirm if cassette tape recording was correctly performed using the VERIFY command (see page 153, VERIFY). If it is not correctly recorded, perform the SAVE operation again.

File Attributes

The file attribute names of programs or data which have been stored by SAVE or PUT are displayed when they are loaded to the computer by LOAD or GET.

Operation for output	Display during loading
SAVE	PF B
SAVE ALL	AF B
SAVE, A	PF A
SAVE, D, address 1, address 2	DF B
SAVE "ABC"	ABC PF B
SAVE ALL "ABC"	ABC AF B
SAVE "ABC", A	ABC PF A
SAVE "ABC", D, address 1, address 2	ABC DF B

The file names are displayed as they are.
Meanings of the file attributes are as follows.

Display during loading

PB-770 PF B
File name (A) (B) (C)

(A) P : Program
A : All programs
D : Data

(B) File

(C) B : Binary (Internal code format)
A : ASCII (ASCII code format)

SYSTEM

Function	Displays program area status, specified angle unit, memory capacity, number of remaining usable bytes, and data area first address.
Format	SYSTEM

First, input SYSTEM ↵ , and a display similar to that shown below will appear. The actual display will differ somewhat if a RAM expansion pack is used, if programs are stored in the computer, or depending on how much data area is available.
Enter SYSTEM ↵ . If there is no program and data is stored in the PB-770, the following is displayed.

```
P 0123456789 ANGLE 0
 8KB 6871B
Ready P0
_
```

The following points should be noted concerning each part of the display.

(1) The P 0123456789 in line 1 indicates whether or not programs are stored in program areas P0 through P9. The program area number will be marked with a ♥ when it contains a program.
(2) The ANGLE 0 in line 1 indicates the specified angle unit mode. ANGLE 0 (DEG) is always specified directly after the unit is turned ON. (ANGLE 1 = RAD, ANGLE 2 = GRAD)
(3) The 8KB in line 2 indicates the memory capacity. This value will be different when RAM expansion packs (OR-8) are used.

RAM	Quantity	Memory capacity
OR-8	None	8KB
OR-8	1	16KB
OR-8	2	24KB
OR-8	3	32KB

(4) The 6871B in line 2 indicates the number of remaining usable bytes. This value will be different when RAM expansion packs (OR-8) are used and when the specified address of the data area is changed.

(5) When a data area is specified, the first address of the data area is displayed after the number of remaining bytes in line 2. This value also changes according to the specified address of the data area. When a data area is not specified, nothing is displayed.
(6) The Ready P0 in line 3 indicates that input is possible to program area P0. P0 is always specified when the unit is turned ON.

SEE **ANGLE, CLEAR**

VERIFY

Function	Performs a parity check of programs or data stored on a cassette tape.
Formats	VERIFY VERIFY "file name"

The VERIFY command checks if programs or data stored on a cassette tape by the SAVE or PUT command are correctly stored.

- This command has the following formats.

(1) VERIFY
Checks the first program found on cassette tape.

(2) VERIFY "file name"
Checks the program with the same file name on cassette tape.

- You can check all programs stored by the 8 formats of SAVE command (see SAVE) by using VERIFY ↵ or VERIFY "file name" ↵. In other words, it is not necessary to specify a format of SAVE command.
- If SAVE was not correctly performed, an RW error is displayed. If this occurs, store the programs again.

Usage

The actual procedure is as follows.

Step 1: Tape Rewind — Return the tape on which programs are stored by the SAVE command to the initial location by using the tape counter.

Step 2: VERIFY Command Input — If the program has a file name, enter VERIFY "file name" ↵, and if it has no file name, enter VERIFY ↵.

Step 3: Press the PLAY button of the cassette tape recorder. When check starts, either PF B, AF B, PF A, DF B or DF A (see page 150) is displayed. If a program or data was correctly stored, Ready P0–P9 is displayed and the cassette tape stops. If the program or data was not correctly stored, an RW error is displayed and the cassette tape stops.

ANGLE

Function	Specifies the angle unit.
Format	ANGLE numerical expression (0≦numerical expression < 3)

The angle unit is usually expressed as 30° and 60° and called DEGREE. However, RADIAN and GRAD are also used in mathematics. The PB-770 can handle any of these units.

The ANGLE command is used to specify the following three angle units.
(1) DEGREE . . . . . (Example) 45°, 90° Input range of x:
$-5400^\circ < x < 5400^\circ$
(2) RADIAN . . . . . (Example) 0.5π, 2π Input range of x:
$-30\pi < x < 30\pi$
(3) GRAD. . . . . .(Example) 300, 1000 Input range of x:
$-6000 < x < 6000$

The relationship between these angle units is as follows.

360 DEG (= 360°) = 2π RAD = 400 GRAD

The angle unit is specified by the ANGLE command as follows.

ANGLE 0 — Specifies DEGREE
ANGLE 1 — Specifies RADIAN
ANGLE 2 — Specifies GRAD

The unit is always set to ANGLE 0 (DEGREE) when the power is turned on.

SAMPLE PROGRAM

```
10 REM *** ANGLE ***
20 ANGLE 0:PRINT SIN30;
30 ANGLE 1:PRINT SIN(PI/6);
40 ANGLE 2:PRINT SIN(100/3)
50 END
```

This program displays the value of SIN by using 3 angle units. All the results are 0.5.

BEEP

Function	Generates a buzzer sound.
Formats	BEEP BEEP 0 BEEP 1

BEEP command is provided in the PB-770 to generate a buzzer sound. There are many ways to use a buzzer sound. For example, when a long period of time is required for the execution of a program, execution termination is indicated by the sounding of the buzzer accomplished by inserting a BEEP command at the position of execution termination. Also, the fun of a game is increased by using this command.

The BEEP command has the following three formats.

(1) BEEP
Generates a relatively low buzzer sound.

(2) BEEP 0
Generates the same sound as BEEP.

(3) BEEP 1
Generates a slightly higher buzzer sound.

SAMPLE PROGRAM

```
100 REM*** BEEP ***
110 IF INKEY$="" THEN 110
120 IF INKEY$="0" THEN BEEP 0
130 IF INKEY$="1" THEN BEEP 1
140 GOTO 110
```

This program was prepared to generate low buzzer sounds when [0] is pressed and high buzzer sounds when [1] is pressed.

CHAIN

Function	Loads a specified program and executes it from the first line.
Formats	CHAIN CHAIN "file name"

When a CHAIN command appears during the execution of a program, program execution is stopped at that point. Then a program with the file name that was specified by the CHAIN command is loaded from a cassette tape, and execution is performed from the beginning of the program.
If there is no file name, the first program found which was stored by SAVE or SAVE " file name" is loaded.
The CHAIN command has the following two formats.

(1) CHAIN
Loads first PF B found (program stored by SAVE or SAVE "file name") and executes it.

(2) CHAIN "file name"
Loads PF B of specified file name and executes it.

- Since the program is loaded in the presently specified program area, the previous program is erased with NEW.
- Programs stored by "SAVE ALL" and "SAVE, A" cannot be read in with the CHAIN command.
- If a password is attached to a loaded program, the password is also read in.
- Even when CHAIN is executed, variables are not cleared.

Usage

Input Lists 1–3 into program areas P1–P3, then store them on a cassette tape.

⟨List 1⟩. . . . . Program that computes the area of a circle.
(File name: "PRO. 1")
⟨List 2⟩ . . . . . Program that computes the area of a triangle.
(File name: "PRO. 2")
⟨List 3⟩ . . . . . Program that computes the area of a rectangle.
(File name: "PRO. 3")

Also, input List 0 into P0 and execute it. List 0 is used to select List 1 – List 3 using the CHAIN commands in lines 60–80, read it into the PB-770, and then execute the computation.

List 0

```
10 REM CHAIN PRO.0
20  CLS :PRINT "AREA CALCULATIONS"
30 PRINT "1CIRCLE  2TRIANGLE  3RECTANGL
   E"
40 PRINT "SELECT NO.
50 BB$=INKEY$:IF VAL(BB$)>3 THEN 50 E
   LSE IF VAL(BB$)<1 THEN 50
60 IF BB$="1" THEN CHAIN "PRO.1"
70 IF BB$="2" THEN CHAIN "PRO.2"
80 CHAIN "PRO.3"
```

List 1

```
10 REM PRO.1
20 INPUT "RADIUS";RR
30 S=PI*RR^2
40 PRINT S
50 END
```

List 2

```
10 REM PRO.2
20 INPUT "HEIGHT";HH
30 INPUT "BASE";LL
40 S=HH*LL/2
50 PRINT S
60 END
```

List 3

```
10 REM PRO.3
20 INPUT "LENGTH";HH
30 INPUT "WIDTH";LL
40 S=HH*LL
50 PRINT S
60 END
```

CLEAR

Function	Clears all variables. Clears all variables and creates a data area.
Format	CLEAR [first address of data area]

The CLEAR command is used to clear all numerical variables and character variables.
Numerical variables are cleared to 0 and character variables are cleared to " " (null-string). Also, at the same time, registered variables in a program are deleted, and the defined array variables are deleted.
Since the FOR nesting stack is cleared, a FOR-NEXT loop cannot be continued.
When the first address of a data area is specified, an area is created to which data can only be written using the POKE command and from which data can only be read using the PEEK function. This method is ideal for the storage of valuable data.

Usage

(1) Clearing variables

The program below is used to display data by totalizing the sum of data and the number of data. However, if the program is executed again after pressing the BRK key, the correct answer cannot be obtained since the numerical values assigned to variables S and N at the first execution remain. Therefore, the CLEAR command is inserted between line 10 and line 30 as shown in the program on the right so that a correct answer can be obtained every time execution is performed.

```
10 PRINT "TOTAL"
30 INPUT "D=",D
40 S=S+D
50 N=N+1
60 PRINT "S(";N;")=";S
70 GOTO 30
```

```
10 PRINT "TOTAL"
20 CLEAR
30 INPUT "D=",D
40 S=S+D
50 N=N+1
60 PRINT "S(";N;")=";S
70 GOTO 30
```

(2) Creating a data area

First of all, actually create a data area using the CLEAR command.

Assume that the above program is stored in program area P0. Perform the following key operation:

CLEAR 3000 ↵

This will create a data area from address 3000 (10).
Next, check the number of remaining bytes by entering:

SYSTEM ↵

CLEAR 3000 ↵
SYSTEM ↵

```
P ♥123456789 ANGLE 0
 8KB 1615B 3000
Ready P0
```

The resulting value should be 1615B (bytes), but this value will differ if a program is stored in other program area or if the RAM area has been expanded.
Now enter:

CLEAR 5000 ↵

and execute the SYSTEM command again.

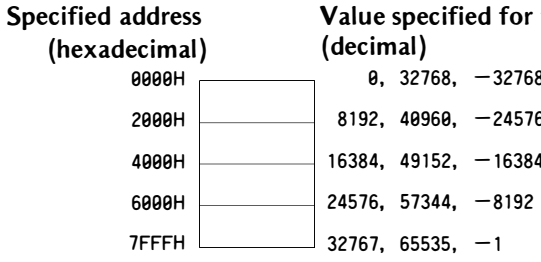
CLEAR 5000 ↵
SYSTEM ↵

```
P ♥123456789 ANGLE 0
 8KB 3615B 5000
Ready P0
```

Confirm that the number of remaining bytes has increased. In this way the number of bytes available for BASIC programming or variables is increased by the amount that the specified size of the data area is decreased. Data area creation using the CLEAR command is required when PEEK and POKE are used to read from and write to the specified addresses. Numerical variables and character variables are cleared by the CLEAR command, but the contents of the data area remain unchanged. Therefore, it is unnecessary to worry about the data area for normal programming (when PEEK and POKE are not used). Data stored in a data area created by specifying an address in the CLEAR command can be input to/read from a cassette tape much more quickly than PUT/GET operations using the LOAD, D, address and SAVE, D, address 1, address 2 commands.

(3) Specifying the starting address

The allowable range for the starting address of the data area is $-32769 <$ address < 65536. However, when the starting address is within the system area (&H0000 through &H0528), and, when a program is stored in the computer, the address must be larger than the last address of the program. The value specified for the starting address of the data area and the actually specified address have the following relationship.



NOTE: 0 through 1320, 32768 through 34088, −32768 through −31448 are within the system area. Therefore, they cannot be specified (an OM error will occur).

The usable address ranges for the various RAM capacities are shown below. It is not necessary to create data areas for programs that do not use PEEK and POKE. Therefore, when a data area has been created, release it using a CLEAR command as shown in the table below.

RAM capacity	Usable address (decimal)	When data area is not created
8KB	0~8191	CLEAR 8192
16KB	0~16383	CLEAR 16384
24KB	0~24575	CLEAR 24576
32KB	0~32767	CLEAR 32767

In this way the program area and variable area can be used to their fullest extent. Using a CLEAR command in which the data area address is not specified does not change the size of the data area.

* If a data area specification is released using NEW ALL ↵, the address number is not displayed.

SEE SYSTEM, PEEK, POKE, NEW ALL

CLS

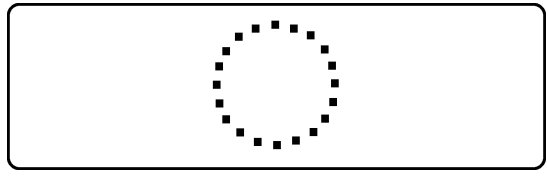
Function	Clears all displays and moves the cursor to the home position (top left corner).
Format	CLS

The CLS command is used to clear the screen and to move the cursor to the home position at the top left corner.
It is used to clear the screen for a graphic display.
(1) When this command is manually executed, the cursor is displayed at the (0,1) position.
(2) When this command is executed during program execution, the cursor is displayed at the (0,0) position.

Usage

```
 5 ANGLE 0
10  CLS
20 FOR I=0 TO 360 STEP 12
30 DRAW(SINI*15+80,COSI*15+15)
40 NEXT I
50 END
```

This program is used to display the following pattern.



When a pattern is drawn on the screen as shown above, the screen must be cleared before drawing starts. Therefore, the CLS command is used at the beginning of the program.

DIM

Function	Declares an array.
Format	DIM array variable name (subscript) [, array variable name (subscript)]

The DIM command declares an array of the specified name in the memory area. Variables assigned by DIM (called array variables) include single-precision numerical arrays, half-precision numerical arrays and string arrays.

The general format of DIM can be indicated as follows.

DIM array variable name (subscript [, subscript]) [, array variable name (. . . . . .
(Maximum value of subscript : 255)

Examples of declaration statements for various arrays are provided as follows.

Format	Classification	Example
DIM array variable name (I)	One-dimensional single-precision numerical array	DIM A (5)
DIM array variable name (I, J)	Two-dimensional single-precision numerical array	DIM A (2, 3)
DIM array variable name ! (I)	One-dimensional half-precision numerical array	DIM A! (5)
DIM array variable name ! (I, J)	Two-dimensional half-precision numerical array	DIM A! (2, 3)
DIM array variable name$(I)*N	One-dimensional string array	DIM A$ (5)*20
DIM array variable name$(I, J)*N *N can be omitted	Two-dimensional string array	DIM A$(2, 3)*20

I, J and N are real numbers or numerical expressions with the ranges of $0 \leqq I < 256$, $0 \leqq J < 256$ and $0 \leqq N < 80$, in which the fractional part of numerical value is discarded.
An array variable name is one character from among the capital alphabetical characters from A to Z.
The maximum dimensional value is 2, which means one-dimensional arrays and two-dimensional arrays can be specified.
A half-precision numerical array can be specified by placing " ! " just after the array variable name, and a string array can be specified by placing " $ " just after the array variable name.
A string array, in which a character string of "N" length can be assigned, is declared by placing " *N". However, if " *N" is omitted, N=16 (i.e. an array in which 16 character long strings can be stored).

Usage

Enter the following program and run it.

```
10  CLEAR
20  DIM A(2,3), B(2,3)
100 END
```

An operational expression may or may not be written between line 20 and line 100.
When characters such as Ready P0 appear after program execution, check the array variable list as follows.

```
LIST   V ↵
A( )   B( )
Ready  P0
```

The array variable name assigned by DIM can be confirmed using

```
LIST  V ↵
```

Let's check the content of each array variable by adding the following list to the program mentioned above. An array variable must be declared by DIM before it is used.

```
30 FOR I=0 TO 2
40 FOR J=0 TO 3
50 PRINT A(I,J);B(I,J);
60 NEXT J:NEXT I
```

When you run the program after adding lines 30 to 60, the following is displayed.

■ **Execution Example**

```
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  0
Ready P0
```

Here, the contents of 24 array variables, A (0, 0), B (0, 0) through A (2, 3), B (2, 3), are displayed as "0".
It is important to note that the contents of all the arrays become "0" when the DIM command is executed.
While the contents of the numerical arrays mentioned above are 0, when string arrays are declared their contents become null-strings in which nothing is displayed. Null-string means that a character string is empty.
The difference between space-strings and null-strings should be noted. A space-string is a string that has one space (A$(I)="␣") and a null-string is a string that is empty (A$(I)="").

SAMPLE PROGRAM 1

* Rearrange (sort) program (one-dimensional numerical array).

```
10 CLEAR
20 DIM D(5)
30 FOR I=1 TO 5
40 PRINT "DATA";I;" =";:INPUT D(I)
50 NEXT I
60 REM SORT
70 F=0
80 FOR I=1 TO 4
```

```
 90 IF D(I)<D(I+1) THEN X=D(I):D(I)=D(
    I+1):D(I+1)=X:F=1
100 NEXT I
110 IF F=1 THEN 70
120 REM RESULT
130 FOR I=1 TO 5
140 PRINT D(I);
150 NEXT I
160 END
```

This program enters 5 numerical data and arranges these data in a sequence with the largest value first.
An array is provided from D(0) through D(5) by the DIM command in line 20, but only D(1) through D(5) are used in this program.
Lines 60 to 110 contain the sort program while lines 120 to 150 contain the program which displays the sorted result with the largest value first.
It is convenient to use array variables by combining them with the FOR-NEXT command as shown in the sample program.

SAMPLE PROGRAM 2

* Vertical and horizontal totalization (Two-dimensional numerical array)

```
 5 CLEAR
10 DIM A(3,3),X(3),Y(3)
20 FOR I=1 TO 3
30 FOR J=1 TO 3
40 PRINT "(";I;",";J;")";"="
50 INPUT A(I,J)
60 NEXT J:NEXT I
70 REM SUBTOTAL
```

```
 80 FOR I=1 TO 3
 90 FOR J=1 TO 3
100 X(I)=X(I)+A(I,J)
110 Y(J)=Y(J)+A(I,J)
120 NEXT J:NEXT I
130 REM RESULT
140 FOR I=1 TO 3
150 PRINT "X(";I;")=";X(I)
160 PRINT "Y(";I;")=";Y(I)
170 FOR K=1 TO 500:NEXT K
180 NEXT I
190 END
```

This program assigns the data in Table 1 to a two-dimensional array as shown in Table 2 in order to obtain vertical and horizontal subtotals X(1), X(2), X(3), and Y(1), Y(2), Y(3).

Table 1

14	9	21	X (1)
35	4	53	X (2)
6	15	11	X (3)
Y (1)	Y (2)	Y (3)	

Table 2

A (1, 1)	A (1, 2)	A (1, 3)	X (1)
A (2, 1)	A (2, 2)	A (2, 3)	X (2)
A (3, 1)	A (3, 2)	A (3, 3)	X (3)
Y (1)	Y (2)	Y (3)	

Lines 80 to 120 obtain a subtotal while lines 130 to 180 display the obtained values.

SEE CLEAR, NEW ALL, ERASE, LIST V

DRAW/DRAWC

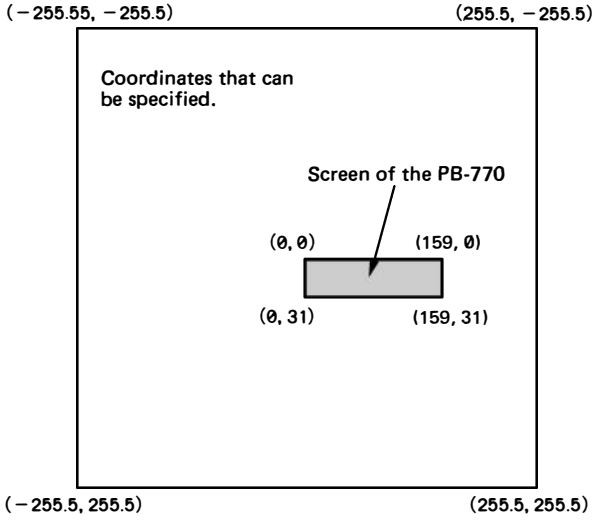
Functions	DRAW: Draws a dot. DRAWC: Clears a dot.
Formats	DRAW (X1, Y1) [–(X2, Y2)] DRAWC (X1, Y1) [–(X2, Y2)]

The DRAW command draws dots or lines on the screen while DRAWC clears them.
Since not only characters but dots and lines can be displayed on the screen, many kinds of graphs, etc. can be made with this command.

The ranges of coordinates that can be specified by DRAW (X, Y) or DRAWC (X, Y) are as follows.

$$-255.5 < X < 255.5 \qquad -255.5 < Y < 255.5$$

Since the range of the screen dot coordinates are $0 \leqq X \leqq 159$, and $0 \leqq Y \leqq 31$, the virtual screen shown below can be considered.



In this figure, ▭ is equivalent to the screen of the PB-770.
The top left corner of the screen is the origin (0, 0).

Usage

DRAW and DRAWC are used as follows.

DRAW (X_1, Y_1)	Draws a dot at coordinate (X_1, Y_1).
DRAWC (X_1, Y_1)	Erases the dot at coordinate (X_1, Y_1).
DRAW (X_1, Y_1)–(X_2, Y_2)	Draws a line from coordinate (X_1, Y_1) to (X_2, Y_2).
DRAWC (X_1, Y_1)–(X_2, Y_2)	Erases the line from coordinate (X_1, Y_1) to (X_2, Y_2)

X and Y mentioned above are numerical values, variable names, and numerical expressions with the following range.

$$-255.5 < X < 255.5 \qquad -255.5 < Y < 255.5$$

They are shown on the actual screen rounded off to integers.
The following program draws a rectangle on the screen.

```
10 REM DRAW
20 CLS
30 DRAW(10,10)-(10,20)-(150,20)-(150,10)-(10,10)
40 END
```

A figure can be drawn by providing continuous coordinates with "–" as shown in the program above.

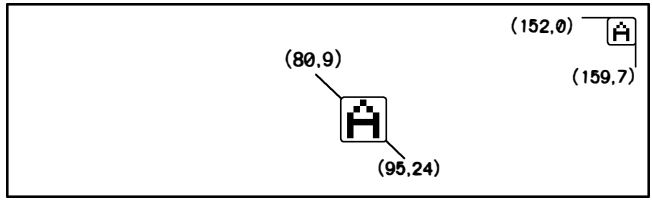
SAMPLE PROGRAM

* A program that displays a character magnified two times.

```
10 CLEAR :DIM A(7,7):CLS
20 K$=INKEY$
30 IF K$="" THEN 20
40 LOCATE 19,0:PRINT K$
50 FOR I=0 TO 7
60 FOR J=0 TO 7
70 A(I,J)=POINT(I+152,J)
80 NEXT J:NEXT I
90 FOR I=0 TO 7
```

```
100 FOR J=0 TO 7
110 IF A(I,J)<>1 THEN 160
120 DRAW(2*I+80,2*J+9)
130 DRAW(2*I+81,2*J+9)
140 DRAW(2*I+80,2*J+10)
150 DRAW(2*I+81,2*J+10)
160 NEXT J:NEXT I
170 LOCATE 0,1
180 END
```

This program uses INKEY$ to enter one character, and first displays the character at the position which is specified by LOCATE (19,0) as shown in the figure below.
This character is displayed using dots in a square area with a diagonal line formed by (152,0)–(159,7) in which the dots that are displayed and cleared are checked with the POINT function. Then values 0 and 1 are assigned to array variable A (I, J).
Using this array variable as data, the character that is magnified is displayed in a square area with a diagonal line formed by (80,9)–(95,24).



SEE POINT

END

Function	Terminates execution of a program.
Format	END

The END command terminates execution of a program. The nesting stack (control of FOR-NEXT loop and GOSUB) is cleared by the execution of an END command. As many END statements as desired can be placed anywhere in a program.
The END command placed at the end of a program can be omitted.

SAMPLE PROGRAM

```
500  FOR I=0 TO 1000
510  K$=INKEY$
520  IF K$="A" THEN 1000
530  IF K$="B" THEN 2000
540  IF K$="C" THEN 3000
550  IF K$="D" THEN END
560  NEXT I
570  END
 ⋮    ⋮
```

This program is used as part of a menu program.
After "A", "B" or "C" is entered, execution jumps to line 1000, 2000, or 3000 respectively.
When "D" is entered, program execution is immediately terminated. However, when key entry is performed with a key other than "A"~"D", or when no key entry is performed, execution is automatically terminated after a certain period of time.

SEE STOP

ERASE

Function	Releases registered variables and array variables.
Format	ERASE variable name [, variable name]

An ERASE command can release registered variables and array variables that can be confirmed by LIST V.
Specification of a variable name for release is performed as follows.

Variable names
displayed by a LIST V: AB, AB$, A ( ), A! ( ), A$ ( )

ERASE AB, AB$, A, A!, A$

The nesting stack is cleared by execution of the ERASE command, so this command should never be used within a FOR-NEXT loop. When an unregistered variable name is specified, execution proceeds to the next operation.

SEE DIM, LIST V

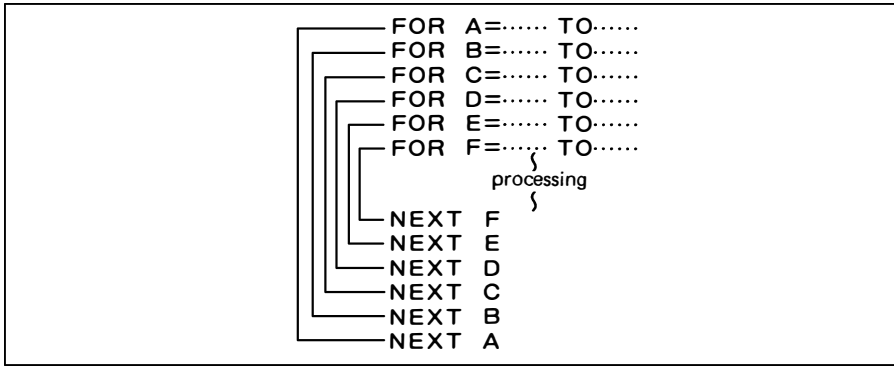
FOR~TO~STEP/NEXT

Function	Repeats the execution from FOR to NEXT for a specified number of times.
Formats	FOR numerical variable = i TO j [STEP k] NEXT numerical variable (same variable as that of the FOR statement) i : Initial value j : Final value k: Increment

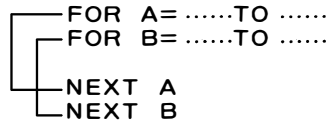
The FOR-NEXT command repeatedly executes each statement between the FOR command and the NEXT command for the specified number of times.

This command has some restrictions as follows.

(1) Variables must be numerical variables.
(2) STEP k can be omitted. In this case +1(STEP 1) is set as the increment.
(3) A variable or a numerical expression can be used for the initial value i, final value j, and increment k.
(4) If $k > 0$ and $i > j$, FOR-NEXT execution is performed only once.
(5) If $k < 0$ and $i < j$, FOR-NEXT execution is performed only once.
(6) When a FOR-NEXT execution is performed and execution proceeds to the following line, the control variable becomes the value of $i + nk$ which is larger than j (n = integer).
(7) Nesting of FOR-NEXT loop can be performed up to 6 levels.



(8) If the interval of a FOR-NEXT loop is crossed, an error (FO error) occurs.



FO error occurs.

(9) When FOR is not executed and NEXT appears, an error (FO error) occurs.
(10) The variable for the NEXT command cannot be omitted.
(11) If ERASE or CLEAR is executed, an FO error occurs for the following NEXT statement because the FOR-NEXT nesting stack is cleared.
(12) The values of the control variables are cleared before the first pass of all loops. Therefore, the same control variable can be used for separate (non-nested) loops within the same program without clearing the control variable each time.
(13) A jump into a loop using GOTO or GOSUB statement, etc. cannot be performed.

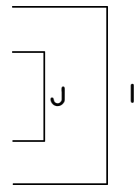
Usage

The following program is used to demonstrate the change of the variable depending on the value of the initial value, final value, and increment.

```
10  INPUT "FOR I=",I,"TO",J,"STEP",K
20  PRINT "FOR I=";I;"TO";J;"STEP";K
30  FOR A=I TO J STEP K
40  PRINT "VARIABLE:A=";A
50  FOR X=1 TO 100:NEXT X
60  NEXT A
70  PRINT :GOTO 10
```

Several execution examples are provided below.
In the following program, the FOR-NEXT loop has two levels.

```
10  CLEAR
20  DIM A(9,9)
30  FOR I=1 TO 9
40  FOR J=1 TO 9
50  A(I,J)=I*J
60  NEXT J
70  NEXT I
```



In this example, multiplication is performed and the result is assigned to array variable A(I, J) between the two FOR-NEXT loops.
Since execution is performed 9 times by the internal FOR-NEXT loop and execution is performed 9 times by the external FOR-NEXT loop, a total of 9 x 9 operations are performed.
It should be noted in the above example that the internal loop (J) is totally enclosed by the external loop (I). This is required whenever loops are nested. (See item 7 on page 172.)
This also applies to the levels from 3 to 6.

The following program changes the operation by jumping from the loop depending on the arithmetic result of the FOR-NEXT statement.

```
10  CLEAR
20  FOR I=1 TO 100
30  X=X+I
40  IF X>=1000 THEN 100
50  NEXT I
60  END
100 PRINT I;X
110 END
```

This program performs the addition of 1 + 2 + 3 . . . . . , and when the result exceeds 1000, it jumps to line 100 and displays the value of the variable at that time along with the addition result.

Jumping out of a loop using an IF-THEN command can be used.
Jumping out of a loop once using a GOSUB command and returning with a RETURN command to continue execution can also be used.

The following program provides an example of when there is nothing to execute between FOR-NEXT.

```
10  LOCATE 8,2 : PRINT "HIT!";
20  FOR I=0 TO 50
30  NEXT I
40   CLS
50  FOR I=0 TO 50
60  NEXT I
70  GOTO 10
```

There is no statement to be processed between the FOR-NEXT loop in lines 20 and 30 and lines 50 and 60. However, the FOR-NEXT command is executed the specified number of times even in this case.
This command which seems to be nonsense is often used to kill time and is called a "wait loop".
In this program, the characters "HIT!" are repeatedly displayed for a certain period of time, erased for a certain period of time, and displayed again for a certain period of time.
As the final value of the variable increases, the waiting time becomes longer.

SAMPLE PROGRAM

* A program that provides a display increasing the number of " * " by one on each successive line.

```
10   CLS
20 N=1
30 FOR I=1 TO N
40 PRINT "*";
50 NEXT I
60 PRINT
70 N=N+1
80 IF N>=20 THEN END  ELSE 30
```

This program increases the final value of the FOR-NEXT command by one each time and increases the number of "*" displayed by one each time. The execution example is as follows.

```
*
**
***
****
*****
******
```

SEE IF~THEN~ELSE

GET

Function	Reads data stored on a cassette tape to a variable.
Formats	GET variable [, variable] GET "file name" variable [, variable]

The GET command is used to read data stored on a cassette tape by the PUT command to a variable.
The file name can be omitted as shown in the above format, but, in that case the first file that appears on the cassette tape (MT) is read. The data that are read can be sequentially assigned to different variables by punctuating the variables with commas.
However, when data is assigned to a numerical variable, the space at the beginning of the data is ignored.

Usage

The following program stores the values of variables A, B, C and D with the file name "TEST" on a cassette tape.

```
10 REM PUT MT
20 A=10 : B=20 : C=30 : D=40
30 PUT "TEST" A, B, C, D
40 END
```

The following example program uses a GET command to read data stored on a cassette tape with the file name "TEST".

```
10 REM GET MT
20 GET "TEST" E, F, G, H
30 PRINT E ; F ; G ; H
40 END
```

In this example, the data A, B, C, and D are read into the variables E, F, G, and H. An important item is that data are stored on cassette tape in an A, B, C, D sequence.
Although data are read in an A, B, C, D sequence during cassette tape playback, the data of A is assigned to E, and data of B is assigned to F, ... sequentially.
If GET is executed with variables that exceed the variable data stored on a cassette tape, an error (DA error) occurs because of data shortage.

Usage

* A program to store a person's name and data on cassette tape, and to read and display a person's data when a name is entered. (Number of characters of name and data are both limited to 16 or less.)

```
 10 CLEAR
 20 DIM N$(10),D$(10)
 30 INPUT "PUT:O GET:I END:E";M$
 40 IF M$="O" THEN GOSUB 100
 50 IF M$="I" THEN GOSUB 300
 60 IF M$="E" THEN END
 70 GOTO 30
100 I=0
110 INPUT "NAME:";N$(I)
120 PUT N$(I)
130 IF N$(I)="0" THEN 180
140 INPUT "DATA:";D$(I)
150 PUT D$(I)
160 I=I+1
170 IF I<10 THEN 110
180 RETURN
300 J=0
310 INPUT "NAME:";A$
```

```
320 GET N$(J)
330 IF N$(J)="0" THEN PRINT "NOT FOUND
    ":GOTO 380
340 GET D$(J)
350 IF A$<>N$(J) THEN 370
360 PRINT A$;"=";D$(J):GOTO 380
370 J=J+1:GOTO 320
380 RETURN
```

This program consists of two subroutines.
Lines 100 to 180 provide a subroutine to store data on a cassette tape, while lines 300 to 380 provide a subroutine to search for and read data from a cassette tape.
A name is entered in array variable N$(I), and data is entered in D$(I). Then storing and reading are performed with these two variables as a pair.
When "O" is entered for the menu in line 30, the storing subroutine from line 100 is executed with the input of a name and data continuously performed. Up to 11 data inputs can be performed. However, since it is necessary to attach "0" to the end of the last name (to indicate data end), up to 10 data inputs can actually be performed.
When "I" is entered for the menu in line 30 to search for the name of a person, then the name and data are read from a cassette tape.
After the name is found, the name and data are displayed and the program returns to the menu in line 30.
Enter "E" for the menu to terminate the program.

SEE PUT, SAVE

GOSUB/RETURN

Function	Causes a jump to a subroutine and a return to a main program.
Formats	GOSUB line number　$1 \leqq$ line number < 10000 GOSUB PROG n　$0 \leqq n < 10$ RETURN　(n indicates the program area No.)

The GOSUB command causes a jump to a subroutine of a line which is specified by a variable or a numerical expression. The RETURN command causes a return from the subroutine to the main program.

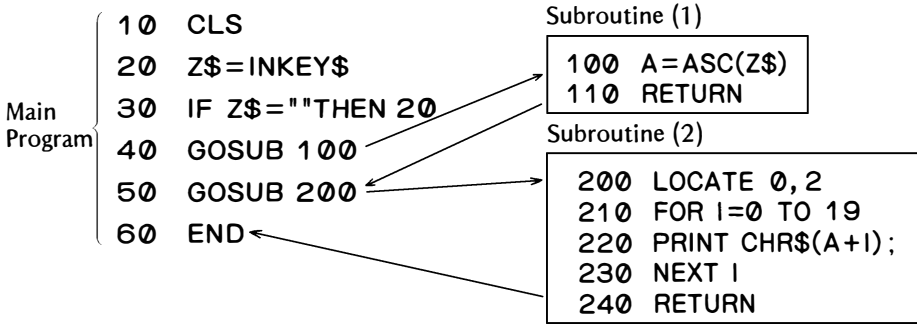
Example

	Line number	Program area
Numerical constant	GOSUB 200	GOSUB PROG 4
Numerical variable	GOSUB N	GOSUB PROG L
Numerical expression	GOSUB N＊5	GOSUB PROG L/10

The PB-770 allows a jump to a subroutine and a return to a main program in other program areas, as well as a jump to a subroutine in the same program area as shown by the above formats.

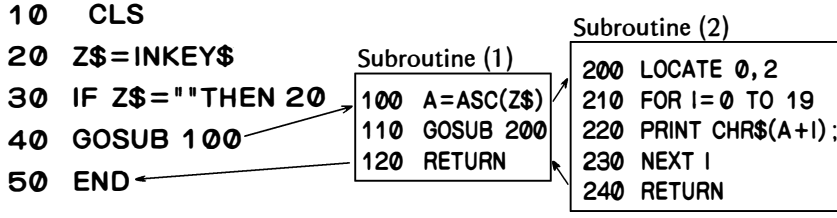
Usage

In the following program, the execution flow is shown by changing the layout.



Lines 10 to 60 provide the main program, lines 100 and 110 provide one subroutine, while lines 200 to 240 provide another subroutine.
When the RETURN command is executed as shown by the arrows, a return is made to the statement following the GOSUB command and execution of the main program continues.

This program can be rewritten as follows by changing its format slightly.



Although the contents of this program is exactly the same as the previously mentioned program, this program has a double structure in which subroutine (2) is included in subroutine (1).
Up to 12 nesting levels can be used with the GOSUB/RETURN commands.
Precautions should be taken concerning the following items when GOSUB/RETURN commands are used.

(1) One subroutine must have at least one RETURN command, and can have as many as desired.
(2) If the line specified by a GOSUB command does not exist, an error (UL error) will occur.
(3) When GOSUB PROG *n* is executed, if a program is not written in program area *n*, an error (UL error) will occur in the present program area.
(4) When there is no GOSUB command for a RETURN command, an error (GS error) will occur.

(5) Up to 12 GOSUB nesting levels can be used. (If there are 13 levels or more, an error (NO error) will occur.)
(6) When a fraction is included in a line number or program area number specified with GOSUB, execution is performed with the fraction discarded.

SAMPLE PROGRAM

* Slot machine game program.

```
 10  CLS
 20 GOSUB 200
 30 GOSUB 300
 40 GOSUB 400
 50 LOCATE 3,0
 60 PRINT "POINT";N
 70 LOCATE 0,0:END
200 X=INT(10*RND)
210 Y=INT(10*RND)
220 Z=INT(10*RND)
230 RETURN
300 LOCATE 6,2
310 PRINT X;"  ";Y;"  ";Z;
320 RETURN
400 IF X=Y THEN IF Y=Z THEN N=100:RETU
    RN
410 IF X=Y THEN N=40:RETURN
420 IF Y=Z THEN N=30:RETURN
430 IF X=Z THEN N=20:RETURN
440 N=0
450 RETURN
```

This program displays 3 numerals. If all of the 3 numerals are identical, 100 points are given, and if 2 numerals are identical, 40, 30 or 20 points are given depending on their locations.
If all 3 numerals are different, 0 points are displayed.
Lines 200 to 230 provide the subroutine that generates the 3 numerals with the RND function, while lines 300 to 320 provide a subroutine that displays the numerals at the center of the screen, and lines 400 to 450 provide a subroutine that checks the number of points given.
Lines 10 to 70 provide the main program which simply controls each subroutine and displays the number of points given.

SEE GOTO

GOTO

Function	Causes an unconditional jump to a specified line.	
Formats	GOTO line number	$1 \leqq$ line number < 10000
	GOTO PROG n	$0 \leqq n < 10$ n : Indicates the program area No.

The GOTO command unconditionally jumps to a specified line number or to the beginning of a program in another program area specified by a variable or a numerical expression.
If the specified line number does not exist, or there is no program in the specified program area, an error (UL error) will occur.
If a fraction is included in a specified line number or in a specified program area number, the fraction is discarded when execution is performed.

Example

	Line number	Program area
Numerical constant	GOTO 500	GOTO PROG 4
Numerical variable	GOTO N	GOTO PROG N
Numerical expression	GOTO N＊5	GOTO PROG N/10

SAMPLE PROGRAM

* Simplified Digital Clock Program.

```
10 INPUT "H=";H,"M=";M,"S=";S
20  CLS
30 IF H<10 THEN LOCATE 8,0:PRINT H;":
   ";:GOTO 50
40 LOCATE 7,0:PRINT H;":";
50 IF M<10 THEN LOCATE 12,0:PRINT M;"
```

```
    :";:GOTO 70
 60 LOCATE 11,0:PRINT M;":";
 70 IF S<10 THEN LOCATE 16,0:PRINT S;"
    :";:GOTO 90
 80 LOCATE 15,0:PRINT S;
 90 FOR I=2 TO 120:NEXT I
100 S=S+1
110 IF S>=60 THEN 200
120 GOTO 70
200 S=0:M=M+1
210 IF M>=60 THEN 300
220 GOTO 50
300 M=0:H=H+1
310 IF H>=24 THEN H=0
320 GOTO 30
```

When you run this program, H (hour), M (minutes), and S (seconds) are requested. After the present time is entered and the ↵ key is pressed, time is continually displayed up to seconds.
Since the internal clock of the microcomputer is not used for this program, it does not show the exact time, but indicates the approximate time.
If it gains, adjust by increasing the final value of the FOR command in line 90, and if it loses, adjust by decreasing the final value.
Many GOTO commands are used in this program.
The program creates an infinite loop using GOTO commands in lines 120, 220, and 320.

SEE GOSUB/RETURN, IF-THEN-ELSE

IF~THEN~ELSE

Function	Executes the contents after THEN or ELSE depending on the condition after IF.
Formats	IF conditional expression THEN {line number / command} ELSE {line number / command} A numerical expression cannot be used as a line number.

The IF-THEN command performs a conditional jump while a GOTO command performs an unconditional jump.
When the conditional expression is true, the THEN statement is executed, and when it is false, the ELSE statement is executed. If there is not an ELSE statement, the next line is executed.
A line number that represents a branch destination, or a command statement can be inserted in THEN and ELSE statements.
Multistatements can be performed in THEN or ELSE statements using colons as shown below.

IF conditional expression THEN ~ : ~ : ~ ELSE ~ : ~ : ~

THEN statement — ELSE statement

An IF statement can be inserted in the THEN or ELSE statement as shown below.

IF conditional expression (1) THEN IF conditional expression (2) THEN ~ ELSE ~ ELSE ~

Executes when conditional expression (1) is true and when conditional expression (2) is false.

Executes when conditional expression (1) is false.

Executes when conditional expressions (1) and (2) are both true.

Usage

Examples of IF ~ THEN ~ ELSE statement usage are shown below. These programs check whether a triangle can be made or not after three sides are entered.

(1)
```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN 40
30 GOTO 50
40 IF ABS(A-B)<C THEN 60
50 PRINT "NOT TRI" : GOTO 10
60 PRINT "TRI" : GOTO 10
```

(2)

```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN IF ABS(A-B)<C THEN 40
30 PRINT "NOT TRI" : GOTO 10
40 PRINT "TRI" : GOTO 10
```

(3)

```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN IF ABS(A-B)<C THEN 40
30 PRINT "NOT TRI" : GOTO 10
40 IF A=B THEN IF B=C THEN PRINT "E.TRI" : GOT
   O 10 ELSE PRINT "I.TRI" : GOTO 10
50 IF B=C THEN PRINT "I.TRI" : GOTO 10
60 IF A=C THEN PRINT "I.TRI" : GOTO 10 ELSE
   PRINT "TRI" : GOTO 10
```

Example programs (1) and (2) check whether a triangle (TRIangle) is formed or not (NOT TRIangle) when three sides A, B, and C are entered. In example(1), two triangle conditions (A+B>C, IA−BI<C) are checked in separate lines (line 20 and line 40), while in example (2), they are checked in one line (line 20).
Also, in example(3), the program checks whether the values form an equilateral triangle (Equilateral TRIangle) or an isosceles triangle (Isosceles TRIangle), as well as whether they form a triangle (TRI) or not (NOT TRI).
In example (3), whether a triangle (TRI) is formed or not is checked first in line 20. If a triangle is not formed, "NOT TRI" is displayed and a return is made to line 10.
If a triangle is formed, whether the three sides are equal or not is checked in line 40. If the three sides are equal, "E.TRI" is displayed. If two sides are equal, "I.TRI" is displayed, and if all three sides are not equal, "TRI" is simply displayed in lines 40 to 60, and a return is made to line 10.

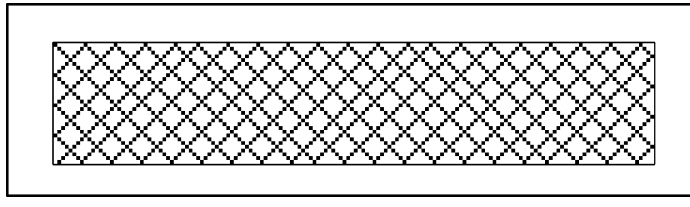
SAMPLE PROGRAM

* Program to draw a pattern on the screen with dots.

```
 10  CLS
 20 X=0:Y=0:N=1:M=1
 30 DRAW(X,Y)
 40 X=X+N:Y=Y+M
 50 IF X>=158 THEN N=-1
 60 IF Y>=31 THEN M=-1
 70 IF X<=0 THEN N=1
 80 IF Y<=0 THEN M=1
 90 IF X=1 THEN IF Y=31 THEN BEEP 1:END
100 GOTO 30
```

This program determines whether or not the value of the dot coordinates (X, Y) on the screen is within the screen limitation by the IF-THEN command (lines 50 to 80) to control whether the values of the X, Y coordinates are increased or decreased.
When the values of X and Y become X = 1, Y = 31 (line 90), a beep sound is generated, and the program is terminated.
The next figure shows the execution result.

■ **Execution Example**



INPUT

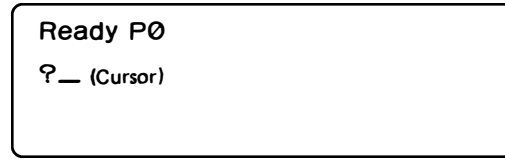
Function	Requests data entry (numerical value, character) from the keyboard to a variable.
Formats	INPUT variable [, variable] INPUT "prompt", variable [, "prompt", variable] INPUT "prompt"; variable [, "prompt"; variable]

The INPUT command is used to enter data from the keyboard to a variable. The basic formats of the INPUT statement are as follows.

```
Example 1    INPUT A
Example 2    INPUT X, Y, Z
Example 3    INPUT "AGE", A
Example 4    INPUT "NAME"; A$
Example 5    INPUT "X="; X, "Y="; Y
```

When the INPUT command is executed, the PB-770 displays an input request message and waits for data input.
For example, when example 1 is executed, " ? " is displayed as follows and the cursor turns on and off at the right of " ? ". The data input setup has been completed.

Display during INPUT command execution.



Data input is performed by pressing keys. Always press the [ENTER] key or [↵] key at the end of data input. It should be noted that the [ENTER] key functions the same as the [↵] key during INPUT statement execution.

■ **Variables that can be used in an INPUT statement are as follows.**

	[Examples]
Numerical variable	INPUT X
Character variable	INPUT X$ (Up to 7 characters can be entered.)
Registered variable	INPUT XY INPUT XY$ (Up to 16 characters can be entered.)
Array variable	INPUT X! (*i*), Y! (*i, j*) Half-precision numerical array INPUT X (*i*), Y (*i, j*) Single-precision numerical array INPUT A$ (*i*), INPUT A$ (*i, j*) String array

Usage

(1) Numerical value input

Let's check INPUT statement usage and functions by using a simple program.

10	INPUT A	Input command. Provides data input to variable A.
20	PRINT A	Output command. Displays the contents of variable A.
30	GOTO 10	Jump command. Moves program execution to line 10.

After inputting this program, enter RUN↵ to execute it. "?" is displayed. Now, enter 3.6↵. If the entry is performed correctly, the same numerals are displayed again by the PRINT statement as follows.

CLS RUN↵ 3.6↵

```
 RUN
?3.6
  3.6
?
```

A value can be entered by inputting a calculation expression that results in the value, but this is limited to INPUT statements using numerical variables.
Let's confirm this using the previous program.

CLS RUN ↵ 100−20/5 ↵

```
RUN
?100-20/5
 96
?_
```

(2) Character input
Perform character input by changing the program of (1). The new program is as follows.

```
10 INPUT A$
20 PRINT A$
30 GOTO 10
```

When character input is performed, a character variable is used as shown above. When you execute this program, input is requested by the display of " ? " the same as for numerical value input. When entering the character string ABC, the display becomes as follows.

CLS RUN ↵ ABC ↵

```
RUN
?ABC
ABC
?_
```

When the numerical value 123 is entered, 123 is displayed. However, it should be noted that this 123 is a character string and not a numerical value.
When data is input to a character variable, it is unnecessary to enclose the character string with " ". If " " is used, the quotation marks would also be entered as character data.

■ **Data that can be input to each variable by an INPUT statement.**

A. Numerical variable
   a. $\pm 1 \times 10^{-99}$ to $\pm 9.99999999999 \times 10^{99}$ and 0
   b. Operational expression for a numerical value (Example: 200 x (5 + 2))
   c. Numerical variable from A to Z (Fixed variable)
   d. Registered variable
   e. Array variable

B. Character variable
   a. Fixed character variable . . . . . Up to 7 characters and symbols.
   b. Registered character variable . . . . Up to 16 characters and symbols.
   c. Array variable . . . . . Up to 79 characters and symbols.

(3) Multiple variable input

Multiple variables can be used in an INPUT statement. (Multiple INPUT statements can be arranged in one statement as shown below.)

```
10 INPUT X
20 INPUT Y  ⟶ 10 INPUT X, Y, Z
30 INPUT Z
```

Punctuate variables with commas.

When you execute this INPUT statement, " ? " is displayed at first to request the input of the value of X. After the value of X is entered, the values of Y and Z are requested in turn. After the value of Z is entered, this INPUT statement is terminated.

Variables, such as numerical variables, character variables, etc. can be used with all combinations and sequences in this kind of INPUT statement as shown below.

```
10 INPUT A$, X
```

However, a " , " (comma) must be used for punctuation between variables.

(4) INPUT statement that displays a message

If a character string enclosed with " " is inserted between INPUT and a variable, the character string is displayed as it is. This is called a prompt. Incorrect input can be reduced by clarifying data for input with this message.
Input the following INPUT statement and execute it.

```
10 INPUT "AGE" ; A
```

Then the following is displayed.

CLS RUN ↵

```
RUN
AGE?_
```

Next, this INPUT statement is changed as follows.

```
10 INPUT "AGE", A
```

When you run this program, the display is as follows.

CLS RUN ↵

```
RUN
AGE_
```

When " ? " (input request display) should appear after a message, a " ; " is used for punctuation between the message and the variable.
This INPUT statement can be changed for input of two or more variables.

```
10 INPUT "HEIGHT=" ;X, "WEIGHT=";Y
```

In effect, this combines the following two INPUT statements into one INPUT statement.

```
10 INPUT "HEIGHT=";X
20 INPUT "WEIGHT=";Y
```

→

```
10 INPUT "HEIGHT=";
   X, "WEIGHT=";Y
```

When you run this program, the display is as follows.

CLS RUN ↵

1 7 5 ↵

6 5

```
RUN
HEIGHT=?175
WEIGHT=?65_
```

■ **Number of characters in character string used for a message**

The maximum number of characters for a message is 79 including the line number and INPUT command.

LET

Function	Assigns data to a variable.
Format	LET variable = expression

The LET command placed at the beginning of an assignment statement is generally used in the following formats.

(Example 1) `LET A=10` `LET A$="GAME"`

(Example 2) `LET X=SIN (S-PI/4)` `LET X$=A$+B$`

The assignment statement assigns the value of the expression on the right side of the = sign to the variable on the left side of the = sign. A numerical expression corresponds to a numerical variable, and a character expression corresponds to a character variable. If the correspondence is not correct, a TM error is displayed.

- **Numerical value range**
  A numerical value can be assigned to a numerical variable within the following range.

$$-10^{100} < \text{numerical value} < 10^{100}$$

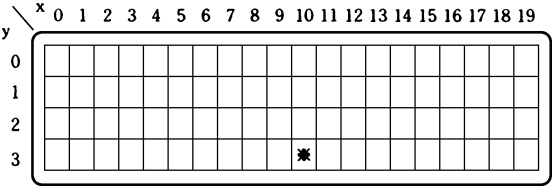
- **Character string range**
  When the left side is a fixed character variable — up to 7 characters.
  When the left side is a registered character variable — up to 16 characters.
  When the left side is a character array variable — up to 79 characters.
- **LET can be omitted.**
  `10 LET A=1` is the same as `10 A=1`

LOCATE

Function	Specifies the cursor position.
Format	LOCATE X, Y $0 \leqq X < 20,\ 0 \leqq Y < 4$

The display screen of the PB-770 is provided with 20 x 4 display positions as shown below. Display is generally performed from the left end of the screen by executing a PRINT statement. However, the display position can be freely changed using the LOCATE command.

For example, when you specify LOCATE 10, 3, the display position is specified at (10, 3).



However, when 3 is specified for the Y coordinate, the display is scrolled. To avoid scroll except for LOCATE 19,3 (lower right corner), use a semicolon at the end of the PRINT statement.

Usage

List A

```
10  X=1
20  X=X+1
40  PRINT "X=";X
50  GOTO 20
```

List B

```
10  X=1
20  X=X+1
30  LOCATE 0,0
40  PRINT "X=";X
50  GOTO 20
```

When you execute List A, the following is displayed.

```
X=2
X=3
X=4
X=5
```

⇧ Numerals appear continuously from the bottom of the screen and disappear toward the top of the screen.

When you add line 30 as shown in List B, the display is changed in such a way that only the "X = Numeral" is gradually increased at the top left corner of the screen.

PB-770 Only

POKE

Function	Writes data to a specified address.
Format	POKE address, data

The POKE command writes data to a specified address. Both the address and the data are specified by integers. (Fractional values are discarded.) Addresses and data must be within the following ranges.

$-32769 < \text{address} < 65536$
$0 \leq \text{data} < 256$

See CLEAR for information on addresses.

CAUTION:
Never write data using POKE in the system area (&H0000 through &H0528) or in a user area where programs have been stored. Otherwise, the computer will not work normally especially when data are written in the system area. In this case switch the power OFF and then ON again, and enter NEW ALL↵ to clear all the programs and variable contents.

SAMPLE PROGRAM

Write data to a specified address using the following program.

```
10 REM POKE EXAMPLE
20 INPUT "-32769<ADDRESS<65536", A
30 INPUT "0<=DATA<256", D
40 POKE A, D
50 PRINT PEEK A
```

SEE CLEAR, PEEK

PRINT/LPRINT

Functions	PRINT: Performs output to the display. LPRINT: Performs output to the printer.	
Formats	Display output	PRINT expression [, expression] PRINT expression [; expression] PRINT $ registered character variable
	Printer output	LPRINT expression [, expression] LPRINT expression [; expression]

The PRINT and LPRINT commands are almost the same with the only difference being that output is either to the display or to the printer. However, when they are used with the TAB function, some differences occur.

Usage

Different kinds of data such as characters, numerical expressions, all types of graphic data, and numerical values can be displayed using a PRINT statement.
As an example, it can be used as follows.

```
PRINT 1.414↵
PRINT A * B-2↵ . . . . .
```
Since A and B are variables, the result is displayed.

When these PRINT statements are executed, line change is performed after data is displayed. Run the following program as an example.

```
10 A=0 : B=3
20 PRINT 1.414
30 PRINT A*B-2
40 INPUT C
```

Then the following is displayed.

CLS RUN ↵

```
RUN
  1.414
-2
?
_ ..... Cursor
```

A PRINT statement can be made to display a plural number of expressions or character strings using commas (,).

```
10  A=0:B=3
20  PRINT 1.414,A*B-2
30  INPUT C
```

When you run this program, line change is performed each time data is displayed, the same as the preceding display.

```
RUN
  1.414
-2
?_
  ..... Cursor
```

Although the display screen of the PB-770 consists of 4 lines, if output is performed on the 4th line, each line is scrolled up.

```
10  PRINT 1, 2, 3, 4
20  END
```

```
4
                RUN
                 1
Ready P0         2   } scrolled up.
_                3
                 4
```

Although output is performed with line change using commas, if a plural number of expressions and character strings are punctuated with semicolons, they are displayed on the same line as follows.

```
10  A=0:B=0
20  PRINT 1.414;A*B-2
30  END
```

```
RUN
 1.414-2
Ready P0
-
```

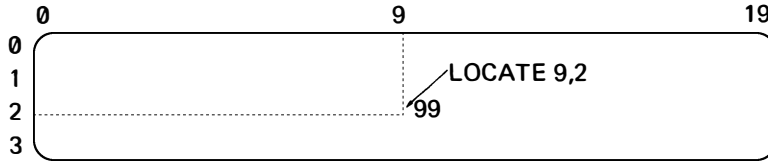
An easy-to-read message can be given using the following method.

```
10  PRINT "ANSWER=" ; A*B-2
20  END
```

```
RUN
ANSWER=-2
Ready P0
-
```

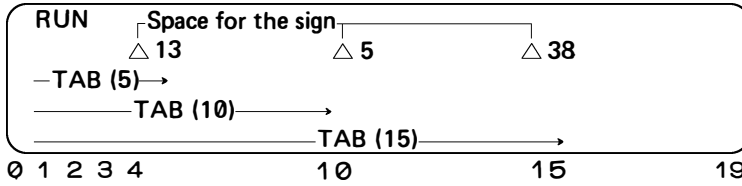
Character and numeral output can both be performed with left justification as shown in the previous output examples. However, since the output of a numerical value is performed by including one position for a sign, a space occurs where the + sign is omitted when the value is positive. A LOCATE command and functions such as TAB and USING, with which the display location and format can be specified, are used with a PRINT statement.

```
10   CLS
20  A=1
30  LOCATE 9, 2
40  PRINT A
50  A=A+1
60  IF A<100 THEN 30
70  END
```



Numerical values 1 through 99 are displayed in sequence at the location of the coordinates (10, 2) and (11, 2) of the character screen as displayed above. The space for the sign is displayed at the (9, 2) coordinate point.

```
10   CLS
20  A=13 : B=5 : C=38
30  PRINT TAB(5) ; A ; TAB(10) ; B ; TAB(15) ; C
40  END
```



Display is performed starting from the location specified by the TAB function when using a PRINT statement together with the TAB function.
Display can be performed with a uniform format in accordance with the USING function format as shown below.

```
10  A=3.1415 : B=31.415 : C=314.15
20  PRINT USING "###.##" ; A
30  PRINT USING "###.##" ; B
40  PRINT USING "###.##" ; C
50  FOR I=1 TO 1000 : NEXT I
60  END
```

```
  3.14
 31.42
314.15
```
..... Rounded to 2 decimal places.

■ **PRINT command expanded function (not available with LPRINT)**

(1) Display pattern definition

Display pattern can be defined by including a "$" directly before registered character variables such as AB. At this time, hexadecimal values (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) must be assigned to the registered character variable.

(2) Control code output

CLS and TAB can be executed using the PRINT command. Output is performed using the CHR$ function as in: PRINT CHR$(A). The control codes (numerical variable A) and output functions are as follows.

Code	Function
&H02	LINE TOP (Cursor to head of line)
&H06	LINE END (Cursor to end of line)
&H07	BEL (Same as BEEP 1)
&H09	TAB (Same as TAB(10))
&H0B	HOME (Cursor to the coordinate (0,0)
&H0C	CLS (Same as CLS)
&H0D	RETURN (Line change)
&H1C	→ (Cursor one space right)
&H1D	← (Cursor one space left)
&H1E	↑ (Cursor one line up)
&H1F	↓ (Cursor one line down)

SEE LOCATE, TAB, USING, CLS

PUT

Function	Stores variable data on a cassette tape.
Formats	PUT variable [, variable] PUT "file name" variable [, variable]

The PUT command stores variable data on a cassette tape. The format of the data file to be stored is ASCII. The GET command is used to read the data file.

(1) PUT A [, B, C . . . .]
(2) PUT "SALES" A [, B, C . . . .]

As shown in the above format (1), the file name may be omitted. In this case, the file name should not be specified in the corresponding GET command. By separating two or more variables with commas, it is possible to store plural variables as a data file by a single PUT command. Variables after PUT are stored on a 'first come, first served' basis. Numerical data is output in the same manner as it is output to the screen when the USING function is not employed.

Usage

The following program stores the contents of array variables A(0) to A(10) with a PUT command. It is assumed that the array variables contain data.

```
10 REM PUT
20 DIM A(10)
30 FOR K=1 TO 10
40 A(K)=K
50 PUT A(K)
60 NEXT K
70 END
```

The following program reads the data stored on a cassette tape into the PB-770 using a GET command.

```
10 REM GET
20 DIM B(10)
30 FOR K=1 TO 10
40 GET B(K)
50 PRINT "B(";K;")=";B(K)
60 NEXT K
70 END
```

In this example, a FOR-NEXT loop is used to move the contents of array A to array B. When storing plural data items, pay attention to the sequence in which they are stored.

SEE GET, SAVE

READ/DATA/RESTORE

Functions	READ: Reads data from a DATA statement into a variable. DATA: Stores data (constants, characters) in a program to be read by a READ statement. RESTORE: Specifies the line from which the DATA statements are read.
Formats	READ variable [, variable] DATA data [, data] [, "character data"] RESTORE RESTORE line number (1 ≦ line number < 10000)

A READ statement is used with a DATA statement.
When a READ statement is executed, data are read from a DATA statement into variables on a one to one basis.

Usage

The following is the simplest example of a READ/DATA statement program.

```
10 READ A
20 READ B$
30 PRINT A ; B$
40 DATA 7,"B"
50 END
```

When you run this program, 7 is assigned to variable A, and "B" is assigned to character variable B$. The variable and data types must match.
Since as many variables as desired can be written continuously in a READ statement, line 10 and line 20 can be written in one line as follows.

```
10 READ A, B$
```

It makes no difference whether character data is enclosed with double quotation marks or not as follows.

```
40 DATA 7, B
```

However, if data is not enclosed with double quotation marks, spaces are ignored as data. Therefore, if a space is required, it must be enclosed with double quotation marks.

```
DATA 7, "␣" "␣A",␣C
                 └── This space is ignored.
```

NOTE: Spaces after data are not ignored. Therefore if a space is provided after a numerical value, an ST error occurs.

```
DATA 1,2␣, 3,4␣
       └──────┘ These spaces are not ignored.
```

Double quotation marks and commas cannot be written in character data except for the above format.
A DATA statement that does not include any data is read as a null string.

DATA , , is regarded as DATA "","",""

Although variables in a READ statement must correspond to DATA statements on a one to one basis, any number of variables or data can be placed in each statement.

```
10 CLEAR
20 DIM C(10)
30 READ A, B
40 FOR I=1 TO 10
50 READ C(I)
60 NEXT I
70 DATA 1,2,3
80 DATA 4,5,6,7,8,9
90 DATA 10,11,12
```

When you run this program, data are assigned to each variable as follows.

```
A  B  C(1)  C(2)  C(3).....C(8)  C(9)  C(10)
↓  ↓  ↓     ↓     ↓        ↓     ↓     ↓
1  2  3     4     5.....10     11    12
```

If the number of data is less than the number of variables to which data are assigned by a READ statement, an error (DA error) occurs. However, if the number of data is more than the number of variables, an error does not occur but the extra data are ignored.
A DATA statement can be placed before a READ statement.
The DATA statement from which data are to be read by a READ statement can be specified using a RESTORE statement.

A RESTORE statement has two different formats, one in which the line number is written, and another in which the line number is not written. If the line number is not written, the following READ statement reads data form the first DATA statement when RESTORE is executed.

```
10 READ A, B
20 RESTORE
30 READ C, D
40 PRINT A;B;C;D
50 DATA 7, 2
60 END
```

When you run this program, the assignments performed are A = 7, B = 2, C = 7 and D = 2.
DATA statements can also be specified by including the line number.

```
10 RESTORE 50
20 READ A, B
30 PRINT A;B
40 DATA 3.7, 6.5
50 DATA 7.1, 9.3 ..... DATA statement specified by RESTORE
60 DATA 5, 10.2          statement in line 10.
70 END
```

When you execute this program, A = 7.1 and B = 9.3 are performed as the assignment.
Variables and numerical expressions can be used for a line number specification of a RESTORE statement, but the variable or numerical expression value used must correspond to a line number which includes a DATA statement.

Precautions should be taken when a READ statement is used in a program whose execution moves to a plural number of program areas.

```
P0  10 READ A,B          P1  10 READ X, Y
    20 GOTO  PROG 1          20 PRINT X;Y
    30 DATA 1,2,3,4          30 DATA 71,65
                             40 END
```

When you execute this program, displayed data are not 71 and 65, but are 3 and 4. In other words, although the execution of this program has been shifted to P1 by GOTO PROG 1, the DATA statement of P0 is still used.
This is useful when the DATA statement of a main program is used in a subroutine.
If it is necessary to read 71 and 65 into X and Y in this program, specify the line number at the beginning of Program P1 as follows.

```
5 RESTORE 30
```

CAUTION: Be sure not to write DATA statements in lines 2200 through 2299 or in lines with 22 in the last two digits of the line numbers (e.g. 22, 322, 922). Otherwise, such DATA statements will be ignored.

REM

Function	Provides comments for programs.
Format	REM comment statement

Unlike other commands, the REM command does not execute anything. Since anything can be freely written after REM, a program explanation can be written at important points in a program as shown below so that the contents of each part of a program can be understood by looking at the list.

```
 10  REM  *PROCESSING RESULTS*
 20  DIM  A(100)
        :
        :
100  REM  *PRINT OUT*
        :
        :
500  END
```

Since all of the characters and symbols written after REM are considered to be comments, it cannot be used before other commands to form a multistatement.

Example:
```
150 REM *TOTAL* : INPUT N
```
Not executed

STOP

Function	Suspends program execution.
Format	STOP

If the STOP command is found in a program during program execution, a STOP message is displayed and program execution is suspended. The execution of a program suspended by the STOP command can be re-started from the instruction following STOP by inputting the CONT command.

Usage

Let's check the function of the STOP command in the following program.

```
10 A=1:B=5
20 C=A+B
30 STOP
40 PRINT C
50 END
```

When you execute this program, the following is displayed.

STOP P0 – 30

This indicates that execution is suspended by the STOP command in line 30 of program area P0.
In this state, the contents of the variables can be checked as follows.

A [ENTER] → 1 Displays the value of A.
C [ENTER] → 6 Displays the value of C.

Also, an optional value can be assigned to a variable by entering

C = 0 [↵]

In actual practice, this command is used to stop execution of a program at a point where the operation is doubtful to confirm the contents of variables and aid in debugging.

[1] Execution can be resumed using CONT even if the following operations are performed while a STOP command is in effect.
(1) Manual calculation.
(2) Assignment to a variable (Assignment without using LET).
(3) Confirmation of the contents of a variable.
(4) Execution of the following commands.

ANGLE, BEEP, CLEAR, CLS, DIM, ERASE, PRINT, LPRINT, TRON, TROFF

[2] If the following operations are executed while a STOP command is in effect, execution cannot be resumed by CONT.
(1) Execution of manual commands (EDIT, SAVE, LOAD, LIST, etc.)
(2) Execution of PUT/GET.
(3) When an error occurs.

SEE CONT

TRON/TROFF

Function	Traces program execution/terminates tracing of program execution.
Formats	TRON TROFF

TRON and TROFF commands are used during program debug.

TRON ..... Specifies the trace mode.
TROFF ..... Releases the trace mode.

When the trace mode is specified, a program is executed while the present program area number and the line number are displayed as follows.

Display [0:110]
Line number being executed
Program area being executed

Since TRON, TROFF are program commands, they can be used by writing them in a program, but they are usually used by direct entry.

Usage

Input the following program and execute it.

```
P0
   10  BEEP 1
   20  GOTO PROG 1
P1
   10  BEEP 0
   20  GOTO PROG 0
```

When you run this program, two beeps start sounding alternately. Press the BRK key and then enter TRON ↵ to specify the trace mode. Now, run the program. The program area and line number being executed are continuously displayed as shown below.

```
(0 : 10)        (0 : 20)
(1 : 10)        (1 : 20)
(0 : 10)        (0 : 20)
```

You will notice that the interval between the BEEP sounds is rather long in trace mode. This is because execution is much slower in this mode.
In addition, during INPUT statement execution, it stops by displaying "?" after the [area number, line number]. And, the result is displayed during PRINT statement execution. This command is very convenient during debug because the program execution process can be traced.

SIN

Function	Gives the sine of X (Sin X).
Format	SIN numerical expression $-5400^\circ <$ Numerical expression $< 5400^\circ$

The SIN function is used to compute Sin X.
Any one of 3 angle units (DEG, RAD, GRA) can be selected.
When the power is turned on, the angle unit is set to DEG (degree).

Example

SIN X computation is performed using DEG (degree).

```
 5 ANGLE 0
10 PRINT SIN30
20 PRINT SIN45
30 PRINT SIN90
50 END
```

When you run this program ([R][U][N] [↵]), the results for SIN 30, SIN 45 and SIN90 flow across the display and disappear.
Run this program again after adding the following line.

```
25 STOP
```

Now the results for SIN30 and SIN45 are displayed and stopped.

Execution Example

```
 0.5
 0.7071067812
STOP P0-25
—
```

To continue, enter either CONT [↵] or [SHIFT] [CONT/W] [↵] , and the next result (SIN90) will be displayed. (See page 131.)

Execution Example

```
 1
Ready P0
—
```

The following program allows you to select one of the angle units (DEG, RAD, GRA) and compute SIN X.

```
10 REM SIN X EXAMPLE
20 INPUT "ANGLE=";K
30 ANGLE K
40 INPUT "SIN X:X=";X
50 PRINT "SIN";X;"=";SINX
60 STOP
70 END
```

When you run this program, the angle unit is requested first as follows.

ANGLE = ?

Next you should specify the angle unit as follows.

ANGLE 0 → DEGREE
ANGLE 1 → RADIAN
ANGLE 2 → GRAD

RADIAN is selected as an example.
Enter 1.

SIN X X = ? is now requested.

Next, when you input the Radian angle, such as PI/4, the display result is as follows.

Execution Example

```
SIN 0.7853981634=0.
7071067812
STOP P0-60
—
```

The angle unit is modified using the ANGLE command as described, and the input range for each angle unit is as follows.

DEG $-5400^{\circ} <$ Numerical expression $< 5400^{\circ}$
RAD $-30\pi <$ Numerical expression $< 30\pi$
GRA $-6000 <$ Numerical expression < 6000

When the value of a numerical expression is outside the ranges shown above, an error (BS error) occurs.
A variable and a numerical expression, as well as a real number (such as 30), can be used for the argument.
When only one real number or variable is used, it makes no difference whether or not the argument is placed inside parentheses. However, when a numerical expression is used, the result will differ depending on whether or not it is placed inside parentheses as follows.

SIN X + Y ..... Add Y to the result of the SIN X computation.
SIN (X + Y) .... Compute the SIN of the result of X + Y.

SEE ANGLE, COS, TAN

Memorandum

There are three different ways to express the angle of a trigonometric function: "Degree (DEG)", "Radian (RAD)" and "Grad (GRA)."

DEG ..... 1° is 1/360 of the circumference of a circle.
RAD ..... 1 rad. is $1/2\pi$ of the circumference of a circle.
GRA ..... 1 grad is 1/400 of the circumference of a circle.

Degree and radian are mainly used, and they have the following relationship.

$1^{\circ} = \pi / 180$ rad $\fallingdotseq 3.141592654 / 180$ rad

COS

Function	Gives the cosine of X (Cos X).
Format	COS numerical expression $-5400^\circ <$ Numerical expression $< 5400^\circ$

The COS function is used to compute COS X.
The angle units, X argument input range, and precision for COS X are exactly the same as for SIN X.

Example

Provide input to a program in which COS X is used.

```
10 REM COS X EXAMPLE
20 INPUT "ANGLE=";K
30 ANGLE K
40 INPUT "COS X:X=";X
50 PRINT "COS";X;"=";COSX
60 STOP
70 END
```

When you run this program, angle unit input is requested by

ANGLE = ?

Next, if the grad angle unit is to be used, input "2". After this, since the angle is requested as follows, provide an input of 1355.

COS X X = ?

The result will be −0.7604059656.

SEE SIN, ANGLE, TAN

TAN

Function	Gives the tangent of X (Tan X).
Format	TAN numerical expression $-5400° <$ Numerical expression $< 5400°$ Except |numerical expression| = $(2n - 1)$ x 1 right angle (n = integer)

The TAN function is used to compute TAN X.
The angle units are the same as for SIN X and COS X.

Example

A program to obtain TAN X.

```
10 REM TAN X EXAMPLE
20 INPUT "ANGLE=";K
30 ANGLE K
40 INPUT "TAN X:X=";X
50 PRINT "TAN";X;"=";TANX
60 STOP
70 END
```

When you obtain TAN 45 using the DEG angle unit (ANGLE 0), the result is displayed as

TAN 45 = 1.

Next, if you try to obtain TAN 90, an "MA error" is displayed.
When the TAN function is used, the value of TAN X suddenly increases as it approaches 90°.
At TAN 90, the value becomes infinite and computation cannot be performed.
As a result, an "MA error" was displayed in the above example. When the value of X is $\pm 90 * (2n-1)$ (n is an integer) in TAN X, precautions should be taken since an error occurs as mentioned above.

SEE SIN, COS, ANGLE

ASN, ACS, ATN

Functions	ASN gives the arcsine (Sin^{-1} X). ACS gives the arccosine (Cos^{-1} X). ATN gives arctangent (Tan^{-1} X).
Formats	ASN X, ACS X, ATN X. $\quad |X| \leqq 1$ (ASNX, ACS X) $|X| \leqq 10^{100}$ (ATN X)

The ASN, ACS and ATN functions are used to compute the inverse trigonometric functions: SIN^{-1} X, COS^{-1} X and TAN^{-1} X.
The trigonometric functions (SIN, COS, TAN) are used to obtain the trigonometric function values of given angles. On the other hand, the inverse trigonometric functions obtain angles when trigonometric function values are given.

Example

A program example which uses ASN X is shown below.

```
10 REM ASN X EXAMPLE
20 INPUT "ANGLE=";K
30 ANGLE K
40 INPUT "ASN X:X=";X
50 PRINT "ASN";X;"=";ASNX
60 STOP
70 END
```

When you run this program, the following two input requests are displayed.

ANGLE = ? 0 ↵ ..... Specifies "degree."
ASN X X = ? 1 ↵ ..... Inputs the trigonometric function value.

When you input these values, the following is displayed.

ASN 1 = 90

In other words, the angle X of SIN X = 1 was obtained.

Try this program again using ACS and ATN to replace ASN.
These inverse trigonometric functions are specified by ANGLE the same as for SIN, COS, and TAN.
The degree (DEG) angle range in the computation result is as follows.

$$-90^\circ \leqq \text{ASN} \leqq 90^\circ$$
$$0^\circ \leqq \text{ACS} \leqq 180^\circ$$
$$-90^\circ \leqq \text{ATN} \leqq 90^\circ$$

Since SIN X and COS X do not theoretically exceed 1, the value of argument X of ASN X and ACS X must not exceed 1.

SEE SIN, COS, TAN, ANGLE

PB-770 Only

HYPSIN/HYPCOS/HYPTAN

Function	Give the hyperbolic functions.
Formats	HYPSIN numerical expression HYPCOS numerical expression $-10^{100} <$ Numerical expression $\leqq 230.2585092$ HYPTAN numerical expression |Numerical expression| $< 10^{100}$

This series of functions expresses the hyperbolic functions. Each respective numerical expression is as follows.

HYP SIN : $\sinh x = (e^{x} - e^{-x}) / 2$

HYP COS : $\cosh x = (e^{x} + e^{-x}) / 2$

HYP TAN : $\tanh x = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$

PB-770 Only

HYPASN/HYPACS/HYPATN

Function	Give the hyperbolic functions.
Formats	HYPASN numerical expression HYPACS numerical expression |Numerical expression| $< 5 \times 10^{99}$ HYPATN numerical expression |Numerical expression| < 1

This series of functions expresses the inverse hyperbolic functions. Each respective numerical expression is as follows.

HYP ASN : $\sinh^{-1} x = \log (x + \sqrt{x^2 + 1})$

HYP ACS : $\cosh^{-1} x = \log (x + \sqrt{x^2 - 1})$

HYP ATN : $\tanh^{-1} x = \frac{1}{2} \log \frac{1 + x}{1 - x}$

SQR

Function	Gives the square root of the argument.
Format	SQR numerical expression Numerical expression ≧ 0

The SQR function is used to obtain a square root as follows.

SQR X = X^0.5 = $\sqrt{X}$

In this case, the value of X must be larger than 0.

Example

The following program is used to input the area of a circle in order to obtain the radius.

```
10 REM SQR X EXAMPLE
20 INPUT "CIRCLE AREA=";S
30 R=SQR(S/PI)
40 PRINT "CIRCLE RADIUS=";R
50 LOCATE 0,2
60 END
```

When you execute this program, the following is requested.

CIRCLE AREA = ?

Enter 100 as an example. Then 5.641895835 is displayed for the value of the radius.

If you run this program again and enter a minus value, an MA error will immediately be displayed because SQR (S/Pi) becomes an imaginary number when the argument (X) of SQR X is a minus value. To avoid this error, it is recommended that the following line be added to check for a positive or negative argument.

```
35 IF S<0 THEN 20
```

LOG, LGT

Functions	LOG X LGT X	Gives the value of natural logarithm $\log_e X$. Gives the value of common logarithm $\log_{10} X$
Formats	LOG numerical expression LGT numerical expression	Numerical expression > 0

LOG X computes the value of natural logarithm $\log_e X$ (lnX). In this case, "e" is the base of a natural logarithm.
The value of e is as follows.

$e = 2.718281828 \ldots\ldots$

LGT X computes the value of common logarithm, $\log_{10} X$. The base of a common logarithm is 10.

Example

The following program computes LOG X for successive given values of X.

```
10 REM LOG X EXAMPLE
20 INPUT "X=";X
30 PRINT "LOG";X;"=";LOGX
40 GOTO 20
```

When you execute this program,

```
X = ?
```

is displayed which requests the value of X for LOG X.

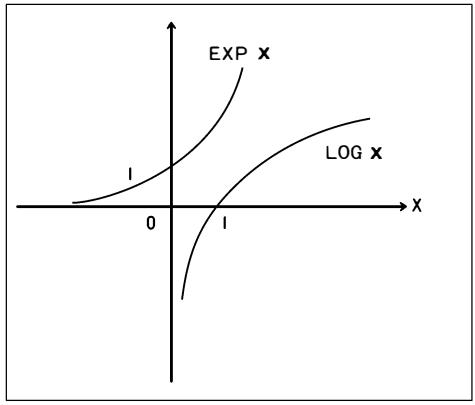
If you enter "1", the value of LOG X is displayed as LOG (1) = 0, and the next value of X is requested.

Execution Example

```
CLS
RUN ↵
  1 ↵
```

```
RUN
X=?
LOG  1 =0
X=?_
```

The logarithmic function LOG X has an inverse relationship with the exponential function EXP X as shown in the following graph.



X > 0 is required in a logarithmic function as shown in the above graph.
If a negative value is entered, an MA error is displayed.
While LOG X is the logarithm of X (which has a base of e), the logarithm $\log_y$ X (which has a base other than e), can be computed by the following formula.

LOG X/LOG Y

Therefore the common logarithm of X, LGT X can also be obtained with the following formula.

LOG X/LOG 10

SAMPLE PROGRAM

* This program allows many different logarithmic values to be obtained when base values are entered.

```
10 REM ** LOG X/LOG Y **
20 INPUT "X=";X
30 INPUT "Y=";Y
40 PRINT "LOG";X;"/LOG";Y;"=";LOGX/LO
   GY
50 LOCATE 0,3
60 STOP
70 END
```

This program computes the value of $\log_Y X$ with the formula, LOG X/ LOG Y.
An execution example is provided below.

Execution Example

RUN ↵
10 ↵
2 ↵

```
LOG 10/LOG 2= 3.3219
28095
STOP P0-60
_
```

SEE EXP

EXP

Function	Gives the exponential function e^x.
Format	EXP numerical expression $-10^{100} <$ Numerical expression $\leqq 230.2585092$

The EXP function is used to compute the value (e^x) of an exponential function. The value of "e", which is the base of an exponential function, is as follows.

$e = 2.718281828\ldots\ldots$

The expression "exponential increase" is often heard in conversation. The nature of this function is that the value of EXP X suddenly increases as the value of argument X increases.

Example

Enter the following program to observe the change in the value of EXP X.

```
10 REM ** EXP X **
20 INPUT "A=";A
30 FOR X=1 TO A
40 PRINT "EXP";X;"=";EXPX
50 FOR I=1 TO 300:NEXT I
60 NEXT X
70 END
```

When you execute this program, a request is made for the maximum argument value X of EXP X.

A = ?

Enter "10" as an example and press the ↵ key. The results shown on the following page are displayed in series.

Execution Example

```
EXP 1= 2.718281828
EXP 2= 7.389056099
EXP 3= 20.08553692
EXP 4= 54.59815003
EXP 5= 148.4131591
EXP 6= 403.4287935
EXP 7= 1096.633158
EXP 8= 2980.957987
EXP 9= 8103.083928
EXP 10= 22026.46579
```

You will see that the value of EXP X increases suddenly.
Run this program again entering "231" for A.
The values of EXP X are displayed in series, and, when the value of the argument is 231, an MA error occurs.
The input range of the argument X of EXP X is actually

$$X \leqq 230.2585.$$

The value of EXP X with X = 230.2585 is as follows.

Execution Example

```
EXP 230.2585= 9.999907006E99
```

ABS

Function	Gives the absolute value of the argument.
Format	ABS numerical expression

ABS X gives the absolute value of X which is mathematically expressed as follows.

ABS X = |X|

In regard to the X of ABS X,
When $X \geqq 0$ (value X is positive) . . . . . ABS X = X,
and when $X < 0$ (value X is negative) . . . . . ABS X = –X.
In other words, ABS X computes in a way that allows the result to become a positive number (absolute number).

Example

Now let's look at a program which uses the ABS function.

```
10 REM ABS X EXAMPLE
20 READ A,B,C,D
30 X=A:GOSUB 40:X=B:GOSUB 40:X=C:GOSU
   B 40:X=D:GOSUB 40:END
40 PRINT "ABS";X;"=";ABSX
50 FOR I=1 TO 200:NEXT I
60 RETURN
70 DATA5,-5,0,-7.5
```

This program reads 5, –5, 0 and –7.5 into variables A, B, C and D respectively using a READ statement and computes ABS A to ABS D.

The result is displayed as follows.

Execution Example

```
ABS 5=5
ABS -5=5
ABS 0=0
ABS -7.5=7.5
```

An operation the same as the ABS function can be performed using the SGN function.

ABS X is the same as X * SGN X

SAMPLE PROGRAM

* A program in which no error occurs when a negative value is entered.

```
10 INPUT X
20 S=SQR(ABSX)
30 L=LOG(ABSX)
40 PRINT "SQRX=";S
50 FOR I=1 TO 200:NEXT I
60 PRINT "LOGX=";L
70 END
```

When argument X is a negative value for functions such as SQR X and LOG X, an MA error occurs.
Therefore, in this program, calculation is performed using the absolute value of X.

SEE SGN

INT

Function	Gives the largest integer which does not exceed the argument value.
Format	INT numerical expression

INT X gives the largest integer that does not exceed the value of X. For example, when the values of X are 3.9, 0.5, –0.5 and –3.9, INT X is as follows for each of these values.

INT 3.9 = 3
INT 0.5 = 0
INT –0.5 = –1
INT –3.9 = –4

When the value of the argument is positive, the value after the decimal point is discarded. However, care should be exercised when the argument is negative. For example, if the argument is –0.5, the integer is not 0 but –1 which is the largest integer that does not exceed –0.5.

Example

Let's try the following program.

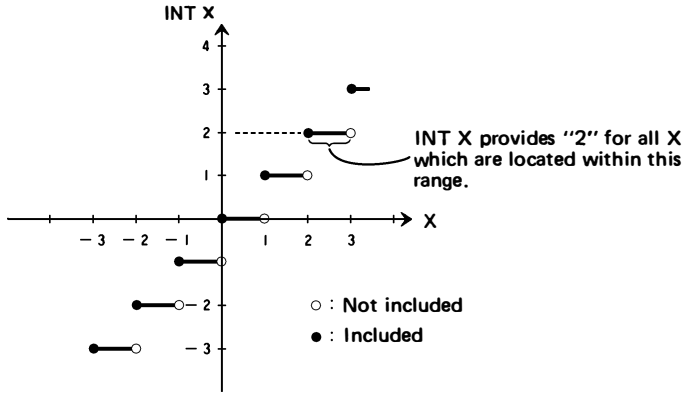
```
10 REM INTX EXAMPLE
20 READ A,B,C
30 X=A:GOSUB 40:X=B:GOSUB 40:X=C:GOSU
   B 40:END
40 PRINT "INT";X;"=";INTX
50 FOR I=1 TO 200:NEXT I
60 RETURN
70 DATA5.3,0.5,-3.9
```

Execution Example

```
INT 5.3= 5
INT 0.5= 0
INT-3.9=-4
```

When you run this program, the results shown on the previous page are displayed.
An INT function graph is drawn as follows by placing values of X horizontally and values of INT X vertically.

INT X Graph



The difference for positive or negative values can be found using this graph.
The INT function is often used by combining it with other functions such as the RND function. In addition, the FRAC function, ROUND function, etc. are similar to the INT function.

SAMPLE PROGRAM

* A program which displays 5 integers between 0 and 9 at random.

```
10 FOR I=1 TO 5
20 PRINT INT(10*RND);
30 NEXT I
40 END
```

This program was prepared by combining the INT function with the RND function as an example.

SEE FRAC, ROUND, RND

FRAC

Function	Gives the value of the fractional part of the argument.
Format	FRAC numerical expression

FRAC X gives the value of the fractional part of X.
Simple examples are provided as follows.

FRAC 1.123 = 0.123
FRAC −1.123 = −0.123

Simply stated, this function discards the integer part as shown above.

Example

Try the following program by entering many different values.

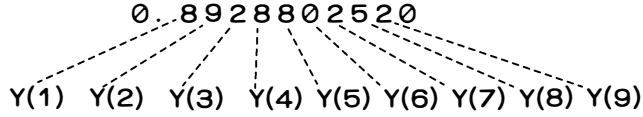
```
10 REM FRAC X EXAMPLE
20 INPUT "NUMBER";X
30 PRINT FRACX
40 GOTO 10
```

SAMPLE PROGRAM

* This program generates a 9-decimal place random number and then fetches the digits one by one to convert them into single-digit integers.

```
10 DIM Y(9)
20 X=RND
30 FOR I=1 TO 9
40 Y(I)=INT(10*X)
50 PRINT "X=";X
60 PRINT "Y(";I;")=";Y(I)
70 FOR J=1 TO 150:NEXT J
80 X=FRAC(10*X)
90 NEXT I
```

This program assigns a value generated by the RND function to variable X.
The value is multiplied by 10 and then the INT function is used to obtain a single-digit integer. The integer is assigned to array variable Y(1) and then the remaining decimal portion is fetched using the FRAC function and assigned to X. Then the process begins again. This is repeated 9 times and results in 9 single-digit integers assigned to array variables Y(1) to Y(9).



SEE INT, RND, ROUND

SGN

Function	Gives a value according to the sign of argument.
Format	SGN numerical expression

The SGN X function judges whether the value of argument X is positive or negative. SGN X provides three different results as follows.

X > 0 (If positive) ..... SGN X = 1
X = 0 (If 0) ..... SGN X = 0
X < 0 (If negative) ..... SGN X = –1

Example

Try the following program.

```
10 REM SGN X EXAMPLE
20 PRINT "JUDGEMENT OF + OR -"
30 INPUT "NUMBER";X
40 A=SGNX
50 IF A=1 THEN PRINT "+":GOTO 30
60 IF A=0 THEN PRINT "0":GOTO 30
70 IF A=-1 THEN PRINT "-":GOTO 30
```

In lines 50 to 70, if the input value is positive, a "+" is displayed, if it is negative, a "–" is displayed, and if it is 0, a "0" is displayed based on the value obtained by SGN X.

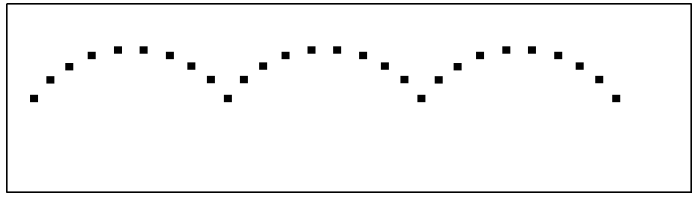
SAMPLE PROGRAM

* A program which provides a sine curve.

```
10  CLS :FOR X=0 TO 540 STEP 20
20 S=SGN(SINX)
30 Y=S*INT(S*10*SINX)
40 IF S<0 THEN 70
50 DRAW(X/4,16-Y)
60 GOTO 80
70 DRAW(X/4,16+Y)
80 NEXT X
90 END
```

This program produces SIN X for X=0 to 540° at 20° increments. These values are multiplied by 10 to produce single-digit values which are assigned to Y.
At this time the SGN function is taken for SIN X, and, even if SIN X produces a negative number, the INT function obtains the integer portion only.
In this way, X and Y values are plotted by the DRAW command centered around coordinates (0,16) – (159,16). If a value is positive, (X/4, 16–Y) is plotted, while, if a value is negative, (X/4, 16+Y) is plotted. (See DRAW.)
When this program is executed, a rough sine curve is plotted as shown below.

Execution Result



ROUND

Function	Gives the value of numerical expression 1 which is rounded at a position specified by numerical expression 2.
Format	ROUND (numerical expression 1, numerical expression 2) Numerical expression 2: Digit position

ROUND (X, Y) gives the value of X which is rounded at the 10^Y position as follows.

ROUND (12345, 2) = 12000

The numerical value 12345 is rounded at the 10^2 position which is the 100 position.

Example

You can confirm the operation of the ROUND function with the following program.

```
10 FOR Y=3 TO -5 STEP -1
20 X=12345.67891
30 Z=ROUND(X,Y)
40 PRINT USING"#####.#####";X,Z:PRINT
50 FOR J=0 TO 150:NEXT J
60 NEXT Y
70 END
```

This program performs a calculation in which the value Y of ROUND (X, Y) is continually decreased by 1 from 3 to −5.
The value specified for the 2nd argument Y is

$|Y| < 100$

If it exceeds this range, a BS error occurs.
When a value with a fraction is specified for the value of Y, the fractional part is discarded.

Execution Example

```
12345.67891
10000.00000
12345.67891
12000.00000
12345.67891
12300.00000
12345.67891
12350.00000
12345.67891
```

SAMPLE PROGRAM

* This program displays binary 8-bit random numbers and their decimal values.

```
10 Y=0
20 FOR I=7 TO 0 STEP -1
30 X=ROUND(RND,-1)
40 Y=Y+(2^I)*X
50 PRINT X;
60 NEXT I
70 PRINT "=";Y
80 LOCATE 0,1
90 FOR J=0 TO 500:NEXT J
100 END
```

Since random numbers generated by the RND function (page 239) are rounded to one decimal place in line 30, the value of X is 0 or 1.
The value of X is generated 8 times to provide an 8-bit random number that consists of 0s and 1s.
At the same time, the binary 8-bit value is converted to a decimal number in line 40 which is continuously displayed. An execution example is provided below.

RUN ↵

```
RUN
00101001=41
```

PI

Function	Gives the ratio of the circumference of a circle to its diameter (π).
Format	PI

PI gives approximations of the ratio of the circumference of a circle to its diameter (π).
The value provided for π is as follows

π = 3.141592654 . . . . . .

Example

The following program computes the area of a circle.

```
10 REM PI EXAMPLE
20 PRINT "CIRCLE AREA"
30 INPUT "RADIUS";R
40 S=PI*R^2
50 PRINT "S=";S
60 END
```

If you enter 5 as the radius value, the area of the circle is displayed as follows.

Execution example

```
S= 78.53981634
```

RND

Function	Gives a random number value.
Format	RND 0 < Random number < 1

The RND function gives a 10 digit pseudo random number value that is larger than 0 and smaller than 1.
Random numbers were first required for the simulation of statistical phenomena or probability models, and are now used for simulations such as economic forecasts or TV games. In particular, the fun provided by TV games is largely due to this random number function.

Usage

The following program generates 10 random numbers.

```
10 FOR N=1 TO 10
20 PRINT RND
30 FOR X=1 TO 500:NEXT X
40 NEXT N
50 END
```

These results are only examples. Of course, each time you run this program, different random numbers will be generated.

```
0.6791506196
0.9598232115
0.2057199883
0.5039057551
0.306977109
0.1065778556
0.4177075471
0.5017414683
0.7551551958
0.4560918328
```

Since a large number of digits are generated, they are not easy to handle as they are. Therefore, when used for games, etc., random number values within an appropriate range are obtained by combining this function with the INT function and ROUND function as follows.

(1) Produce integers up to a desired digit.

INT (RND ∗ 10^L) . . . . . L indicates the number of digits.

(2) Produce integers from N to the upper limit M.

ROUND (RND ∗ (M−N), −1) + N . . . . N and M are integers. (N<M)

DEG

Function	Converts sexagesimal to decimal.
Format	DEG (degrees [, minutes [, seconds]])

The DEG function converts a sexagesimal value to a decimal value. The degrees, minutes and seconds of a sexagesimal value have the following relationship with a decimal value.

$$\text{DEG (deg., min., sec.)} = \text{deg.} + \text{min.}/60 + \text{sec.}/3600$$

Input must be within the following range.

$$|\text{DEG (deg., min., sec.)}| < 10^{100}$$

Input sexagesimal values into the following program to confirm operation of the DEG function.

```
10 REM DEG EXAMPLE
20 INPUT "DEG. = ",A
30 INPUT "MIN. = ",B
40 INPUT "SEC. = ",C
50 D=DEG(A,B,C)
60 PRINT "DEG(";A;",";B;",";C;")"
70 PRINT "=";D
80 GOTO 20
```

SEE DMS$

PB-770 Only

PEEK

Function	Gives the memory contents at a specified address.
Format	PEEK (address / numerical expression)

The PEEK function produces the value stored in specified fraction address. The range of the addresses is given below. Any values included in the address are discarded.

$$-32769 < \text{address} < 65536$$

See CLEAR for information on addresses.
The following program displays the contents of an input address.

```
10 REM PEEK EXAMPLE
20 INPUT "Address = ",A
30 B=PEEK(A)
40 PRINT B
50 GOTO 20
```

SEE CLEAR, POKE

ASC

Function	Gives the decimal code for the first character of a character string.
Formats	ASC "Character string" ASC (Character variable)

All characters, numerals, and symbols displayed by the PB-770 have a number which is called ASCII code (character code).
Examples are as follows.

"A" 65
"B" 66
"6" 54

(See page 327, CHARACTER CODE TABLE.)

These character numbers (codes) can be directly determined by the PB-770 using the ASC function, and can also be determined using the CHARACTER CODE TABLE.

Example

Enter PRINT ASC ("E") ↵

69 is displayed as the
ASCII code value of "E".

When an entry is made to determine two character codes or more such as

PRINT ASC ("EF") ↵

only the character code for "E", the initial character, is displayed.
Therefore, to determine the codes for a long character string (such as "ABCDEF ....") serially from the beginning, use a program that includes the MID$ function (see MID$).

SAMPLE PROGRAM

* A program that displays character codes of input characters.

```
10 REM ASCII CODE
20  CLS
30 INPUT "WHICH CHARACTER";A$
40 PRINT ASC(A$)
50 GOTO 30
```

When character input is performed in line 30, the code is displayed in line 40. Since execution returns from line 50 to line 30, character input can be continuously performed to determine codes.

The execution result is as follows.

Execution Example

```
WHICH CHARACTER? A
 65
WHICH CHARACTER? T
 84
WHICH CHARACTER? R
 82
WHICH CHARACTER? 7
 55
WHICH CHARACTER? 1
 49
```

Since this program indefinitely requests character codes, press the BRK key to stop execution.

SEE CHR$

CHR$

Function	Gives the character represented by a specified ASCII code.
Format	CHR$ (Code) $0 \leq$ Code < 256

The CHR$ function is used to produce the character, number, or symbol that corresponds to a specified ASCII code.

Example

Enter PRINT CHR$ (66) ↵

The character "B" is displayed for ASCII code 66.
To determine two characters at one time, enter

PRINT CHR$ (71); CHR$ (80) ↵

The characters "G" and "P" which correspond to ASCII codes 71 and 80 are displayed.
Characters that can be entered on the PB-770 by direct key input are numerals, capital alphabetic characters, small alphabetic characters, and some symbols. Other characters (such as graphic characters) are displayed using CHR$ (see page 327, CHARACTER CODE TABLE). Numbers (codes) that can be specified by CHR$ are within a range of $0 \leq$ Code $<$ 256, and the fractional part is ignored.

SAMPLE PROGRAM

```
 5 V=0
10 FOR I=33 TO 254
20 PRINT I
30 V=V+1
40 FOR J=1 TO 14
50 PRINT CHR$(I);
60 NEXT J
70 PRINT
80 IF V<3 THEN 110
90 K$=INKEY$:IF K$="" THEN 90
100 V=0
110 NEXT I
120 END
```

* This program displays the characters for character codes 33 to 254.

When this program is executed, the characters corresponding to ASCII codes 33 to 35 are displayed by lines 14 characters long, and then the characters corresponding to the next three sequential ASCII codes are displayed when any key is pressed. Execution continues until the 254th character. A display example is shown below.

```
 33
!!!!!!!!!!!!!!
 34
""""""""""""""
 35
##############
 36
$$$$$$$$$$$$$$
 37
%%%%%%%%%%%%%%
 38
&&&&&&&&&&&&&&
```

ASC

VAL

Function	Converts a character string into a numerical value.
Formats	VAL "Character string" VAL (Character variable)

VAL is a function that converts a character into a numerical value.
The difference between a character and a numerical value must be explained in order to understand VAL.
Compare the following two program examples.

Program (1)

```
10 READ A,B
20 C=A+B
30 PRINT C
40 END
50 DATA3,5
```

Program (2)

```
10 READ A$,B$
20 C$=A$+B$
30 PRINT C$
40 END
50 DATA3,5
```

In program (1), the numerical data are read into the numerical variables A and B, and the arithmetic result is displayed by assigning it to C.
The result of program execution is

␣8

In the above, ␣ indicates the space for the " + " sign which is always omitted.
On the other hand, in Program (2), 3 and 5 are read into the character variables A$ and B$, respectively, as character data.
With character variable operations, only addition can be performed. In this case, the result is assigned to C$.
When this program is executed,

35

is displayed. The result is just the display of a character string. There is no "—" sign or blank for the " + " sign.

This blank is very significant, and the difference will be clarified by comparing the Program (3) and (4) execution examples which follow.

Program (3)

```
10 FOR I=1 TO 10
20 READ A
30 PRINT A;
40 NEXT I
50 END
60 DATA3,8,-6,7,21
70 DATA223,18,8,1,0
```

Execution Example

```
 3 8-6 7 21 223 18 8
 1 0
```

Program (4)

```
10 FOR I=1 TO 10
20 READ A$
30 PRINT A$;
40 NEXT I
50 END
60 DATA3,8,-6,7,21
70 DATA223,18,8,1,0
```

Execution Example

```
38-672122318810
```

Usage

The VAL function is used to perform calculation as in Program (1) above using the numerals read in character variables A$ and B$ as in Program (2).

```
10 READ A$,B$
20 C=VAL(A$)+VAL(B$)
30 PRINT C
40 END
50 DATA3,5
```

When the program is executed, "␣8" is displayed as for Program (1).
The following precautions should be taken when the VAL function is used.

(1) When a character other than a numerical value, decimal point, sign (+, −) or exponent sign "E" appear in the character string, everything following that character is ignored. (Second and subsequent appearances of the exponent sign "E" are ignored.)
(2) The first space in a character string is disregarded.
(3) When the initial character of a string is not a numerical value, decimal point, or a sign, or when a character string is only a sign or a decimal point, 0 (zero) is provided.
(4) When more than three numerals exist after an exponent sign "E", an SN error occurs.

SAMPLE PROGRAM

* An input subroutine where no error occurs when any key is pressed as a response to the input request of Menu Nos. 1 to 5.

```
100 Z$=INKEY$:IF Z$="" THEN 100
110 IF Z$<"1" THEN 100
120 IF Z$>"5" THEN 100
130 GOSUB VAL(Z$)*1000
                    }
```

In this program, when a key from 1 to 5 is pressed, the program jumps to the corresponding subroutine at lines 1000 to 5000, and when a key other than one of these is pressed, re-input is requested. This subroutine is convenient to use as an input routine for job selection.

SEE STR$

STR$

Function	Converts a numerical value into a character string.
Format	STR$ (Numerical expression)

The STR$ function converts a numerical value into a character string.

Usage

What are the execution results of Programs (1) and (2)?

Program (1)

```
10 A=25:B=30
20 C$=STR$(A+B)
30 PRINT C$
40 END
```

Program (2)

```
10 A=25:B=30
20 C$=STR$(A)+STR$(B)
30 PRINT C$
40 END
```

Although these two programs seem to be identical, "55" is displayed in Program (1) and "25 30" is displayed in Program (2).
In Program (1), the result of the numerical expression A+B is converted into a character by the STR$ function. In Program (2), the contents of numerical variables A and B are converted into their respective characters and are then added. This difference appears in the execution result.

SAMPLE PROGRAM

* Addition practice program.

```
10 REM ADDITION
20 FOR I=1 TO 5
30 X=INT(RND*100)
40 Y=INT(RND*100)
50 Z=X+Y
60 PRINT STR$(X);"+";RIGHT$(STR$(Y),L
   EN(STR$(Y))-1);
70 INPUT "=",A
```

```
 80 IF A=Z THEN PRINT "OK" ELSE 60
 90 NEXT I
100 END
```

(See RIGHT$, LEN.)

This program produces five addition problems. The operator supplies the answer for the addition of two integers up to two digits long. The pair of numbers used cannot be predicted because the RND function is used.
The STR$ function is used in line 60 where the problem is displayed.
Although

```
60 PRINT X; " + "; Y;
```

seems to be reasonable without using STR$, a blank for the + sign occurs before the numerical value as follows.

```
␣15+␣30 = ?
```

This is a good example of the utility of the STR$ function.

Reference The VAL function is the reverse of the STR$ function.

LEFT$

Function	Fetches a specified number of characters from the left of a character string.
Formats	LEFT$ ("Character string", numerical expression) LEFT$ (Character variable, numerical expression)

LEFT$ is a function that fetches a specified number of characters from the left of a character string.

Example

```
10 AB$="LEFT RIGHT" . . .
20 B$=LEFT$(AB$,4)
30 PRINT B$
```

It should be noted that since the assigned character string is more than 6 characters, a registered variable is used (assigned up to 16 characters.).

When this program is executed, LEFT is displayed. In other words, four characters from the left of the character string in AB$ are fetched.
When 0 is specified as the number of characters to be fetched, a null is provided, and, when the number of characters specified exceeds the total length of the string, all characters in the string are fetched. However, if the number of characters specified exceeds 255, a BS error occurs.
The number of characters to be fetched can be specified by a variable or numerical expression.

SAMPLE PROGRAM

* A program which serially increases the character string display.

```
10 AB$="READ LEFT"
20 N=LEN(AB$)
30 FOR I=1 TO N
40 BC$=LEFT$(AB$,I)
50 PRINT BC$
60 NEXT I
70 END
```

Execution Example

```
R
RE
REA
READ
READ
READ L
READ LE
READ LEF
READ LEFT
```

SEE RIGHT$

RIGHT$

Function	Fetches a specified number of characters from the right of a character string.
Formats	RIGHT$ ("Character string", numerical expression) RIGHT$ (Character variable, numerical expression)

RIGHT$ is a function that fetches a specified number of characters from the right of a character string.

Example

```
10 AB$="LEFT RIGHT"
20 B$=RIGHT$(AB$,5)
30 PRINT B$
```

It should be noted that since the character string is more than 6 characters, a registered variable is used.

When the program is executed, RIGHT is displayed which indicates that five characters from the right of the character string in AB$ are fetched. When 0 is specified for the number of characters to be fetched, a null is provided, and, when the number of characters is specified that exceeds the total length of the string, all characters in the string are fetched. However, if the number of characters specified exceeds 255, a BS error occurs.
The number of characters to be fetched can be specified by a variable or numerical expression.

SAMPLE PROGRAM

* A program which inserts a character string in a character string.

```
10 AB$="AM PM"
20 BC$="NOON"
30 CD$=LEFT$(AB$,2)
40 CD$=CD$+" "+BC$
50 CD$=CD$+RIGHT$(AB$,3)
60 PRINT CD$
70 END
```

In this program, the character string in BC$ is inserted in the character string in AB$ using LEFT$ and RIGHT$.

Execution Result

```
AM NOON PM
```

SEE LEFT$

MID$

Function	Fetches a specified number of characters to the right of a specified position within a character string.
Formats	MID$ ("Character string", numerical expression 1, numerical expression 2) MID$ (Character variable, numerical expression 1, numerical expression 2)

MID$ is a function that fetches a specified number of characters to the right of a specified position within a character string. This function is a kind of combination of LEFT$ and RIGHT$.

```
MID$ ( A$ , 3 , 2 )
```

The above expression indicates that two characters should be fetched from A$. The two characters are the 3rd and 4th from the beginning of the character string in A$.

Example

```
10  CLEAR
20  DIM A$(0)*20                    .....Up to 20 characters
30  A$(0)="LEFT␣CENTER␣RIGHT"       can be assigned to
40  B$=MID$(A$(0),6,6)              a string array.
50  PRINT B$                        (See Page 162.)
60  END
```

When this program is executed, a string consisting of 6 characters starting from the 6th character from the left of A$(0) is fetched, and CENTER is displayed.
The following precautions should be taken when the MID$ function is used.
When MID$ (character expression, n, m) is entered:
(1) Fractional parts of values of n and m are discarded.
(2) When m is 0 and there is no character to be fetched, a null is provided.

(3) When ",m" is omitted, all the characters starting with the nth digit are provided.
(4) When m exceeds the remaining length of the original string, all characters starting with the nth digit are provided.
(5) When n is larger than the total length of the original string, a null is provided.
(6) Variables and numerical expressions can be used for n and m.
(7) A BS error occurs when n and m are outside the range of

$1 \leqq n < 256$ and $0 \leqq m < 256$

SAMPLE PROGRAM

* A program which produces the frequency count of lower case " r " alphabetical characters in a composition.

```
10 DIM A$(0)*50
20 N=0
30 INPUT "DATA=";A$(0)
40 M=LEN(A$(0))
50 FOR I=1 TO M
60 IF MID$(A$(0),I,1)="r" THEN N=N+1
70 NEXT I
80 PRINT N
90 END
```

In this program, a text entry is made, the number of lower case " r " alphabetical characters is counted and displayed.
Up to 50 characters (including spaces) can be entered in the text.
For example, when the following statement is entered:

Learning ␣ to ␣ master ␣ your ␣ CASIO ␣ Personal ␣ Computer

a check is made of each character to see if it is " r ", and the number of " r " characters is counted. When the text shown above is entered, "5" is displayed. Try this yourself.

SEE LEFT$, RIGHT$

LEN

Function	Provides the length of a character string.
Formats	LEN ("Character string") LEN (Character variable)

LEN is a function that provides the length of a character string assigned to a character variable.

Example

When the following is entered,

CLEAR ↵

PRINT LEN (A$) ↵

"0" is displayed.
This is natural because variable A$ is emptied by the CLEAR command.
If:

AB$ = "CASIO ␣ COMPUTER" ↵

PRINT LEN (AB$) ↵

is entered, "14" is displayed.
The range of values provided by the LEN function is 0 to 79.

SAMPLE PROGRAM

* A character string is displayed with right justification.

```
10 INPUT AB$
20 L=20-LEN(AB$)
30 LOCATE L,3
40 PRINT AB$;:LOCATE 0,0
50 END
```

When character string input is performed in this program, the string is displayed with right justification up to the last column of the screen. Up to 16 characters can be entered for one line.

INKEY$

Function	Provides the entry of 1 character from the keyboard.
Format	INKEY$

The INKEY$ function is one type of input command that resembles INPUT, but its operation is slightly different. With the INKEY$ function, character data produced only by a single key operation can be input. If a key is not pressed, nothing (null) is input and execution proceeds to the next command. It is unnecessary to press the ↵ key to input data. The differences between INKEY$ and INPUT are shown below.

Use	INKEY$	INPUT
Display during execution	Nothing displayed.	Input request message. ("?" display can be eliminated.)
Data input	Key pressed during execution. (No input without pressing key.)	Data entered when ↵ is pressed.
Kinds of data	1 character (All inputs treated as characters.)	Digits or number of characters within the range of a variable such as numerical values, or characters.
Execution of next command	Immediate execution (↵ is unnecessary.)	Suspended until ↵ is pressed.

Example

Since INKEY$ treats key inputs as characters, it is generally used in the form of an assignment expression as follows.

Character variable = INKEY$

```
100  A$=INKEY$
110  IF A$=""THEN 100
120  IF A$="E"THEN END
130  .....
```

When the "E" key is pressed, the program is terminated. When other keys are pressed, execution jumps to the next command. If no key is pressed, execution endlessly loops between lines 100 and 110.
INKEY$ reads all keys except the BRK key, SHIFT, CAPS and F keys. Pressing the SHIFT or CAPS key together with another key produces the corresponding shift mode or capital mode for the pressed. However, one-key command operation produces a null.

SAMPLE PROGRAM

* This is a subroutine which assigns a predetermined number of characters to a character variable.

```
10 REM INKEY$ EXAMPLE
20 AB$=""
30 FOR I=1 TO 10
40 K$=INKEY$
50 IF K$="" THEN 40
60 IF K$="*" THEN 90
70 AB$=AB$+K$
80 NEXT I
90 PRINT AB$
```

The number of characters that can be assigned to a character variable is limited.
If a character string is entered using an INPUT statement, exceeding the input range generates an ST error. However, in this program, when 10 characters have been entered, no further input is accepted and the program moves to the next command (line 90).
Also, to stop input at 10 characters or less, input " * " to move execution to the next line (line 90).

SEE INPUT

PB-770 Only

DMS$

Function	Converts a decimal value to a sexagesimal value and expresses it as a character string.
Format	DMS$ (numerical expression)

This function converts a decimal value to a sexagesimal value and expresses it as a character string. The range of the numerical expression is as follows.

$$|\text{Numerical expression}| < 10^{100}$$

Also, when |numerical expression| ≧ 1E6, minutes and seconds are not displayed (the input value is converted as it is into a character string).

SAMPLE PROGRAM

In the following sample program, input various decimal values to see how they are converted.

```
10 REM DMS$ EXAMPLE
20 INPUT "NUMBER= ";A
30 PRINT "DMS$(";A;")"
40 PRINT "= ";DMS$(A)
50 GOTO 20
```

SEE DEG

PB-770 Only

HEX$

Function	Converts a decimal value to a hexadecimal value and expresses it as a character string.
Format	HEX$ (numerical expression)

This function converts a decimal value to a hexadecimal value and expresses it as a character string. The numerical expression must be within the following range. Fractional values are discarded.

$$-32769 < \text{numerical expression} < 65536$$

The resulting character string is a 4-digit hexadecimal value. Negative numbers are expressed as two's complements.
When a number is 32768 or greater, 65536 is subtracted from it and the result is converted to a hexadecimal value.

Usage

In the following program, input decimal values are converted to hexadecimal values and then displayed. Input some values to confirm proper operation.

```
10 REM HEX$ EXAMPLE
20 INPUT "NUMBER= ",A
30 PRINT "HEX$(";A;")";
40 PRINT "= &H";HEX$(A)
50 GOTO 20
```

SEE &H

TAB

Function	Moves the cursor by specified number of digits to the right.
Format	TAB (Numerical expression) 0 ≦ Numerical expression < 80

This function is used in PRINT and LPRINT statements to move the cursor to a designated position.

```
10 FOR X=1 TO 5
20 PRINT X;"^2:";
30 PRINT TAB(10);X^2
40 NEXT X
```

When you run this program, the display is as follows.

```
△1^2:␣␣␣␣␣△1
△2^2:␣␣␣␣␣△4
      ⋮
└──────────┘
   TAB(10)
```

△ indicates the space where the + sign is omitted.

When TAB (10) is specified as mentioned above, the cursor is moved by 10 display positions, and subsequent display begins after that.
The range that can be specified by the TAB function is 0 to 79 including variables and numerical expressions.
Fractional values are discarded.

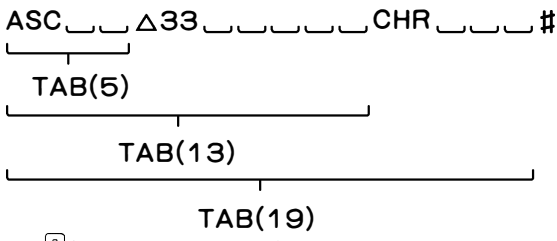
SAMPLE PROGRAM

* Displays a character which corresponds to an ASCII code at a designated location.

```
10 FOR I=33 TO 254
20 PRINT "ASC";
30 PRINT TAB(5);I;
```

```
40 PRINT TAB(13);"CHR";
50 PRINT TAB(19);CHR$(I)
60 NEXT I
70 END
```

When you run this program, the display will be as follows.



When the ↵ key is pressed, the next code and character are displayed at the same position.

Display position

1. Counting the head of the line as 0, count 1 for each space moved to the right.
2. When a tab position to the left of the present screen position is specified, the specified position is determined counting from the beginning of the next line.
3. When a position exceeding the range of 1 screen line is specified, the specified position is determined counting from the beginning of the current line.

NOTE:
When TAB is used during the LPRINT command, the FA-10 or FA-11 printer can execute a normal TAB function, but this is sometimes not true when using CENTRONICS printers connected through the FA-4 interface. When the TAB function is executed, the following data is output to the printer.

$(1B)_H$ + (54) + (numerical expression value) + $(0D)_H$

* $(\ )_H$ represents a hexadecimal value.

When TAB is not used with an exclusive printer, use the function code (Horizontal tab) for that printer.

SEE USING, PRINT, LPRINT, LOCATE

USING

Function	Specifies a display format.
Format	USING "Format character string";

The USING function displays a numerical value or a character string in a PRINT or LPRINT statement in a certain specified format.

Example

When numerical values are displayed in several lines, sometimes digit or decimal point deviations occur. However they can be arranged properly utilizing the USING statement as follows.

```
10 A=18.5
20 B=2.67
30 C=135.78
40 PRINT USING "#####.###";A
50 PRINT USING "#####.###";B
60 PRINT USING "#####.###";C
70 END
```

When you execute this program, the display can be easily read as follows.

```
␣␣␣18.500
␣␣␣␣2.670
␣␣135.780
```

" # " and " . " as used here are the format character string.
USING can also be utilized for a character string display. The format character string is specified with &(s).

```
10 AB$= "CASIO␣COMPUTER" :B$= "!!"
20 PRINT USING "&&&&&&&&&&&&&&&&&&" ;AB$;B$
30 END
```

When you execute this program, the display will be as follows.

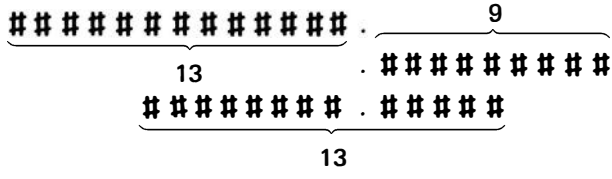
CASIO␣COMPUTER␣␣!!

The number of characters for the CASIO␣COMPUTER is 14, however since a 16 format character string is used, the two extra characters are displayed as blanks.
When a USING statement is used, the following precautions should be taken.

(1) Characters other than # , • , ^ and & cannot be used in a character string format.
(2) # , • , ^ and & cannot be used together.
(3) Numerical format specification

..... Numerical digit specification
• ..... Decimal point specification
^ ..... Exponent specification

(a) # can be specified up to 13 digits before the decimal point and up to 9 digits after the decimal point, and altogether up to13 digits.



(b) The position of the minus sign should also be designated by #.
(c) ^ is specified at the end of a format character string.
(d) When the fractional portion exceeds the format, output is performed by rounding off the next digit of a specified digit.
PRINT USING "###.###"; 12.3456 → 12.346
(e) When the integral portion exceeds the format, % is placed at the beginning to allow output without following the format.
PRINT USING "##.##"; 1234.56 → %␣1234.56
(f) Numerical value output is performed with right justification.

(4) & character string format specification
  (a) & can be written as much as desired.
  (b) If the number of & is smaller than the character string, output is performed from the beginning by the number of & positions.

    PRINT USING "&&&&"; "ABCDEF" → ABCD

  (c) Character string output is performed with left justification.
  (d) Spaces are provided when the number of & is larger than the character string.

(5) One USING specification is only effective in one PRINT or LPRINT statement.
(6) A USING specification is renewed by a new USING specification.
(7) A USING specification can be released by USING " " ; .

SAMPLE PROGRAM

* A program which outputs a person's name, the height, and weight in a certain format to the printer.

```
100 REM USING EXAMPLE
110 CLEAR :DIM A$(2),A!(2),B!(2)
120 FOR I=0 TO 2
130 INPUT "NAME ",A$(I),"HEIGHT (cm) "
    ,A!(I)
140 INPUT "WEIGHT (Kg) ",B!(I)
150 NEXT I
160 FOR I=0 TO 2
170 LPRINT TAB(2);USING"&&&&&&&&&&&&&
    &&";A$(I);
180 LPRINT USING"######";A!(I);"cm";B!
    (I);"Kg"
190 NEXT I
200 END
```

Execution Example

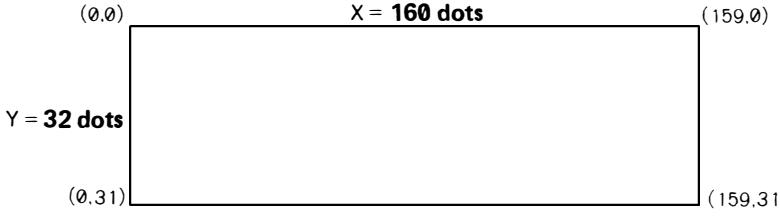
```
JOHN SMITH          190cm     85Kg
BOB JONES            68cm      7Kg
MARY KING           165cm     53Kg
```

POINT

Function	Checks whether a display dot is lit or not.
Format	POINT (X, Y) $0 \leqq X \leqq 159$ (Horizontal position) $0 \leqq Y \leqq 31$ (Vertical position)

A character or sign consists of small square dots on the display screen.
For example, the character A is displayed as follows.

Each point is called a dot.
The entire display consists of 5120 dots.



By assuming that the horizontal direction is the X axis and that the vertical direction is the Y axis, a dot can be displayed using

Coordinate (X, Y)

The POINT function checks whether a dot (X, Y) is lit or not.
When a dot is lit on the coordinate (X, Y), "1" is given and when it is turned off, "0" is given.

Example

```
10  X=0:Y=0
20  A=POINT(X,Y)
30  PRINT  A
```

When you execute this program, if the (0, 0) dot is lit, "1" is displayed, and if it is turned off, "0" is displayed.
The following precautions should be taken when the POINT function is used.
(1) Values rounded at 1 decimal place are used for X and Y.
(2) When X and Y exceed the range of the coordinates, an error (BS error) occurs.

SAMPLE PROGRAM

* Laser gun program.

```
100  CLS :AA$="2499DBFFFFDB9924":BB$="
     00808080C1E3E3F7"
110 X=INT(RND*7)+7
120 LOCATE X,0:PRINT $AA$;:LOCATE 10,3
    :PRINT $BB$;
130 N$=INKEY$
140 IF N$=" " THEN GOSUB 300
150 LOCATE X,0:PRINT " ";
160 GOTO 110
300 DRAW(83,24)-(83,7)
310 A=POINT(83,2)
320 IF A=1 THEN LOCATE 0,3:PRINT "BEE!
    ";:BEEP
330 DRAWC(83,24)-(83,7)
340 FOR I=0 TO 30:NEXT I
350  CLS :RETURN
```

STAT

Function	Allows input of statistical data.
Format	STAT [x data value] [, y data value] [; frequency]

The STAT command is employed for the input of statistical data in the following manner.

(1) STAT x ; n
This format is for the input of single-variable data. The default value for ; n is 1.

(2) STAT x , y ; n
This format is for the input of paired-variable data. The default value for ; n is 1. If either the value for x or y is omitted, the value entered immediately before is used (repeat function). Both x and y values, however, cannot be omitted.

Usage

See 3-18 "STATISTICAL FUNCTIONS".

SEE STAT CLEAR, STAT LIST

STAT CLEAR

Function	Clears the registers used for statistical calculations.
Format	STAT CLEAR

This command clears the registers used for statistical calculations. With this command, the contents of CNT, SUMX, SUMY, SUMX2, SUMY2 and SUMXY all become 0. This command should always be used before beginning a new set of statistical calculations.

SEE STAT

STAT LIST/STAT LLIST

Function	Displays or prints basic statistics.
Formats	STAT LIST STAT LLIST

(1) STAT LIST
Displays data names and values for basic statistics in the order of CNT, SUMX, SUMY, SUMXY, SUMX2, SUMY2.
The output display can be suspended by pressing the ⏎ key.
Pressing this key again will resume output display.
(2) STAT LLIST
The same contents as STAT LIST are output to the printer.

SEE STAT

CNT

Function	Gives the number of statistically processed data (n).
Format	CNT

The CNT function gives the number of statistically processed data.

Reference

Refer to section 3-18 for information on the use of the following statistical functions.

COR

Function	Gives the correlation coefficient (r).
Format	COR

COR gives a correlation coefficient (r) as a numerical expression shown below.

$$\text{COR}: \frac{n\cdot\Sigma xy-\Sigma x\cdot\Sigma y}{\sqrt{\{n\cdot\Sigma x^2-(\Sigma x)^2\}\ \{n\cdot\Sigma y^2-(\Sigma y)^2\}}}$$

(n: number of data)

SUMX
SUMY
SUMX2
SUMY2
SUMXY

Functions	SUMX: Gives sum of x data. SUMY: Gives sum of y data. SUMX2: Gives sum of squares of x data. SUMY2: Gives sum of squares of y data. SUMXY: Gives sum of products of x data and y data.
Formats	SUMX SUMY SUMX2 SUMY2 SUMXY

These functions give sums, sums of squares and sums of products.

SUMX : Σx

SUMY : Σy

SUMX2 : Σx^2

SUMY2 : Σy^2

SUMXY : Σxy

MEANX
MEANY

Functions	MEANX: Gives mean value of x data. MEANY: Gives mean value of y data.
Formats	MEANX MEANY

These functions give the mean values of data.

MEANX : $\Sigma x/n$

MEANY : $\Sigma y/n$

SDX
SDY
SDXN
SDYN

Functions	SDX: Gives sample standard deviation of x data. SDY: Gives sample standard deviation of y data. SDXN: Gives population standard deviation of x data. SDYN: Gives population standard deviation of y data.
Formats	SDX SDY SDXN SDYN

These functions give sample standard deviations and population standard deviations as numerical expressions shown below.

$$\text{SDX} : \sqrt{\frac{n\cdot\Sigma x^2-(\Sigma x)^2}{n(n-1)}} \quad (x\sigma_{n-1})$$

$$\text{SDY} : \sqrt{\frac{n\cdot\Sigma y^2-(\Sigma y)^2}{n(n-1)}} \quad (y\sigma_{n-1})$$

$$\text{SDXN} : \sqrt{\frac{n\cdot\Sigma x^2-(\Sigma x)^2}{n^2}} \quad (x\sigma_{n})$$

$$\text{SDYN} : \sqrt{\frac{n\cdot\Sigma y^2-(\Sigma y)^2}{n^2}} \quad (y\sigma_{n})$$

(n: number of data)

EOX
EOY

Functions	EOX: Gives an estimated value of x in terms of y. EOY: Gives an estimated value of y in terms of x.
Formats	EOX numerical expression EOY numerical expression

These functions give estimated values as numerical expressions shown below.

$$\text{EOX(y)} : \frac{y - \text{LRA}}{\text{LRB}} \quad (\hat{x})$$

$$\text{EOY(x)} : \text{LRA} + x \cdot \text{LRB} \quad (\hat{y})$$

LRA
LRB

Functions	LRA: Gives linear regression constant term. LRB: Gives linear regression coefficient.
Formats	LRA LRB

These functions give linear regression constant term and linear regression coefficient.

$$\text{LRA} : \frac{\Sigma y - \text{LRB} \cdot \Sigma x}{n}$$

$$\text{LRB} : \frac{n \cdot \Sigma xy - \Sigma x \cdot \Sigma y}{n \cdot \Sigma x^2 - (\Sigma x)^2}$$

(n: number of data)

PB-770 Only

&H

Function	Converts a hexadecimal value (up to 4 digits) to a decimal value.
Format	&H hexadecimal value

This function converts a hexadecimal value up to four digits long to a decimal value. The following shows a number of examples.

Hexadecimal	Decimal
&H1 [ENTER]	1
&HA [ENTER]	10
&H000B [ENTER]	11
&HABCD [ENTER]	−21555
&HG [ENTER]	Error
&H12345 [ENTER]	Error

&H is not considered a function in BASIC.

Usage

(1) Manual calculation
Affix a hexadecimal value to &H and press the [ENTER] key.

Example: &H1B7F [ENTER] → 7039

(2) Program
The following shows a sample program application. A variable cannot be used after &H, so a hexadecimal character string is affixed to &H and then the VAL function is used to convert to a decimal value.

```
10 REM &H EXAMPLE
20 INPUT "&H";A$
30 H=VAL("&H"+A$)
40 PRINT "&H ";A$;"=";H
50 GOTO 10
```

SEE HEX$, VAL

CHAPTER 5

PROGRAM LIBRARY

PLEASE NOTE:

Programs in this chapter may be used freely without permission. However, it must be understood that the company is not responsible for any damage or loss as a result of using these examples.

In the case of executing programs in this chapter without the optional plotter-printer, press "N" when the PB-770 requests whether printouts are made or not by displaying "PRINTER ON? (Y/N)".

STOCK PRICE MANAGEMENT AND PROPER SELLING/BUYING PRICES

This program stores stock prices for the past 53 weeks. After 53 weeks of data have been input, each time data of a new week is entered, the data of the oldest week is discarded. Based on the stock price data, the program outputs the current deviation value and helpful information for judgment on buying or selling. The program also permits display of the deviation value and moving average, and graphically displays stock price fluctuations.

Explanation

First, start the program in P0, and the menu will be displayed on the screen. Then, enter a number from 1 to 7 given with the menu. Entering a number other than 1 to 7 causes the menu to be displayed again.

(1) Data input

To input data, press [1] after the menu is displayed, and "Initial? (Y/N)" is displayed. If you are inputting data for the first time, press the [Y] key. If you are inputting data following other data, press the [N] key. When the [Y] key is pressed, "CLEAR OK? (Y/N)" is displayed on the screen. This is to prevent data from being lost by erroneous input. Normally, press the [Y] key.
For initial input, "1) DATA=" is displayed on the screen. Enter the appropriate stock price and press the key. For the second and subsequent data inputs, the screen displays "WEEK=". For input of data for the 54th and subsequent weeks, the oldest data is sequentially erased. Therefore, the time required for input becomes a little longer. To terminate data input, enter a negative number, and the menu will be displayed again.

(2) Judgment on sell or buy

When menu 2 is selected, "Current Price? " is displayed. At this time, enter the current stock price, and the program compares the current stock price with the past data and outputs the deviation value. To exit from this routine, enter a negative value. The menu will be displayed again.

(3) Checking reasonable stock price

If new data has been entered using this routine, routine (2) must be executed before correct values can be output. This is because the two routines share part of the same variables. Entering a negative number causes the menu to be displayed again.

(4) Data output

When this menu is selected, all the data stored in memory are displayed on the screen, then the menu appears again.

(5) Moving average

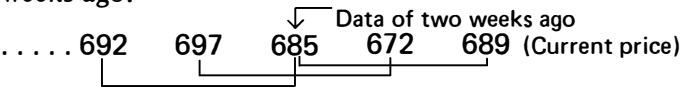
This routine calculates the moving average. When "No. of movements? " is displayed, enter the number of weeks for which the moving average is to be calculated. The routine calculates the moving average for the period between the current week and the specified week. After this routine is executed, the menu is automatically displayed.

(6) Past moving average

This routine permits reviewing the change in moving average in the past, so this can be used to determine whether the stock price is rising or falling.
When "No. of movements? " is displayed, enter the number of weeks for which the moving average is to be calculated. Then, "FROM WHEN? " is displayed. Here, enter the week from which the number of weeks is to be counted.

Example:

Calculating the moving average for each three weeks, starting from two weeks ago.

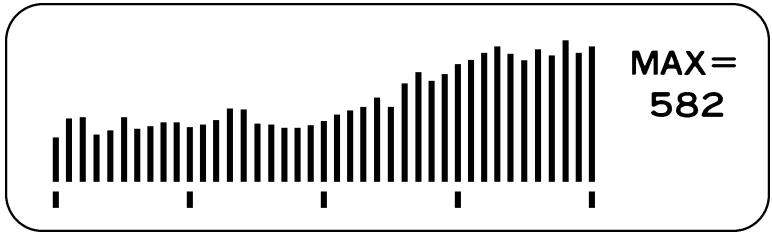


The menu automatically appears after executing this routine. Be careful when entering data in routines (5) and (6) to prevent incorrect values from being output, especially when the amount of data (the number of weeks) stored is relatively small.

(7) Graph

This routine graphically displays the stored data to permit easy recognition of the general trend of the stock price.
The menu display appears again after executing this routine.

Execution Example of a Graph Display



Program

```
P0
  10  CLS
  20 GOSUB 80
  30 INPUT "INPUT    NO.";PR
  40 IF PR>7 THEN 10 ELSE IF PR<1 THEN
     10
  50 IF PR=1 THEN GOSUB PROG 1 ELSE IF
     PR=2 THEN GOTO PROG 2
  60 IF PR=3 THEN GOSUB PROG 3 ELSE IF
     PR=4 THEN GOTO PROG 4
  70 IF PR=5 THEN GOSUB PROG 5 ELSE IF
     PR=6 THEN GOTO PROG 6 ELSE GOTO PR
     OG 7
  80 PRINT :PRINT "DATA INPUT 1"
  90 FOR I=1 TO 100:NEXT I
 100 PRINT "PRICE CHECK 2"
 110 FOR I=1 TO 100:NEXT I
 120 PRINT "REASONABLE PRICE 3"
 130 FOR I=1 TO 100:NEXT I
 140 PRINT "DATA OUTPUT 4"
 150 FOR I=1 TO 100:NEXT I
 160 PRINT "MOVING AVE  5"
 170 FOR I=1 TO 100:NEXT I
 180 PRINT "PAST MOVEMENT 6"
 190 FOR I=1 TO 100:NEXT I
 200 PRINT "GRAPH DISPLAY 7"
 210 RETURN
P1
  10  CLS
  20 PRINT " ** DATA INPUT **"
  30 INPUT "Initial?(Y/N)",P$:IF P$="Y"
      THEN GOSUB 200
  40 IF P$="N" THEN A=A-1 ELSE IF P$<>"
     Y" THEN 30
  50 A=A+1:GOSUB 300
```

```
  60 IF A>=53 THEN 80
  70 INPUT "DATA";Z(A):IF Z(A)<0 THEN Z
     (A)=0:GOTO PROG 0 ELSE GOTO 50
  80 INPUT "DATA+";DZ:C=C+1:A=53:IF DZ<
     0 THEN GOTO PROG 0 ELSE Z(53)=DZ
  90 FOR B=1 TO 53
 100 Z(B-1)=Z(B)
 110 NEXT B
 120 GOTO 80
 200 INPUT "CLEAR OK?(Y/N)";T$:IF T$="Y
     " THEN 220 ELSE IF T$="N" THEN 260
 210 GOTO 200
 220 ERASE Z:DIM Z(53)
 230 PRINT "STOCK PRICE "
 240 FOR K=0 TO 50:NEXT K
 250 INPUT "1)DATA";Z(1):A=1
 260 P$="Y":RETURN
 300 PRINT "WEEK=";A
 310 FOR K=0 TO 10:NEXT K:RETURN
P2
  10  CLS
  20 PRINT " ** PRICE CHECK **"
  30 S=0:Q=0
  40 FOR D=0 TO 52
  50 S=S+Z(D):Q=Q+Z(D)^2
  60 NEXT D
  70 IF A<54 THEN 90
  80 E=S/53:V=Q-53*E*E:F=SQR(V/52)
  90 E=S/A:V=Q-A*E*E:F=SQR(V/(A-1))
 100 INPUT "Current Price";Y:IF Y<0 THE
     N 140
 110 D=ROUND(50+10*(Y-E)/F,-3)
 120 PRINT "Deviation=";D
 130 GOTO 100
 140 GOTO PROG 0
P3
  10  CLS
  20 PRINT "**REASONABLE PRICE**"
```

```
  30 INPUT "Deviation=";D:Y=ROUND((D-50
     )*F/10+E,-2):IF D<0 THEN 50
  40 PRINT " PRICE=";Y:GOTO 30
  50 GOTO PROG 0
P4
  10  CLS
  20 PRINT " ** DATA OUTPUT **"
  30 FOR U=0 TO A-2
  40 PRINT "DATA";U+1;"=";Z(U+1)
  50 FOR K=0 TO 50
  60 NEXT K:NEXT U
  70 PRINT "DATA END"
  80 FOR K=1 TO 200:NEXT K
  90 GOTO PROG 0
P5
  10  CLS
  20 PRINT " ** MOVING AVE. **"
  30 X=0
  40 INPUT "No. of movements";N
  50 IF A<=53 THEN 100
  60 FOR L=53-N TO 52
  70 X=X+Z(L)
  80 NEXT L
  90 GOTO 130
 100 FOR K=A-N TO A
 110 X=X+Z(K)
 120 NEXT K
 130 M=X/N
 140 PRINT "MOVING AVERAGE";M
 150 FOR K=0 TO 300:NEXT K
 160 GOTO PROG 0
P6
  10  CLS
  20 PRINT "** PAST MOVEMENT **"
  30 INPUT "No. of movements";I
  40 INPUT "FROM WHEN ";O
  50 X=0
  60 IF A<=53 THEN 110
```

```
 70 FOR J=53-0-I TO A-0-I
 80 X=X+Z(J)
 90 NEXT J
100 GOTO 140
110 FOR J=A-0-I TO A-0-1:IF A<=J THEN
    180
120 X=X+Z(J)
130 NEXT J
140 M=X/I
150 FOR K=0 TO 100:NEXT K
160 PRINT "Moving Ave.";M
170 0=0-1:X=0:FOR K=0 TO 10:NEXT K:GOT
    O 60
180 PRINT " END"
190 FOR K=0 TO 300:NEXT K
200 GOTO PROG 0
P7
 10  CLS
 20 PRINT "** GRAPH DISPLAY **"
 30 MX=0:FOR MD=1 TO 53
 40 IF Z(MD)>MX THEN MX=Z(MD)
 50 NEXT MD
 60  CLS :LOCATE 15,1:PRINT "MAX=":LOC
    ATE 14,2:PRINT INT(MX)
 70 FOR K=1 TO 5
 80 PO=130-K*20:DRAW(PO,27)-(PO,30)
 90 NEXT K
100 FOR K=1 TO A-1
110 J1=K*2+10: J2=25-25/MX*Z(K)
120 DRAW(J1,J2)-(J1,25)
130 NEXT K
140 IF INKEY$="" THEN 140 ELSE GOTO PR
    OG 0
```

Number of bytes used: 1872

Example Data							
19 weeks ago	584	14 weeks ago	545	9 weeks ago	635	4 weeks ago	685
18 weeks ago	580	13 weeks ago	550	8 weeks ago	652	3 weeks ago	697
17 weeks ago	579	12 weeks ago	563	7 weeks ago	673	2 weeks ago	685
16 weeks ago	570	11 weeks ago	589	6 weeks ago	701	Last week	672
15 weeks ago	562	10 weeks ago	620	5 weeks ago	692	This week	689

Variable contents					
A	Data counter	J 2	Y axis of graph	Q	Sum of squares of data
B ~ D	Counters	MD	Counter		
DA	Character data	MX	Max. data	S	Sum of data
DZ	Stock price	N	No. of movements	T$	CLEAR OK?
E	Average	O	Starting week for calculating of moving average	U	Counter
H	Deviation value			V	Variance
J ~ L	Counters (for periods to calculate moving average in P5)			X	Counter
		P$	Y or N (Initial?)	Z (X)	Stock price
		P0	Graph scale		
J 1	X axis of graph	PR	Program selection		

Operation

	Step	Key operation	Display
Menu display →		SHIFT P0	DATA INPUT 1 PRICE CHECK 2 REASONABLE PRICE 3 DATA OUTPUT 4 MOVING AVE. 5 PAST MOVEMENT 6 GRAPH DISPLAY 7 INPUT NO.? _
Selects data input. →	1	1 [↵]	**DATA INPUT** Initial ? (Y/N)_
Press [Y] for initial data input and [N] for additional data input. →	2	[Y] [↵]	CLEAR OK ? (Y/N) _
Press [Y] to clear the stored data and [N] not to clear them. →	3	[Y] [↵]	STOCK PRICE 1) DATA ? _
(1) Data input	4	584 [↵]	WEEK=2 DATA ? _
Enter a negative number to terminate data input. →	5	−1 [↵]	INPUT NO. ? _
(2) Stock price check	6	2 [↵]	**PRICE CHECK** Current Price ? _
Enter the current stock price and its deviation value is displayed based on the past data. →	7	585 [↵]	Deviation =49.23 Current Price ? _
Entering a negative number causes the menu to be displayed. →	8	−1 [↵]	INPUT NO. ? _
(3) Reasonable stock price	9	3 [↵]	**REASONABLE PRICE** Deviation= ? _
Enter a deviation value. →	10	55 [↵]	PRICE=670.1 Deviation= ? _
The stock price is displayed. →			
Entering a negative number causes the menu to be displayed. →	11	−1 [↵]	INPUT NO. ? _

	Step	Key operation	Display
(4) Data output All data are displayed and then the menu display appears. Menu display	12	4 ↵	**DATA OUTPUT** DATA 1=584 DATA 2=580 DATA 3=579 ⋮ DATA END INPUT NO.? _
(5) Moving average	13	5 ↵	**MOVING AVE.** No. of movements ? _
Enter the number of movements (weeks).	14	15 ↵	MOVING AVERAGE 643.2 INPUT NO.? _
(6) Moving average in the past. Enter the number of movements (weeks) for which the moving average is to be calculated.	15	6 ↵	**PAST MOVEMENT** No. of movements ? _
	16	3 ↵	FROM WHEN ? _
Enter the starting week. Each moving average is displayed and then the menu display appears.	17	2 ↵	Moving Ave. 689 Moving Ave. 684.666…… Moving Ave. 682 END INPUT NO.? _
(7) Graph	18	7 ↵	A graph is displayed.

TELEPHONE DIRECTORY

This program permits immediate recall of a desired telephone number by entering previously stored names. It also permits recalling a phone number using the initial letter of a name. Names can also be arranged in alphabetical order.

Explanation

This program can be used to store and recall the phone numbers of your friends and acquaintances. Up to 180 names can be stored at a time. Once the names and phone numbers have been stored, it is possible to recall the desired phone number merely by entering the initial letter or the first few letters of the name.
When the RAM capacity is expanded to 16KB or more using the RAM expansion pack(s), up to 255 names can be stored. In this case, it is necessary to change the program in P1 as follows.

(8KB)

```
30 IF N=181 THEN 90
40 IF N=1 THEN DIM A$(180), B$(180) * 12
```

(16KB or more)

```
30 IF N=256 THEN 90
40 IF N=1 THEN DIM A$(255), B$(255) * 12
```

First, enter CLEAR [↵] after inputting all of the programs.
Then, execute the program in P0, and the menu is displayed on the screen. Enter the appropriate number, 1 to 4. There is no need to press the [↵] key. If an OM error (Out of Memory) occurs in line 40 of P1, erase the other programs or data or expand the RAM capacity using the RAM expansion pack(s).

(1) INPUT

Stores names and phone numbers. Input data by the following procedure. Data to be input is underlined.

```
NAME?    CASIO [↵]
TEL NO.? 123-4567 [↵]
```

Input all names and phone numbers by repeating the above procedure. When the last name and phone number are entered, enter END [↵]. The menu is displayed again.

(2) SORTING

Arranges the stored names in alphabetical order. While the names are being sorted, 'SORTING . . . ' is displayed on the screen. The sort operation is completed in several seconds to a few minutes depending on the amount of data stored. The sorted names (and the associated phone numbers) can be sequentially displayed on the screen by pressing any key on the keyboard. After all the names and phone numbers are displayed, the menu is displayed again.

(3) LOOK FOR

Recalls the phone number when a name is entered.
Enter the name as follows:

NAME? CASIO ⏎

The name and phone number are displayed as follows:

CASIO
123-4567

Note that only the initial letter or the first few letters of the name may be entered to recall the phone number. In this case, if there is more than one name with the same initial letter or first few letters, all of them are displayed. Note also that if identical names are stored, they are all displayed. If any name which has not been stored is entered, 'NO DATA' is displayed on the screen. In this case, press any key on the keyboard to return to the menu.

(4) DELETE

Deletes data which has been stored.
Enter the name to be deleted as follows:

NAME? ABCDE ⏎

Then, the screen displays:

ABCDE
XXX–XXXX Y/N? (XXX–XXXX: phone number of ABCDE)

If you really wish to delete the name and phone number, press the [Y] key. If not, press the [N] key. This is to assure you that no data is erroneously deleted.
To clear all the data which have been stored or to cancel a DD error, execute the CLEAR command. Since menu 1 (INPUT) has the function to add data, it may be combined with menu 4 (DELETE) to add or delete data freely.

Program

```
P0
  10  CLS
  20 PRINT "1-INPUT    2-SORTING"
  30 PRINT "3-LOOK FOR"
  40 PRINT "4-DELETE"
  50 K$=INKEY$:IF K$="" THEN 50
  60 IF K$="1" THEN GOTO PROG 1
  70 IF K$="2" THEN GOTO PROG 2
  80 IF K$="3" THEN GOTO PROG 3
  90 IF K$="4" THEN GOTO PROG 4
 100 GOTO 50
P1
  10  CLS
  20 N=N+1
  30 IF N=181 THEN 90
  40 IF N=1 THEN DIM A$(180),B$(180)*12
  50 INPUT "NAME    ";A$(N)
  60 IF A$(N)="END" THEN 100
  70 INPUT "TEL NO.";B$(N)
  80 GOTO 10
  90 PRINT "FULL":BEEP 1
 100 N=N-1
 110 GOTO PROG 0
P2
   5  CLS :PRINT "SORTING..."
  10 FOR I=1 TO N
  20 MM$=A$(I):X=I
  30 FOR J=I TO N
  40 KK$=A$(J)
  50 GOSUB 200
  60 NEXT J
  70 A$(X)=A$(I):A$(I)=MM$
  80 MM$=B$(X)
  90 B$(X)=B$(I):B$(I)=MM$
 100 NEXT I
```

```
 110 GOTO PROG 5
 200 KU=0
 210 KU=KU+1
 230 O1=LEN(MM$):O2=LEN(KK$)
 240 IF KU>O1 THEN RETURN  ELSE IF KU>O
     2 THEN 300
 250 MI$=MID$(MM$,KU,1):KI$=MID$(KK$,KU
     ,1)
 260 IF ASC(MI$)=ASC(KI$) THEN 210
 270 IF ASC(MI$)>ASC(KI$) THEN X=J:MM$=
     KK$:RETURN
 280 RETURN
 300 X=J:MM$=KK$:RETURN
P3
  10  CLS
  20 INPUT "NAME  ";MM$
  30 X=LEN(MM$)
  40 I=0
  50 I=I+1
  60 IF I=N+1 THEN IF F=1 THEN 120 ELSE
      130
  70 IF MM$=LEFT$(A$(I),X) THEN 90
  80 GOTO 50
  90 F=1:PRINT A$(I)
 100 PRINT B$(I)
 110 K$=INKEY$:IF K$="" THEN 110 ELSE 5
     0
 120 F=0:GOTO PROG 0
 130 PRINT "NO DATA"
 140 K$=INKEY$:IF K$="" THEN 140 ELSE 1
     20
P4
  10  CLS
  20 INPUT "NAME  ";MM$
  30 X=LEN(MM$)
  40 I=0
  50 I=I+1
  60 IF I=N+1 THEN 180
```

```
 70 IF MM$=LEFT$(A$(I),X) THEN 100
 90 GOTO 50
100 PRINT A$(I)
110 PRINT B$(I);" Y/N ";
120 INPUT K$
130 IF K$="Y" THEN 150
140 GOTO 50
150 A$(I)=A$(N)
160 B$(I)=B$(N)
170 N=N-1
180 GOTO PROG 0
P5
 10 FOR I=1 TO N
 20  CLS
 30 PRINT A$(I)
 40 PRINT B$(I)
 50 K$=INKEY$:IF K$="" THEN 50
 60 NEXT I
 70 GOTO PROG 0
```

Number of bytes used: 1003

Operation

	Step	Key operation	Display
Menu display →		SHIFT P0	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(1) INPUT Select menu 1 →	1	1	NAME ?_
Input the first person's name. →	2	SMITH, JOHN ↵	NAME ? SMITH, JOHN TEL NO. ?_
Input telephone number. →	3	03-583-4111 ↵	NAME ?_
Input the second person's name. →	4	BROWN, MARY ↵	NAME ? BROWN, MARY TEL NO. ?_
Input telephone number. →	5	052-264-1453 ↵	NAME ?_ ⋮
After completing data input, enter END ↵ and then the menu is displayed on the screen. →	6	END ↵	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(3) LOOK FOR Whose telephone number do you want to know? →	7	3	NAME ?_
Only a family name can be entered. →	8	SMITH, JOHN ↵	NAME ? SMITH, JOHN SMITH, JOHN 03-583-4111
The menu is displayed on the screen. →	9	↵	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(2) SORTING Alphabetical order →	10	2	SORTING...
After "SORTING..." disappears →	11	↵	ALLEN, ROBERT 06-314-2681
The sorted names can be sequentially displayed. →	12	↵	BROWN, MARY 052-264-1453 ⋮

Step	Key operation	Display
13	↵	1−INPUT 2−SORTING 3−LOOK FOR 4−DELETE
14	4	NAME ?_
15	SMITH, JOHN ↵	NAME ? SMITH, JOHN SMITH, JOHN 03-583-4111 Y/N ?_
16	Y	1−INPUT 2−SORTING 3−LOOK FOR 4−DELETE
17	3	NAME ?_
18	SMITH, JOHN ↵	NAME ? SMITH, JOHN NO DATA

After all the names and phone numbers are displayed, the menu is displayed.

(4) DELETE
What name do you want to delete?

If you really want to delete, press Y. If not, press N .

The menu is displayed.

Confirm that the data is deleted using menu 3.

"NO DATA" indicates that the name has not been stored.

Variable contents	
N	(Number of NAMEs/TEL NOs)−1
X	Length of character string to be looked for
A$(1) ≀ A$(180)	Names
B$(1) ≀ B$(180)	Telephone numbers
MM$	Character string to be looked for
KK$ KU O1 O2	For sorting

CROSS TOTAL

This program obtains the sums of horizontal (X) and vertical (Y) data, or sorts the data to determine the percentage of each data element. For example, item X may be a certain product and item Y may be a certain month.

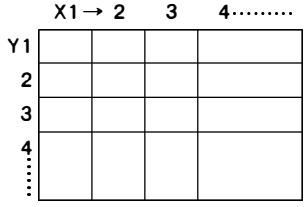
Explanation

First, execute the program P0, and the following menu is displayed on the screen:

```
1  DATA INPUT          ←For inputting data
2  TOTAL               ←For obtaining horizontal and vertical totals
3  SORT                ←For arranging data
4  DATA OUTPUT ?_      ←For checking all the data
```

(1) Data input

When menu 1 (DATA INPUT) is selected, "CLEAR (Y/N)? " is displayed. To input data for the first time, press [Y]. Then, the program requests you to input the values of X and Y (the value of X is the number of data elements in horizontal direction, and the value of Y is the number of data elements in vertical direction).
Input the appropriate values.
After the values of X and Y have been input, you are requested to input the data. Input the data according to the element numbers displayed on the screen:



X = 1, Y = 1 → X = 2, Y = 1 → X = 3, Y = 1 . . .
X = 1, Y = 2 → X = 2, Y = 2 → X = 3, Y = 2 . . .

After all data are input, the grand total is displayed and the menu display appears.
If the data entered contains an error, enter 1 again. When "CLEAR (Y/N)? " is displayed, enter N. Then, the program asks you about the element number whose associated data is to be corrected. Input the appropriate element number, and "DATA? " is displayed. Input the correct data. The correct grand total is displayed and the menu display appears.

(2) Sum of data in item X or Y

When menu 2 (TOTAL) is selected, "PRINTER ON? (Y/N)" is displayed. The subtotals are printed by entering Y. The program asks you whether you wish to obtain the sum of item X or item Y. Input X or Y, whichever is appropriate.

When X is input, the program outputs the subtotals of X from 1 to the preset value, and returns to the menu display after outputting the grand total. When Y is entered, the program outputs the subtotals of Y from 1 to the preset value, and returns to the menu display after outputting the grand total.

(3) Sorting

When menu 3 (SORT) is selected, the program asks you whether the subtotals of X or Y are to be sorted after displaying "PRINTER ON? (Y/N)". When X or Y is input, "SORTING NOW" is displayed and a sort operation is started. The data sorted in descending order is output, together with the ranking, item name, subtotal and percentage of each element. Then the menu is displayed. The sort operation requires some time. For example, it takes approximately one minute and 10 seconds to sort 20 data elements. When the sort operation is completed, a buzzer sounds and the data output begins. After execution of this program, the menu display appears again.

Since this program uses many half-precision variables, it can handle a relatively large volume of data. Note, however, that the maximum number of digits of input data is five.

To review the result of sorting, press the BRK key and run line 140 of the program P4.

(4) Data output

When menu 4 (DATA OUTPUT) is selected, data such as "X=1 Y=1 DATA=233" is displayed after displaying "PRINTER ON? (Y/N)". After all the data are displayed, the menu display appears again.

Program

```
P0
  10 PRINT "1 DATA INPUT","2 TOTAL","3
     SORT","4 DATA OUTPUT ";
  20 INPUT R
  30 IF R=1 THEN GOTO PROG 1
  40 INPUT "PRINTER ON?(Y/N)",F$
  50 IF R=2 THEN GOTO PROG 2
  60 IF R=3 THEN GOTO PROG 3 ELSE IF R=
     4 THEN GOTO PROG 5
P1
  10 INPUT "CLEAR (Y/N)?",S$
  20 IF S$="Y" THEN CLEAR :GOTO 30 ELSE
      IF S$<>"N" THEN 10 ELSE 110
  30 INPUT "X";X,"Y";Y
  40 DIM D!(X,Y),X!(X),Y!(Y)
  50 FOR J=1 TO Y:Y!(J)=0:FOR I=1 TO X
  60 PRINT "INPUT DATA X=";I;" Y=";J,
  70 INPUT D!(I,J)
  80 Y!(J)=Y!(J)+D!(I,J)
  90 NEXT I:NEXT J
 100 GOSUB 200:GOTO PROG 0
 110 PRINT "CORRECTION(X,Y)";
 120 INPUT I,J
 130 INPUT "DATA";D!(I,J)
 140 FOR J=1 TO Y:Y!(J)=0:FOR I=1 TO X
 150 Y!(J)=Y!(J)+D!(I,J)
 160 NEXT I:NEXT J
 170 GOSUB 200:GOTO PROG 0
 200 S=0:FOR I=1 TO X:X!(I)=0:FOR J=1 T
     O Y
 210 X!(I)=X!(I)+D!(I,J):S=S+D!(I,J)
 220 NEXT J:NEXT I
 230 PRINT "GRAND T.";S:FOR K=0 TO 100:
     NEXT K
```

```
 240 RETURN
P2
  10 INPUT "X-SUM OR Y-SUM";P$
  20 IF P$="Y" THEN 160 ELSE IF P$="X"
     THEN 80 ELSE 10
  80 FOR K=1 TO X
  90 PRINT "X=";K;" SUM=";X!(K)
  95 IF F$="Y" THEN GOSUB 300
  97 IF INKEY$="" THEN 97 ELSE 100
 100 NEXT K
 110 PRINT "GRAND T.=";S
 115 IF F$="Y" THEN GOSUB 340
 120 IF INKEY$="" THEN 120 ELSE 130
 130 GOTO PROG 0
 160 FOR K=1 TO Y
 170 PRINT "Y=";K;" SUM=";Y!(K)
 173 IF INKEY$="" THEN 173 ELSE 175
 175 IF F$="Y" THEN GOSUB 320
 180 NEXT K
 190 PRINT "GRAND T.=";S
 195 IF F$="Y" THEN GOSUB 340
 200 IF INKEY$="" THEN 200 ELSE 210
 210 GOTO PROG 0
 300 LPRINT "X=";K;" SUM=";X!(K):RETURN

 320 LPRINT "Y=";K;" SUM=";Y!(K):RETURN

 340 LPRINT "GRAND T.";S
 350 GOTO PROG 0
P3
  10 INPUT "SORT X OR Y?",P$
  20 IF P$="Y" THEN GOSUB 100 ELSE IF P
     $="X" THEN GOSUB 200 ELSE 10
  30 GOTO PROG 4
 100 ERASE A!
 110 DIM A!(Y,2)
 120 FOR J=1 TO Y
```

```
 130 A!(J,1)=Y!(J):A!(J,2)=J
 140 NEXT J
 150 N=Y:RETURN
 200 ERASE A!
 210 DIM A!(X,2)
 220 FOR I=1 TO X
 230 A!(I,1)=X!(I):A!(I,2)=I:NEXT I
 240 N=X:RETURN
P4
  10  CLS
  20 PRINT "SORTING NOW"
  30 REM SORT
  40 FOR K=N-1 TO 1 STEP -1
  50 FOR L=1 TO K
  60 IF A!(L,1)>A!(L+1,1) THEN 100
  65 FOR M=1 TO 2
  70 T=A!(L,M)
  80 A!(L,M)=A!(L+1,M)
  90 A!(L+1,M)=T
  95 NEXT M
 100 NEXT L
 110 NEXT K
 120 REM PRINT
 130 FOR K=1 TO 10:BEEP :NEXT K: CLS
 140 FOR K=1 TO N:GOSUB 220
 150 PRINT USING"##";K;" ";P$;"=";USING
     "##";A!(K,2);USING"########";A!(K,
     1);
 160 PRINT USING"###";A;"%"
 170 IF F$="Y" THEN GOSUB 300:NEXT K:GO
     SUB 320:GOTO PROG 0
 180 FOR L=1 TO 300:NEXT L:NEXT K
 190 PRINT "GRAND T.";S
 200 FOR K=0 TO 300:NEXT K
 210 GOTO PROG 0
 220 REM RATIO
 230 A=ROUND(A!(K,1)/S,-3)*100
```

```
 240 RETURN
 300 LPRINT USING"##";K;" ";P$;"=";USIN
     G"##";A!(K,2);USING"########";A!(K
     ,1);
 310 LPRINT USING"###";A;"%":RETURN
 320 PRINT "GRAND T.";S
 330 LPRINT "GRAND T.";S
 340 RETURN
P5
   5  CLS
  10 FOR J=1 TO Y:FOR I=1 TO X
  20 PRINT "X=";I;" Y=";J;" DATA=";D!(I
     ,J)
  30 IF F$="Y" THEN 50
  40 IF INKEY$="" THEN 40 ELSE 60
  50 LPRINT "X=";I;"  Y=";J;" DATA=";D!
     (I,J)
  60 NEXT I:NEXT J
  70 IF F$="Y" THEN LPRINT :LPRINT :LPR
     INT
  80 GOTO PROG 0
```

Number of bytes used: 1737

Variable contents					
A	Percentage.	I, J	Array subscripts.	S$	Y or N.
A! ( )	For storing data during sorting.	K~M	Counters.	T	Variable for data exchange.
		N	Number of data elements during sorting.	X! ( )	Array for sums of X.
D! ( )	Data array.			Y! ( )	Array for sums of Y.
F$	Determines whether the printer is used or not.	P$	X or Y.		
		R	Menu selection.		
		S	Grand total.		

* If you wish to handle data more than 5 digits, change variables A! ( ), D! ( ), X! ( ) and Y! ( ) as follows: A( ), D( ), X( ) and Y( )

Sample Data

```
X= 1  Y= 1  DATA= 321
X= 2  Y= 1  DATA= 369
X= 3  Y= 1  DATA= 357
X= 1  Y= 2  DATA= 159
X= 2  Y= 2  DATA= 147
X= 3  Y= 2  DATA= 123
X= 1  Y= 3  DATA= 842
X= 2  Y= 3  DATA= 862
X= 3  Y= 3  DATA= 579
```

Operation

	Step	Key operation	Display
Menu display →	1	SHIFT P0	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?
(1) Data input Do you want to clear the stored data? →	2	1 ↵	CLEAR(Y/N)? _
How many horizontal items? →	3	Y ↵	X? _
How many vertical items? →	4	3 ↵	Y? _
Input data (X=1, Y=1). →	5	3 ↵	INPUT DATA X=1 Y= 1 ? _

	Step	Key operation	Display
Input data (X=2, Y=1).	6	321 ↵ ⋮ Input all the data by repeating above.	INPUT DATA X= 2 Y= 1 ?_ GRAND T. 3759
Grand total display			
Menu display (2) Sum of data in item X	7		1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
Is the printer used? Enter X ↵ for X subtotals and Y ↵ for Y subtotals.	8	2 ↵ N ↵	PRINTER ON?(Y/N)_ X−SUM OR Y−SUM?_
Output of each subtotal (SUM) and grand total.	9	X ↵ ↵ ↵ ↵	X=1 SUM=1322 X=2 SUM=1378 X=3 SUM=1059 GRAND T. =3759
Menu display (2) Sum of data in item Y	10	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
Is the printer used?	11	2 ↵ N ↵	PRINTER ON?(Y/N)_ X−SUM OR Y−SUM?_
Enter Y ↵ for Y subtotals.	12	Y ↵ ↵ ↵ ↵	Y=1 SUM=1047 Y=2 SUM=429 Y=3 SUM=2283 GRAND T. =3759
Menu display	13	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_

(3) Sorting

If the printer is connected, enter Y ↵ .

Arrangement for each subtotal (X or Y)

Sort being executed.

The data sorted in descending order is output, together with ranking, item name, subtotal and percentage.

Menu display

If the printer is connected, enter Y ↵

When sorting data in item Y, enter Y ↵ .

The data sorted in descending order is output.

(4) Data output

If the printer is connected, enter Y ↵

Step	Key operation	Display
14	3 ↵	PRINTER ON ? (Y/N) _
15	N ↵	SORT X OR Y ? _
16	X ↵	SORTING NOW
17		1 X=2 1378 37% 2 X=1 1322 35% 3 X=3 1059 28% GRAND T. 3759
18		1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ? _
19	3 ↵	PRINTER ON ? (Y/N) _
20	N ↵	SORT X OR Y
21	Y ↵	SORTING NOW
22		1 Y=3 2283 61% 2 Y=1 1047 28% 3 Y=2 429 11% GRAND T. 3759
23		1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ? _
24	4 ↵	PRINTER ON ? (Y/N) _
25	N ↵ ↵ ↵ ↵	X=1 Y=1 DATA=321 X=1 Y=1 DATA=369 X=1 Y=1 DATA=357 X=1 Y=2 DATA=159 ⋮

* The sum of percentages is not always 100% depending on data values.

GRAPH MAKING PROGRAM

This program draws various types of graphs with the plotter-printer (FA-10 or FA-11). Up to 12 data items can be input. The range of data is as follows.

$$|\text{Value of data}| \leqq 1E90$$

The program can draw beautiful band, bar, and line graphs, taking advantage of the 4-color plotter-printer.

* This program is stored on the cassette tape which comes with the FA-11 optional plotter-printer. It is also stored on the microcassette tape which comes with the CM-1 optional microcassette tape recorder.

Explanation

When the program is executed, the menu is displayed. First, data must be entered. This can be done by pressing [1].

The range of data is shown above. Data may be negative numerical data. Up to 12 data items can be entered. After the 12th data item is entered, the menu display automatically appears. To terminate the input in the middle, press the [↵] key without entering any numerical data.

Menu 2 is used to correct input data. Pressing [2] causes the first input data to be displayed, and pressing the [↵] key causes the next data to appear. Pressing the [SHIFT] and [↵] keys causes the previous data to appear. Input the correct data when data to be corrected appears.

Menu 3 is the routine to make graphs. Three kinds of graph names are displayed by pressing [3]. Select the type of graph by pressing [1], [2], or [3]. (Pressing [4] causes return to the menu display.)

Type 1 is a band graph. The entire length of band represents 100%, with the percentage of each data element represented by the length it occupies. For easy recognition, the individual data elements are shown in different colors and stripes. When the band graph is selected, negative data causes the menu to be displayed.

Type 2 is a bar graph. The scale is automatically set according to the size of input data. In this graph, positive values are output in green and negative values are output in red.

Type 3 is a line graph. When [3] is pressed, "Over previous graph?" is displayed. If a bar graph has been drawn just before, a line graph can be overwritten on the bar graph by pressing [Y]. If no bar graph has been drawn or a line graph should not be overwritten on a bar graph, press [N]. In this case, the appropriate scale is automatically set and the line graph is drawn.

Regardless of the type selected, the menu display appears after the graph is drawn. The menu display also appears when no data is entered. Note that starting the program again clears the existing data.

Menu 4 terminates program execution.

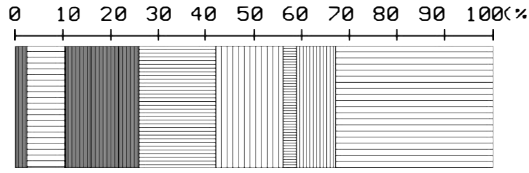
Menu 5 is provided to output the data to the plotter-printer. It can also be used to output the total of data.

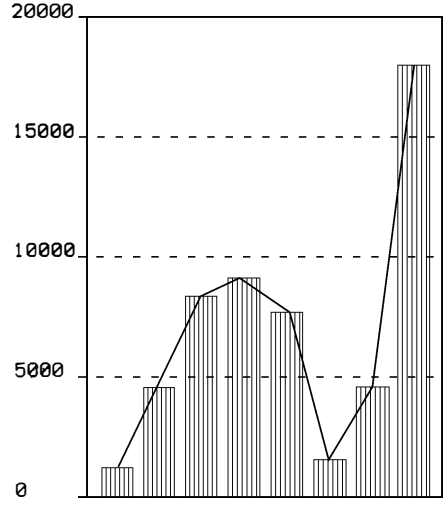
Print-out Example

```
Print Data

D(1)= 1200
D(2)= 4500
D(3)= 8383
D(4)= 9102
D(5)= 7701
D(6)= 1532
D(7)= 4562
D(8)= 18020

Total= 55000
```





Program

```
P0
  10 CLEAR
  20  CLS :PRINT "    ---- DATA ----    "
  30 PRINT TAB(2);"1:Input 2:Correct",T
     AB(2);"3:Graph 4:END",TAB(2);"5:Pr
     int Data";
  40 K=VAL(INKEY$):IF K<1 THEN 40 ELSE
     IF K>5 THEN 40
```

```
 50 BEEP :GOTO K*100
100  CLS :ERASE D:DIM D(13):Z=1
110 LOCATE 2,3:PRINT "<RETURN> : END";
120 LOCATE 6,0:PRINT "                ";
130 LOCATE 0,0:PRINT "D(";Z;")=";:INPU
    T "",AB$
140 IF AB$<>"" THEN D(Z)=VAL(AB$):IF Z
    <12 THEN Z=Z+1:GOTO 120 ELSE 20 EL
    SE Z=Z-1:GOTO 20
200 IF Z<1 THEN 20
210  CLS :I=1:PRINT TAB(46);":Shift RE
    TURN",TAB(6);":RETURN ";
220 LOCATE 4,2:PRINT CHR$(228);CHR$(22
    9):LOCATE 4,3:PRINT CHR$(230);CHR$
    (231);
230 DRAWC(32,23)-(47,23)-(47,24)-(32,2
    4):LOCATE 0,0:PRINT "D(";
240 LOCATE 2,0:PRINT I;")=";D(I);"
                ":LOCATE 6,1:PRINT "
                "
250 LOCATE 6,1:INPUT AB$:K$=INKEY$:IF
    AB$<>"" THEN D(I)=VAL(AB$)
260 IF K$=CHR$(24) THEN I=I-1:IF I<1 T
    HEN 20 ELSE 240
270 IF K$=CHR$(13) THEN I=I+1:IF I>Z T
    HEN 20 ELSE 240
280 IF K$=CHR$(23) THEN I=I+1:IF I>Z T
    HEN 20 ELSE 240
290 IF K$="" THEN I=I+1:IF I>Z THEN 20
     ELSE 240
300 IF Z<1 THEN 20 ELSE  CLS
310 PRINT TAB(5);"1:Band",TAB(5);"2:Ba
    r"
```

```
 320 PRINT TAB(5);"3:Line",TAB(5);"4:ME
     NU";
 330 K=VAL(INKEY$):IF K<1 THEN 330 ELSE
      IF K>4 THEN 330
 340 BEEP :GOTO K*1000
 400 LPRINT CHR$(28);CHR$(46):END
 500 IF Z<1 THEN 20 ELSE GOSUB 6000:LPR
     INT "Q1":V=0
 510 LPRINT "M93,0","PPrint Data"
 520 FOR I=1 TO Z
 530 LPRINT "M";90-4*I;",0","PD(";MID$(
     STR$(I),2);")=";D(I)
 540 V=V+D(I):NEXT I
 550 LPRINT "M";90-I*4-5;",0","PTotal="
     ;V:GOSUB 6000:GOTO 20
1000 GOSUB 6000:A=0
1010 FOR I=1 TO Z:IF D(I)<0 THEN ERASE
     I:GOTO 20 ELSE A=A+D(I):NEXT I
1020 IF A<=0 THEN 20
1030 LPRINT "O5,0","X1,8,10","M85,1","P
     (";CHR$(37);")"
1040 FOR I=100 TO 0 STEP -10:LPRINT "M"
     :-4+8*I/10;",1","P";I:NEXT I
1050 B=0:C=0
1060 FOR I=1 TO Z
1070 C=ROUND(D(I)/A*80+C,-2)
1080 LPRINT "J";I MOD 4,"A";B;",-3,";C;
     ",-23"
1090 LPRINT "G";I MOD 2+1;",";C-B;",-20
     ,";(I MOD 3)/4+.5
1100 B=C:NEXT I
1110 LPRINT "H30":GOTO 20
2000 GOSUB 6000:GOSUB 7000:GOSUB 8000
```

```
2010 FOR I=1 TO Z
2020 IF D(I)>=0 THEN J=2 ELSE J=3
2030 LPRINT "J";J
2040 LPRINT "A";0;",";6-8*I;",";ROUND(D
     (I)/A*90/N+0,-2);",";-8*I
2050 LPRINT "M";0;",";6-8*I,"G1,";ROUND
     (D(I)/A*90/N,-2);",-6"
2060 NEXT I
2070 LPRINT "M0,";-8*(Z+1):GOTO 20
3000 IF Z<2 THEN 20 ELSE CLS :PRINT "Ov
     er previous graph?",TAB(8);"Y/N"
3010 K$=INKEY$:IF K$="Y" THEN BEEP :GOT
     O 3080 ELSE IF K$<>"N" THEN 3010 E
     LSE BEEP
3020 GOSUB 6000
3030 GOSUB 7000:GOSUB 8000
3040 LPRINT "L1","J0"
3050 FOR I=1 TO Z
3060 F=3-8*I:IF I MOD 2=1 THEN LPRINT "
     D0,";F;",90,";F ELSE LPRINT "D90,"
     ;F;",0,";F
3070 NEXT I:LPRINT "L0","M0,0"
3080 S=S+1:IF S>3 THEN S=0
3090 T=T+.25:IF T>3.9 THEN T=0
3100 IF T<2 THEN LPRINT "B1.6" ELSE LPR
     INT "B6.4"
3110 LPRINT "J";S,"L";T
3120 G=ROUND(D(1)/A*90/N+0,-2):H=-5
3130 FOR I=2 TO Z
3140 U=ROUND(D(I)/A*90/N+0,-2):V=3-8*I
3150 LPRINT "D";G;",";H;",";U;",";V
3160 G=U:H=V
3170 NEXT I
```

```
3180 LPRINT "B3.2","M0,";-8*(Z+1):GOTO
     20
4000 GOTO 20
6000 LPRINT CHR$(28);CHR$(37),"O0,0","J
     0","L0","S1","Q0","Y0","B3.2","H20
     "
6010 S=0:T=0:RETURN
7000 Y=-9E99:B=9E99
7010 FOR I=1 TO Z
7020 IF D(I)>Y THEN Y=D(I)
7030 IF D(I)<B THEN B=D(I)
7040 NEXT I
7050 IF Y>=0 THEN D(0)=Y ELSE D(0)=0
7060 IF B>0 THEN D(Z+1)=0 ELSE D(Z+1)=B
7070 RETURN
8000 IF SGND(0)*SGND(Z+1)<=0 THEN M=ABS
     (D(0)-D(Z+1)) ELSE M=D(0):IF M<0 T
     HEN M=ABSD(Z+1)
8010 IF M=<0 THEN 20 ELSE R=INTLGTM:A=1
     0^R
8020 IF A*INT(M/A)*.75<M THEN A=A*.5
8030 D=LEN(STR$(A))*2.4+5
8040 IF SGND(0)*SGND(Z+1)<0 THEN N=INT(
     M/A)+2 ELSE N=INT(M/A)+1
8050 C=ABSINT(D(0)/A)+SGND(0)
8060 FOR I=N TO 0 STEP -1:IF C=0 THEN O
     =I*90/N
8070 C=C-1:NEXT I
8080 LPRINT "D0,0,90,0","Q1":W=18/N:V=0
8090 V=V+5:IF V*W<18 THEN 8090
8100 IF D(0)<=0 THEN 8150 ELSE X=O
8110 K=ROUND(X,-2):LPRINT "D";K;",2,";K
     ;",-2"
```

```
8120 LPRINT "M";K;",";D,"P";ROUND((X-O)
     *A*N/90,R-2)
8130 LPRINT "L1","D";K;",0,";K;",";-Z*8
     -2,"L0":X=X+W*U
8140 IF X<90 THEN 8110
8150 LPRINT "D90,2,90,-2","M90,";D,"P";
     ROUND((90-O)*A*N/90,R-2)
8160 IF D(Z+1)>=0 THEN 8220 ELSE X=O-W*
     U
8170 K=ROUND(X,-2):LPRINT "D";K;",2,";K
     ;",-2"
8180 LPRINT "M";K;",";D,"P";ROUND((X-O)
     *A*N/90,R-2)
8190 LPRINT "L1","D";K;",0,";K;",";-Z*8
     -2,"L0":X=X-W*U
8200 IF X>0 THEN 8170
8210 LPRINT "D0,2,0,-2","M0,";D,"P";ROU
     ND(-O*A*N/90,R-2)
8220 LPRINT "D0,0,0,";-8*Z-2;",90,";-8*
     Z-2;",90,0","M0,0"
8230 RETURN
```

Number of bytes used:2712

If one-key commands are used for the input of lines 30, 140, 240, 3010, 3060 and 8000, spaces are automatically entered after the commands and a whole line cannot be input because the input range of 79 characters is exceeded. Therefore, input each line after deleting extra spaces using the [DEL] key.

Example: 30␣PRINT␣TAB(2);....

⇩

30 PRINTTAB(2);....

Operation

	Step	Key operation	Display
Menu display →		SHIFT P0 0	---- DATA ---- 1 : Input 2 : Correct 3 : Graph 4 : END 5 : Print Data
(1) Data input	1	1	D(1) = <RETURN> : END
After the 12th data item is entered, the menu display automatically appears. →	2	2112 ↵ ⋮	D(2) =
(2) Data correction After the correction is made to the last, the menu is displayed. →	3	2 1200 ↵ ⋮	D(1) = 1200 ? ▲Shift RETURN (Previous data) ▼RETURN (Next data)
(3) Graph making	4	3	1 : Band 2 : Bar 3 : Line 4 : MENU
Band graph is printed out. →	5	1	(After the graph is output, the menu is displayed.)
Bar graph is printed out. →		2	(After the graph is output, the menu is displayed.)
Determine whether a line graph is over written on the bar graph or not. →		3	Over previous graph? Y/N
Line graph is printed out. →		Y or N	(After the graph is output, the menu is displayed.)
(5) Data output. Data are printed out. →	6	5	(After the data are output, the menu is displayed.)
(4) Execution termination.	7	4	

CHAPTER 6

REFERENCE MATERIAL

6-1 PB-770 COMMAND TABLE

6-1-1 Operational Symbols

1 Arithmetic operators

Name	General format	PB-770 format	Meaning	Priority
Power	x^y	X ^ Y	Raise X to power Y.	1
Multiplication	$x \times y$	X*Y	Multiply X by Y.	2
Division	$x \div y$	X/Y	Divide X by Y.	2
Remainder	$x \div y = z$	X MOD Y	Remainder when X is divided by Y.	3
Addition	$x + y$	X + Y	Add Y to X.	4
Subtraction	$x - y$	X − Y	Subtract Y from X.	4
Assignment	$x = y + 5$	X = Y + 5	Assign Y + 5 to X.	5

2 Relational operators (conditional expressions)

General format	PB-770 format	Meaning
$x = y$	X = Y	X is equal to Y.
$x \neq y$	X<>Y, X><Y	X is not equal to Y.
$x < y$	X < Y	X is smaller than Y.
$x > y$	X > Y	X is greater than Y.
$x \leqq y$	X<=Y, X=<Y	X is smaller than or equal to Y.
$x \geqq y$	X>=Y, X=>Y	X is greater than or equal to Y.

- The relational operators are valid only in IF statements.
- A comparison can be made between numerical constants, numerical variables, and numerical expressions, and between character constants and character variables.

3 Character expression operators

+ ····· Two or more character strings can be concatenated by a + (plus sign).

6-1-2 Special Character

General format	PB-770 format	Meaning
$x \times 10^y$	X E Y	Exponent entry (Multiply number X by Y power of 10.)

- If the absolute value of the operation result is equal to or greater than 10^{10} or smaller than 10^{-3} (0.001), it is automatically indicated by exponential notation.

6-1-3 Built-in Functions

Name	General format	PB-770 format	Meaning / Remarks	Page
Trigonometric	$\sin x$	SIN X	Gives the sine of X.	214
	$\cos x$	COS X	Gives the cosine of X.	217
	$\tan x$	TAN X	Gives the tangent of X.	218
Inverse trigonometric	$\sin^{-1} x$	ASN X	Gives the arcsine of X.	219
	$\cos^{-1} x$	ACS X	Gives the arccosine of X.	
	$\tan^{-1} x$	ATN X	Gives the arctangent of X.	
Hyperbolic	$\sinh x$	HYPSINX	Gives sinh X.	221
	$\cosh x$	HYPCOS X	Gives cosh X.	
	$\tanh x$	HYPTAN X	Gives tanh X.	
Inverse hyperbolic	$\sinh^{-1} x$	HYPASN X	Gives $\sinh^{-1} x$.	221
	$\cosh^{-1} x$	HYPACS X	Gives $\cosh^{-1} x$.	
	$\tanh^{-1} x$	HYPATN X	Gives $\tanh^{-1} x$.	
Logarithmic	$\log x$	LGT X	$\log_{10}$ X (common logarithm)	223
	$\ln x$	LOG X	$\log_e$ X (natural logarithm)	
Exponential	e^x	EXP X	X power of natural logarithm base (e)	226
Power	x^y	X ^ Y	Y power of X.	26
Square root	$\sqrt{x}$	SQR X	Gives the square root.	222
Absolute value	$\|x\|$	ABS X	Gives the absolute value of X.	228
Integer		INT X	When X>0, the fraction portion of X is discarded. When X<0, the fraction portion of |X| is rounded up, a − (minus sign) is prefixed to |X|. INT 1.2 → 1 INT −1.2 → −2	230
Fraction		FRAC X	Eliminates the integer portion to obtain only the fraction portion.	232
Circular constant	π	PI	Gives an approximate ratio of the circumference of a circle to its diameter in 11 digits: 3.1415926536.	238
Random number		RND	Generates a 10-digit pseudo random number (0 < RND < 1).	239

Name	General format	PB-770 format	Meaning/remarks	Page
Sign		SGN X	Checks the sign of an argument: X < 0 → −1 X = 0 → 0 X > 0 → 1	**234**
Rounding		ROUND (X, Y)	Rounds off the value of X at 10^{Y} positions.	**236**
Degree, minute, second	Sexagesimal →decimal	DEG (d[,m[,s]]) (d, m, s: numerical expressions)	Gives the decimal equivalent of a hexadecimal value.	**241**
Memory contents reading		PEEK X	Gives the contents of address X	**242**

6-1-4 Character Functions

Use	Function	Example	Meaning	Page
Gives character code of first character of a string.	ASC	PRINT ASC ("E")	Displays the character code of character E.	**243**
Gives one character designated by character code.	CHR $	PRINT CHR $ (69)	Displays character (E) equivalent to character code 69.	**245**
Converts numeral in a character string to numerical value.	VAL	A = VAL (X$)	Converts a character string of numerals stored in character variable X$ to a numerical value.	**247**
Converts numerical value to character string.	STR $	C$ = STR $ (X)	Converts a numerical value stored in numerical variable X to a character string.	**250**
Fetches specified number of characters from left of character string.	LEFT $	C$ = LEFT $ (X$, 3)	Fetches the three characters on the left of character string stored in X$ and assigns them to C$.	**252**

Use	Function	Example	Meaning	Page
Fetches specified number of characters from right of character string.	RIGHT $	C$ = RIGHT $ (X$, 3)	Fetches the three characters on the right of character string stored in X$ and assigns them to C$.	**253**
Fetches specified number of characters starting from the specified position.	MID $	C$ = MID $ (X$, 3, 5)	Fetches the five characters starting from the third character of character string stored in X$ and assigns them to C$.	**254**
Counts the number of characters in a character string.	LEN	A = LEN (X$)	Assigns the number of characters in character string stored in X$ to A.	**256**
Inputs one character from the keyboard.	INKEY $	A$ = INKEY $	When INKEY$ is executed, if one key on the keyboard is pressed, it is assigned to A$. Only one character can be assigned to A$.	**257**
Converts a decimal value to sexagesimal.	DMS$	C$=DMS$(X)	Converts the numerical value assigned to X to a character string that represents the sexagesimal value of X.	**259**
Converts a decimal value to hexadecimal.	HEX$	C$=HEX$(X)	Converts the numerical value assigned to X to a character string that represents the hexadecimal value of X.	**260**

6-1-5 Display Functions

Use	Function	Example	Meaning	Page
Checks whether a dot on the screen is on or off.	POINT	POINT (10, 20)	Checks whether the dot represented by coordinates (10, 20) is on (displays 1) or off (displays 0)	**266**
Moves cursor by specified number of positions.	TAB	PRINT TAB(10)	Tabs cursor to position 10 on the screen.	**261**
Specifies output format.	USING	PRINT USING "###.##"; A	Displays a numerical value stored in numerical variable A according to format "###.##"	**263**

6-1-6 Statistical Functions

Function	General format	Meaning	Page
CNT	n	Number of statistical data processed.	**270**
SUMX	Σx	Sum of X data.	**271**
SUMY	Σy	Sum of Y data.	
SUMX2	Σx^2	Sum of squares of X data.	
SUMY2	Σy^2	Sum of squares of Y data.	
SUMXY	Σxy	Sum of products of X data and Y data.	
MEANX	$\overline{x}$	Mean of X data.	**272**
MEANY	$\overline{y}$	Mean of Y data.	
SDX	$x\sigma_{n-1}$	Sample standard deviation of X data.	**273**
SDY	$y\sigma_{n-1}$	Sample standard deviation of Y data.	
SDXN	$x\sigma_n$	Population standard deviation of X data.	
SDYN	$y\sigma_n$	Population standard deviation of Y data.	
LRA	a	Linear regression constant term.	**274**
LRB	b	Linear regression coefficient.	
COR	r	Correlation coefficient.	**270**
EOX	$\hat{x}$	Estimated value of X for Y.	**274**
EOY	$\hat{y}$	Estimated value of Y for X.	

6-1-7 Manual Commands

Use	Command	Example	Meaning	Page
Automatically generates line numbers.	AUTO	AUTO	Line numbers starting with line 10 and incremented by 10.	130
		AUTO 100	Line numbers starting with line 100 and incremented by 10.	
		AUTO 50, 20	Line numbers starting with line 50 and incremented by 20.	
Resumes program execution.	CONT	CONT	Resumes the execution of a program that has been stopped by a STOP statement or by the STOP (ANS) key.	131
Deletes program.	DELETE	DELETE 50	Deletes line 50.	132
		DELETE 30–	Deletes from line 30 to the end.	
		DELETE –100	Deletes up to line 100 from the beginning.	
		DELETE 150 – 200	Deletes from line 150 to line 200.	
Modifies program.	EDIT	EDIT	Displays the first line and specifies the EDIT mode.	134
		EDIT 30	Displays line 30 and specifies the EDIT mode.	
Displays program list.	LIST	LIST	Displays the program in the presently specified program area.	137
		LIST 50	Displays line 50.	
		LIST 30 –	Displays from line 30 to the end.	
		LIST – 50	Displays up to line 50.	
		LIST 30 – 50	Displays from line 30 to line 50.	
		LIST ALL	Displays all programs in the entire program area.	
		LIST V	Displays registered variable names.	

Use	Command	Example	Meaning	Page
Prints program list.	LLIST	LLIST	Prints the program in the presently specified program area.	137
		LLIST 50	Prints line 50.	
		LLIST 30 –	Prints from line 30 to the end.	
		LLIST – 50	Prints from the beginning to line 50.	
		LLIST 30 – 50	Prints from line 30 to line 50.	
		LLIST ALL	Prints all programs in all program areas.	
		LLIST V	Prints registered variable names.	
Reads program from cassette tape.	LOAD	LOAD	Reads a program in internal code format to the presently specified program area.	139
		LOAD ALL	Reads all programs in internal code format to the all program areas.	
		LOAD, A	Reads a program in ASCII code format to the presently specified program area.	
		LOAD, M	Links the program in the presently specified program area with the program read in ASCII code format.	
		LOAD, D, 4096	Reads internal code format data from memory address 4096(10).	
		LOAD "ABC"	Perform functions similar to the above respectively in regard to programs with file names.	
		LOAD ALL "CASIO"		
		LOAD "TEST", A		
		LOAD "TEST", M		
		LOAD "PB", D, 4096		
Erases program.	NEW	NEW	Erases a program in the presently specified program area.	143
		NEW ALL	Clears the entire RAM area.	

Use	Command	Example	Meaning	Page
Protects program.	PASS	PASS "KEY"	Sets a password named "KEY."	**144**
Specifies a program area.	PROG	PROG 2	Specifies the program area 2.	**146**
Starts program execution.	RUN	RUN	Starts the execution of a program from the beginning of the presently specified program area.	**147**
		RUN 100	Starts the execution of a program from line 100.	
Stores programs to a cassette tape.	SAVE	SAVE	Stores the program in the presently specified program area on a cassette tape in the internal code format.	**148**
		SAVE ALL	Stores all programs in all program areas on a cassette tape in the internal code format.	
		SAVE, A	Stores the program in the presently specified program area on a cassette tape in the ASCII code format.	
		SAVE, D, 3000, 3999	Stores internal code format data on a cassette tape from address 3000 (10) to 3999(10). * (10) indicates decimal values as opposed to hexadecimal values.	
		SAVE "ABC"	Perform functions similar to the above respectively in regard to programs with file names.	
		SAVE ALL "CASIO"		
		SAVE "TEST", A		
		SAVE "PB", D, 3000, 3999		
Displays status of program areas.	SYSTEM	SYSTEM	Displays program area status, ANGLE setting, memory capacity, remaining number of bytes, and data area starting address.	**151**
Checks programs stored on a cassette tape.	VERIFY	VERIFY	Performs a parity check of the program file which appears first.	**153**
		VERIFY "ABC"	Performs a parity check of the program with a specified file name.	

6-1-8 Program Commands

Use	Command	Example	Meaning	Page
Specifies angle unit.	ANGLE	ANGLE 0	Specifies degrees.	**154**
		ANGLE 1	Specifies radians.	
		ANGLE 2	Specifies grads.	
Generates buzzer sound.	BEEP	BEEP	Same as BEEP 0.	**155**
		BEEP 0	Generates a low pitched beep.	
		BEEP 1	Generates a high pitched beep.	
Reads and executes program.	CHAIN	CHAIN	Loads and executes the PF B that first appears.	**156**
		CHAIN "XYZ"	Loads and executes the program with a specified file name.	
Clears all variables. Specifies the starting address of the data area.	CLEAR	CLEAR	Clears all variables.	**158**
		CLEAR 4000	Clears all variables and sets up a data area from address 4000(10).	
Clears display screen.	CLS	CLS	Clears the entire screen and moves the cursor to the home position.	**161**
Stores data.	DATA	DATA 1,2,3	Stores data to be referenced by the READ statement.	**205**
Declares array.	DIM	DIM A (3)	Declares a one-dimensional single-precision numerical array.	**162**
		DIM B (2, 3)	Declares a two-dimensional single-precision numerical array.	
		DIM C! (4)	Declares a one-dimensional half-precision numerical array.	
		DIM D! (3, 4)	Declares a two-dimensional half-precision numerical array.	
		DIM E$ (5)	Declares a one-dimensional fixed-length character array.	
		DIM F$ (4,5)	Declares a two-dimensional fixed-length character array.	
		DIM G$ (2)*3	Declares a one-dimensional defined-length character array and specifies 3 as the character length.	
		DIM H$ (4, 5)*6	Declares a two-dimensional defined-length character array and specifies 6 as the character length.	
Draws point and straight line.	DRAW	DRAW (0, 0)	Draws a point at coordinates (0, 0).	**167**
		DRAW (1, 0)–(5,10)	Draws a straight line from coordinates (1, 0) to (5, 10).	

Use	Command	Example	Meaning	Page
Erases point and straight line.	DRAWC	DRAWC (0,0)	Erases the point at coordinates (0, 0).	167
		DRAWC (1,0)–(5,10)	Erases the straight line between coordinates (1, 0) and (5, 10).	
Terminates program execution.	END	END	Terminates the execution of a program.	170
Releases array name.	ERASE	ERASE A	Releases the definition of registered variable A or array variable A.	171
Loop (repeat).	FOR TO STEP ≀ NEXT	FOR I=5 TO 20 STEP 0.5 ≀ NEXT I	Repeatedly performs the processing between FOR and NEXT while incrementing the value of variable I by 0.5 from 5 to 20.	172
Reads variable data from cassette tape.	GET	GET A	Reads the variable data that first appears.	177
		GET "MAX" B	Reads the variable data having a file name "MAX."	
Jumps to subroutine.	GOSUB	GOSUB 100	Jumps to the subroutine in line 100.	180
		GOSUB PROG 3	Jumps to the subroutine in program area 3.	
End of subroutine.	RETURN	RETURN	Returns to the command following the GOSUB statement.	180
Unconditional jump.	GOTO	GOTO 500	Jumps to line 500.	184
		GOTO PROG 5	Jumps to program area 5.	
Conditional jump.	IF ~ THEN ~ ELSE ~	IF I > 9 THEN 50 ELSE 80	Jumps to line 50 if I is greater than 9; otherwise, jumps to line 80.	186
Data input from keyboard.	INPUT	INPUT S	Displays a ?, then waits for data to be entered in S.	189
		INPUT "NAME", T$	Displays NAME, then waits for data to be entered in T$.	
		INPUT "NAME"; U$	Displays NAME ?, then waits for data to be entered in U$.	
Assigns data to variable.	LET	LET A=B	Assigns B to A.	195
Specifies cursor position.	LOCATE	LOCATE 2, 3	Specifies coordinates (2, 3) as the cursor position.	196

Use	Command	Example	Meaning	Page
Writes data to a memory address.	POKE	POKE 3500, 10	Writes the value 10(10) to address 3500(10).	197
Displays data.	PRINT	PRINT C, D	Displays the values of C and D on separate lines.	198
		PRINT C; D	Displays the values of C and D on the same line.	
		PRINT AB	Displays the pattern defined by AB$.	
Prints data.	LPRINT	LPRINT C, D	Prints the values of C and D on separate lines.	198
		LPRINT C; D	Prints the values of C and D on the same line.	
Stores variable data on cassette tape.	PUT	PUT A	Stores data of variable A on cassette tape.	203
		PUT "DATA" A	Stores data of variable A with file name on cassette tape.	
Reads stored data.	READ	READ X	Reads data stored in variable X by DATA statement.	205
Remark	REM	REM ***	Provides a comment in a program.	209
Specifies sequence of execution of DATA statements.	RESTORE	RESTORE	Reads from the first DATA statement when executing READ statement.	205
		RESTORE 100	Reads from the DATA statement on line 100 when executing READ statement.	
Inputs statistical data.	STAT	STAT 1, 2; 2	Inputs values of x data, y data and frequency.	268
Clears the statistical registers.	STAT CLEAR	STAT CLEAR	Sets contents of CNT, SUMX, SUMY, SUMX2, SUMY2 and SUMXY to 0.	269
Displays the statistical register contents.	STAT LIST	STAT LIST	Displays contents of CNT, SUMX, SUMY, SUMX2, SUMY2 and SUMXY.	269
Prints the statistical register contents	STAT LLIST	STAT LLIST	Outputs contents of CNT, SUMX, SUMY, SUMX2, SUMY2 and SUMXY to printer.	269
Halts program execution.	STOP	STOP	Suspends the execution of a program.	210
Traces program execution.	TRON	TRON	Specifies the trace mode and traces the status of program execution.	212
Releases tracing of program execution.	TROFF	TROFF	Releases trace mode.	212

6-2 ERROR MESSAGE TABLE

Message	Meaning	Countermeasure
BS error (Bad Subscript)	• Subscript of an array variable is a negative value or the value exceeds 255. **Example)** DIM A (256) • Specified numerical value is outside the argument range. **Example)** The POINT function has the following argument ranges: $0 \leqq X \leqq 159$ and $0 \leqq Y \leqq 31$ But the specified numerical value is outside this range.	• Change the value of subscript to a value within the specified range. When the subscript is a variable, check the assigned value. • Re-specify the subscript within the specified argument range.
BV error (Buffer oVerflow)	• An overflow of the input or output buffer.	• Each operation or program statement cannot exceed 79 characters in length.
DA error (DAta error)	• A READ statement or GET statement was executed even though there is no data to read.	• Check the relation between the READ and DATA statements. Ensure that there is data for each READ.
DD error (Duplicate Definition)	• Arrays having the same array name and a different subscript were doubly defined. **Example)** The following array variable declarations by a DIM statement cause a DD error. .... DIM A(1), A(2, 3) DIM A! (1), A!(2, 3) DIM A$(1), A$(2, 3) DIM A$(1)*20, A$(2, 3)*20 DIM A$(1)*20, A$(2, 3)*20	• Check the variable on the line in which an error has occurred. Also check for subscripts having the same array name. Change either of the array names and reorganize the program. • Input CLEAR or ERASE before the DIM statement or execute it manually to clear the array.
FC error (illegal Function Call)	• An attempt was made to execute any of the following manual commands in a program: CONT, PASS, RUN, EDIT, DELETE • An attempt was made to execute any of the following program commands manually: END, LET, REM, STOP, LOCATE, DRAW, DRAWC, GOTO, GOSUB, RETURN, INPUT, DATA, READ, RESTORE, FOR~NEXT, IF~THEN~ELSE~ • A CONT command was used when program execution could not be resumed.	• Remove the incorrect command from the program. • Execute the command with a line number attached. • Press the BRK key. Then either re-execute the program from the beginning or, if the stopped line is known, re-execute the program from the line next to the stopped line by "RUN line number".

Message	Meaning	Countermeasure
FO error (NEXT without FOr)	● FOR statement without corresponding NEXT statement.	● Check the nesting structure.
GS error (RETURN without GoSub)	● RETURN statement without corresponding GOSUB statement.	● Check the nesting structure in the GOSUB statement, and clearly distinguish between the main routine and the subroutine.
MA error (MAthematical error)	● An arithmetic operation involving numerical values or numerical functions is uncertain or impossible. **Example)** Division by 0.	● Check the numerical expression on the line in which an error has occurred. Also check the value of the any variables.
NO error (Nesting Overflow)	● The number of nesting levels exceeded the specified limit. **Example)** GOSUB~RETURN Max. 12 levels FOR~NEXT Max. 6 levels	● Check the nesting structure and reduce the nesting level within the allowable range.
NR error (device Not Ready)	● I/O device is not correctly connected. **Example)** No magnetic tape recorder is connected.	● Check that the relevant device is properly connected and switched on.
OM error (Out of Memory)	● RAM capacity is insufficient. ● Insufficient number of bytes required for variable or PRINT (LPRINT) in the work area. **Example)** 5 bytes remains. Q$="12345" PRINT "12345"	● Delete unnecessary programs. Decrease the size of the data area using the CLEAR command. Expand memory capacity using RAM expansion pack(s). Confirm the number of remaining bytes using the SYSTEM command.
	● Bad specified address location by the CLEAR statement.	● Use a higher address in the CLEAR command and decrease the size of the data area.
OV error (OVerflow error)	● The result of an operation or the numerical value entered exceeds 10^{100}.	● Check the numerical expression in the line in which error occurred. Insert a PRINT statement in the program to check the value of the variable.

Message	Meaning	Countermeasure
PR error (PRotected error)	● An attempt was made to execute any of the following commands which cannot be used with a program having a password: DELETE, LIST, LLIST, NEW, EDIT ● An attempt was made to add a new line to, or delete a line from, a program having a password.	● Re-enter the password, release the locked condition, and execute the program.
	● A different password was entered.	● Input the correct password.
	● An attempt was made to load a program whose password is different from the PB-770 password.	● Release the PB-770 password before loading. In this case, the newly loaded password becomes the PB-770 password.
RW error (Read Write error)	● Parity error during execution of a LOAD or VERIFY command. ● Printer function is not activated.	● Store the program again using the SAVE command. ● Activate the printer function.
SN error (SyNtax error)	● Command format error.	● Use the EDIT command to call the line where the error has occurred and correct the line.
	● The line number has a fraction.	● Correct the line number.
	● An array having three or more dimensions was declared.	● The number of dimensions must not exceed 2.
SO error (Stack Over error)	● Numerical value stack exceeds 8 levels. ● Operator stack exceeds 20 levels.	● Simplify or divide the numerical expression so that the stack level can fall within the specified range.
	● Character stack exceeds 10 levels.	● Simplify or divide the character expression so that the stack level can fall within the specified range.
ST error (STring error)	● An attempt was made to assign a character string whose length exceeds the allowable character variable length. The allowable character variable length is as follows: – Fixed character variable Max. 7 characters – Registered character variable Max. 16 characters – Fixed-length character array variable Max. 16 characters – Defined-length character array variable Max. 79 characters	● Change the variable to another variable that can contain more characters. Decrease the number of characters of the character string to be assigned to the variable. Be careful when concatenating character strings.

Message	Meaning	Countermeasure
TM error (Type Missmatch)	● In an assignment statement, the left and right sides are of different variable types. ● The argument types do not match during an assignment.	● Both the left and right sides of an assignment statement must be either numerical variables or character variables. ● Assign a numerical value to a numerical variable, and a character string to a character variable.
UL error (Undefined Line number)	● The specified line number does not exist in an IF~THEN, GOTO or GOSUB statement. ● No program exists in the program area specified by a GOTO or GOSUB statement.	● Create a line number to which a jump is to be made or change the specified line number to which a jump is to be made. ● Create a program area to which a jump is to be made, or change the location to which a jump is to be made.
UV error (Undefined Variable)	● Undefined variable used. ● Array variable used without declaring it using DIM statement. ● Subscript of an array variable exceeds the range specified by DIM.	● Check the initial value of variable. ● Declare the array by the DIM statement at the beginning of the program. ● Change the size of the subscript within the specified range.
VA error (VAriable error)	● An attempt was made to register more than 40 variables.	● Up to 40 registered and array variables can be used. Check the variable names by LISTV, and delete unnecessary variable names by CLEAR or ERASE.

6-3 CHARACTER CODE TABLE

Hexa-decimal	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	0	16	[SPACE] 32	48	64	80	96	112	128	144	[SPACE] 160	176	192	208	224	240
x1	1	[DEL] 17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
x2	[LINE TOP] 2	[INS] 18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
x3	3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
x4	4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
x5	5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
x6	[LINE END] 6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
x7	[BEL] 7	[ENTER] 23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
x8	8	[▲] 24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
x9	[TAB] 9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
xA	10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
xB	[HOME] 11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
xC	[CLS] 12	[CSR right] 28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
xD	[CR] 13	[CSR left] 29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
xE	14	[CSR up] 30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
xF	15	[CSR down] 31	47	63	79	95	111	[NULL] 127	143	159	175	191	207	223	239	[NULL] 255

SPECIFICATIONS

■ Type

PB-770

■ Fundamental calculation functions

Negative numbers, exponentials, parenthetical addition/subtraction/multiplication/division (with priority sequence judgement function – true algebraic logic), MOD .

■ Commands

AUTO, CONT, DELETE, EDIT, LIST, LLIST, LOAD, NEW, NEW ALL, PASS, PROG, RUN, SAVE, SYSTEM, VERIFY, ANGLE, BEEP, CHAIN, CLEAR, CLS, DIM, DRAW, DRAWC, END, ERASE, FOR-TO-STEP, NEXT, GET, GOSUB, RETURN, GOTO, IF-THEN-ELSE, INPUT, LET, LOCATE, POKE, PRINT, LPRINT, PUT, READ, DATA, RESTORE, REM, STOP, TRON, TROFF.
Statistical commands – STAT, STAT CLEAR, STAT LIST, STAT LLIST.

■ Functions

SIN, COS, TAN, ASN, ACS, ATN, HYPSIN, HYPCOS, HYPTAN, HYPASN, HYPACS, HYPATN, SQR, LOG, LGT, EXP, ABS, INT, FRAC, SGN, ROUND, PI, RND, DEG, PEEK, ASC, CHR$, VAL, STR$, LEFT$, RIGHT$, MID$, LEN, INKEY$, DMS$, HEX$, TAB, USING, POINT, &H.
Statistical functions – CNT, COR, SUMX, SUMY, SUMX2, SUMY2, SUMXY, MEANX, MEANY, SDX, SDY, SDXN, SDYN, EOX, EOY, LRA, LRB.

■ Calculation range

$\pm 1 \times 10^{-99} \sim \pm 9.999999999 \times 10^{99}$
(Internal calculation uses 12-digit mantissa.)

■ Program language

BASIC

■ Memory capacity for programs

RAM: Standard 8K bytes, expandable up to 32K bytes.
(Including 1321 bytes in system area.)
ROM: Approx. 32K bytes.

■ Number of program areas

Maximum 10 (P0 – P9)

■ Number of stacks

Subroutine	12 levels
FOR-NEXT loop	6 levels
Numerical values	8 levels
Operators	20 levels

■ **Display system**
Liquid crystal display (20 digits x 4 lines)

■ **Display elements**
32 x 160 dots (20 x 4 characters)

■ **Display contents**
10-digit mantissa + 2-digit exponent

■ **Main component**
LSI

■ **Power consumption**
0.1W

■ **Power source**
Main: 4 AA size batteries.
Sub (for RAM backup): 1 lithium battery (CR1220).

■ **Battery life**
Main: Approximately 100 hours on type SUM-3 (continuous operation).
Sub: (see page 15)

■ **Auto power off**
Power is automatically turned off approximately 6 minutes after last operation.

■ **Ambient temperature range**
0°C to 40°C (32°F to 104°F)

■ **Dimensions**
23mmH x 200mmW x 88mmD ($\frac{7}{8}$"H x 7$\frac{7}{8}$"W x 3$\frac{1}{2}$"D)

■ **Weight**
315 g (11.1 oz) including batteries.

INDEX

GUIDELINES LAID DOWN BY FCC RULES FOR USE OF THE UNIT IN THE U.S.A. (not applicable to other areas).

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.

Epilogue

This manual overhaul is dedicated to Alex Istomin, aka HWR0, you were a giant, my friend.

I decided to scan in this manual because there wasn't one available Online and I thought I might be able to give something back to the Pocket Computer community that has given me so much over the years. This was a labour in love, completed between March 15 to July 27, 2024. It was scanned in page by page using a Brother DCP-135C all-in-one printer/scanner (Thank you VueScan for still supporting my ancient scanner!), edited in GIMP and OCR-ed with Acrobat Pro. I completely re-drew all tables and about 90% of the graphics from scratch, corrected obvious errors (spelling, code and others). I added a page on character coding for Kana mode PB-770s (Japanese models, or modded European ones). The TOC, index and hopefully all keyword references have been linked. I'll upload two versions, a colour scan (cover only) and a Kindle/bandwidth friendly monochrome version. If you find any huge errors that might have slipped in during editing, please let me know. Most of you know how to get in touch with me.

-R. Swartz 2024

CASIO

075C GI ㊞ Printed in Japan