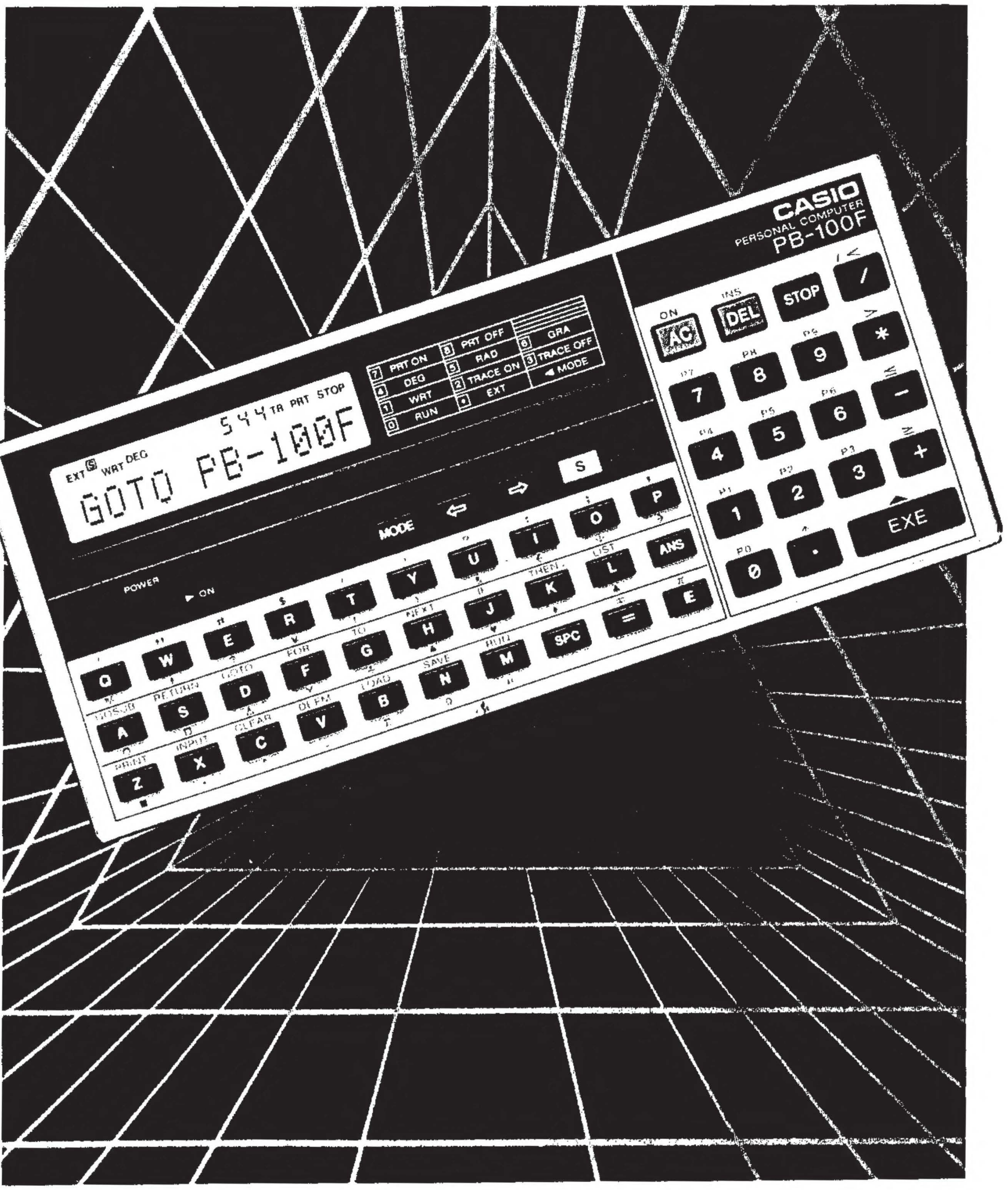
Personal Computer

PB-100F

OWNER'S MANUAL



CASIO®

CONTENTS

CHAPTER 6 REFERENCE MATERIAL

Personal Computer

PB-100F

CASIO®

CHAPTER 1

GENERAL GUIDE

Users who have not used a computer as well as those who are accustomed to a computer should read this chapter.

INTRODUCTION

This manual provides an explanation of the computer so that BASIC program beginners as well as users, who have a complete knowledge of BASIC and intend to fully utilize it, can easily understand and utilize the computer immediately.
Users who are new to BASIC programming should read this manual from Chapter 1 in order to master programming. Especially in Chapter 3, the explanation of program preparation and commands should be carefully read. A program flow explanation is provided in Chapter 3. See Chapter 4 "Command Reference" for the command formats and detailed explanation.
Users who have a knowledge of BASIC should utilize the computer while reading Chapter 4 "Command Reference" after mastering the basic operations explained in Chapters 1 and 2.
Users who intend to use programs immediately by entering them can utilize the programs in Chapter 5 "Program Library ".

PRIOR TO OPERATION

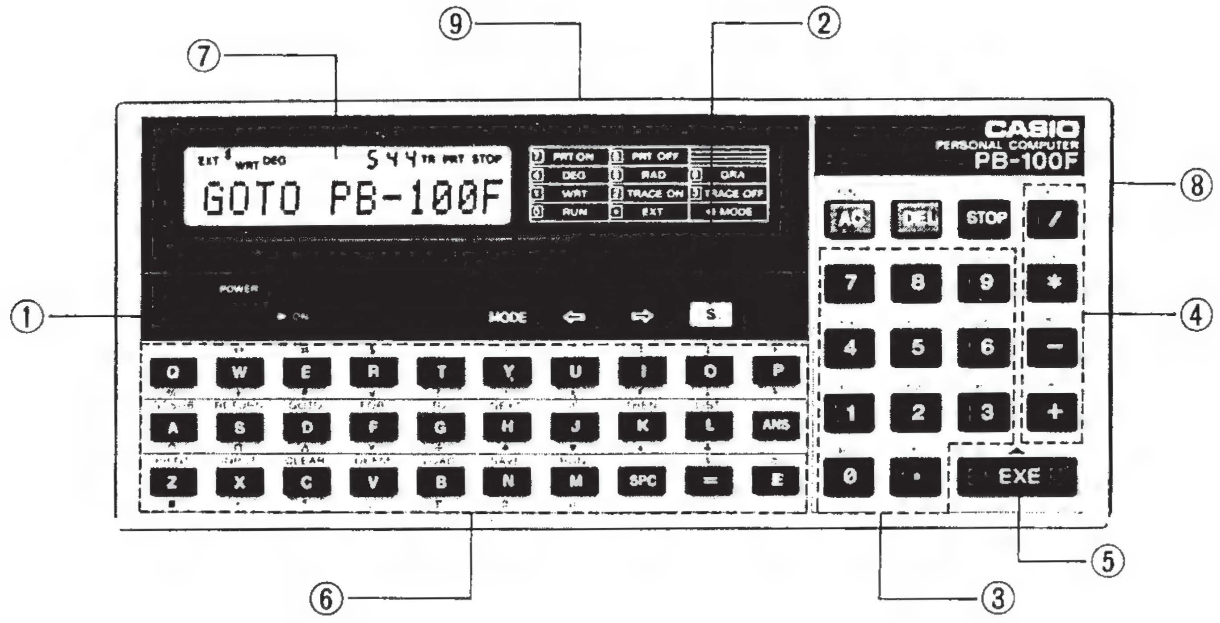
This computer was delivered to you through CASIO's strict testing process, high level electronics technology, and strict quality control.
To ensure a long life for your computer, please observe the following precautions.

■ Utilization precautions

- Since this computer consists of precision electronic parts, do not disassemble it. Also do not apply an impact to it by throwing or dropping it, or do not expose it to rapid temperature changes. In addition, do not store it in a place with high temperatures or high humidity, or in a dusty place. When the computer is utilized in low temperatures, sometimes the display response is slow or does not operate. When normal temperature conditions are restored, however, the computer operation will become normal.
- Special care should be taken not to damage the computer by bending. For example, do not carry it in your hip pocket.
- Please do not connect units other than the FA-3 and FP-12S to the connector portion.
- Since "–" is displayed during calculation in which key operation is invalid except for certain keys, always confirm the display before pressing a key.
- Although the display sometimes becomes faint while buzzer is sounding, it is not a malfunction. However, if the display becomes very faint, replace the batteries with new ones as soon as possible.
- Every two years, replace the batteries with new ones even if the computer is not utilized. Do not leave exhausted batteries inside it because trouble may occur due to battery leakage.
- Always keep the cap for the connector portion when only the computer is used.
- If strong static electricity is applied to the computer, sometimes the memory content is changed, or key operation cannot be performed. If this occurs, remove the batteries, then replace them again.

- Always connect optional equipments after turning the computer power off.
- To clean the computer, do not use volatile liquids such as benzine or thinner, but wipe it with a soft dry cloth, or a cloth dampened with a neutral detergent solution.
- Do not turn the power off during program execution or operation.
- Since the computer is made up of precision electronic parts, avoid giving a strong shock while a program is being executed; otherwise the program execution may be stopped or the memory contents may be changed.
- When a malfunction occurs, contact the store where the computer was purchased or a nearby dealer.
- Before seeking service, please read this manual again, check the power supply, check the program for logic errors, etc.

1-1. NOMENCLATURE AND OPERATION



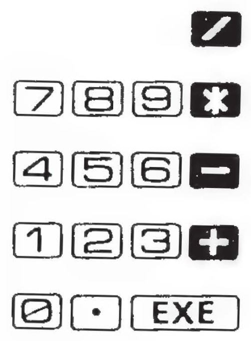
① Power switch	⑥ Alphabetical keys
② Shift key	⑦ Display window
③ Numeral and decimal point keys	⑧ Display contrast control
④ Calculation keys	⑨ Connector portion
⑤ Execution key	

Since many additional keys are provided compared to an ordinary calculator, the key functions might be unclear. Therefore each key and operation are explained.

- **Power switch**
  When this switch is moved to the right, the power is turned on, and when it is moved to the left, the power is turned off.

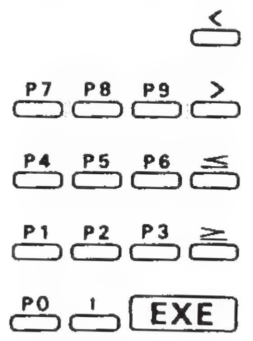
- **Shift key (Red [S] key)**
  If this key is pressed, the shift mode is selected (" [S] " is displayed) and the command or symbol printed above each key can be displayed. When it is pressed again, the shift mode is released and " [S] " disappears. (To distinguish this key from the alphabetical [S] key, it will be written as [SHIFT] from now on in this manual.)

- **Numeral keys, decimal point key, calculation keys, and execution key**
  Examine this key array carefully. It is the same as that of an ordinary calculator, isn't it? This part is used when the four arithmetic calculations (addition, subtraction, multiplication, division) are performed. However, the following differences exist. The [×] (multiplication) and [÷] (division) keys are different and there is no [=] key while there is an [EXE] (execution) key. This occurs because a computer uses an ∗ (asterisk) for X and / (slash) for ÷, while the answer is obtained by the [EXE] key instead of the [=] key.

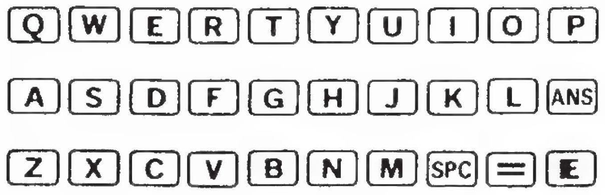


For example, an operation is performed by an ordinary calculator as 12 [×] 4 [÷] 3 [+] 7 [−] 5 [=] while this computer uses 12 [×] 4 [/] 3 [+] 7 [−] 5 [EXE] .

This computer can be used as an ordinary calculator as shown above. When followed by the [SHIFT] key, one of the numeral keys ([0] to [9]) can be used to specify a program area from P0 to P9 while the [·] key is used for power calculation ($x^y \rightarrow x\uparrow y$) and the [+] [−] [×] [/] keys are used to enter relational operators (≧, ≦, >, <).

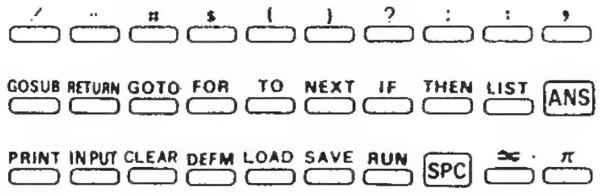


• Alphabetical keys, space key



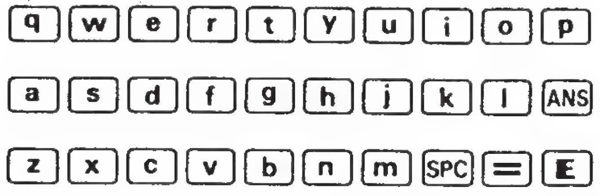
Using these keys, commands are entered, or progams are written. Each of the 26 alphabetical keys from A to Z functions as a memory (for storage locations).
Also, the A — Z keys have another function. When they are pressed after the SHIFT key, a symbol or BASIC command is displayed.
Press the SPC key when a space is required.

Example) SHIFT A → GOSUB、 SHIFT U → ?

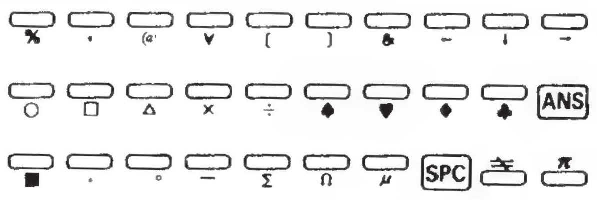

' " # $ ( ) ? : ; ,
GOSUB RETURN GOTO FOR TO NEXT IF THEN LIST ANS
PRINT INPUT CLEAR DEFM LOAD SAVE RUN SPC ≠ π

In addition, the alphabetical keys have another use in the extension mode (When · is pressed after the MODE key, "EXT" is displayed). When they are directly pressed, small alphabetical characters are displayed, and when they are pressed after the SHIFT key, special symbols are displayed.

Extension mode functions:



Functions provided when a key is pressed after the SHIFT key in the extension mode:



To release the extension mode, press MODE · again.

- **Equal key (=)**
  This key is not used to provide an answer for calculation, but is used for an assignment statement (see page 44) and for a condition in an IF statement (see page 66).
  Also, when this key is pressed after the SHIFT key, a ≠ (not equal) symbol is displayed.

- **Exponent/Pi key (E)**
  When this key is directly pressed, it is used to provide an exponent. For example, operate 1 · 2 3 E 4 for 1.23 X 10^4. When an exponent is a negative number, press the − key after this key. For example, operate 7 · 4 1 E − 9 for 7.41 X 10^{-9}.
  When this key is pressed after the SHIFT key, Pi (the ratio of the circumference of a circle to its diameter) is displayed.

- **Answer key (ANS)**
  When this key is pressed after the SHIFT key, the result of manual or program calculation executed immediately before is displayed.

- **Mode key (MODE)**
  This key is used together with · and 0 to 9 when the computer status or angle unit is specified.

MODE · ······ "EXT" is displayed to indicate extension mode in which small alphabetical characters and special symbols can be used. To release the extension mode, press these keys again.
MODE 0 ······ "RUN" is displayed for the performance of manual and program calculations.

MODE 1 ······ "WRT" is displayed for the performance of program write-in, checking, and editing.
MODE 2 ······ "TR" is displayed for the performance of execution trace. (see page 61 for details.)
MODE 3 ······ When "TR" is displayed, execution trace mode is released and "TR" disappears.
MODE 4 ······ "DEG" is displayed to indicate that degree is specified as the angle unit.
MODE 5 ······ "RAD" is displayed to indicate that radian is specified as the angle unit.
MODE 6 ······ "GRA" is displayed to indicate that grade is specified as the angle unit.
MODE 7 ······ "PRT" is displayed for the performance of printing when a printer is connected.
MODE 8 ······ When "PRT" is displayed, print mode is released and "PRT" disappears.

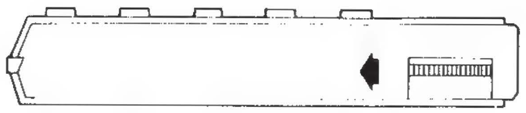
- **Cursor keys (← →)**
  These keys are used to move the cursor (blinking "–" in the display window) to the left or right as a convenience when correcting a displayed character. When they are pressed once, the cursor is moved one character, and when they are continuously pressed, the cursor moves continuously within the range of written characters.

- **All clear key (AC)**
  This key erases any display. Also, it is pressed when an error occurs, or when the display blanks out by auto power off (see page 19). When a program is being executed, program execution is suspended by pressing this key.

- **Delete/Insert key (INS DEL)**
  This key is used to delete a character where the blinking cursor is positioned. After deletion, the character to the right of the cursor moves to the left. When it is pressed after the SHIFT key, the character where the blinking cursor is positioned is moved to the right to provide a space.

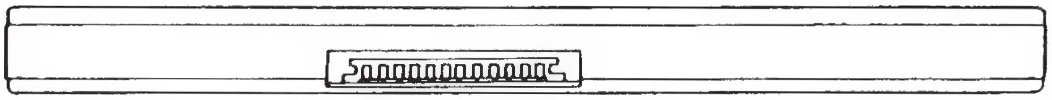
- **Stop key (STOP)**
  When this key is pressed during program execution, it is temporarily stopped. To resume it, press the EXE key.

- **Display contrast control**
  When the display is dark or faint, depending on the battery condition or display view angle, adjust it by moving the control located on the right side of the computer.



The display becomes darker when the control is turned in the direction of the arrow, and becomes lighter when turned in the opposite direction. If the display is still faint when this control is placed in the darkest position, the batteries are weak and should be replaced with new ones.

- **Connector portion**
  This connector is used when an optional equipment is connected. When printing is required, the FP-12S is connected to it, and when program storing on a tape required, the FA-3 is connected to it.



Do not connect any equipment other than the FP-12S (or FP-12) and FA-3 to this connector portion. When these optional equipments are not connected, always place the attached connector cap on it.

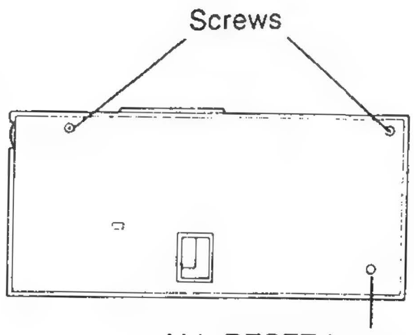
1-2. POWER SOURCE

Power for the computer is provided by two lithium batteries (CR2032). When only the computer is used, the battery life is about 140 hours. However, it is shortened if the buzzer is used often. If the display is faint even after the contrast is adjusted (see page 17), this is caused by weak batteries which should be replaced with new ones as soon as possible. Always replace both of the batteries at the same time.

* Replace the batteries with new ones every two years even if they are not used since leakage might occur.

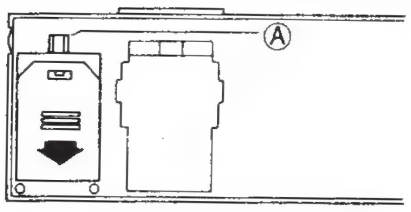
■ How to replace the batteries

(1) After turning the power switch off, loosen the two screws on the rear panel and remove it.

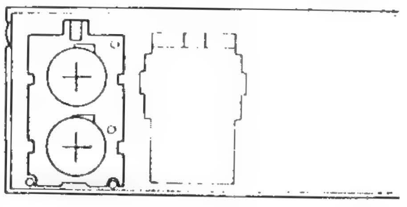


(After replacing the batteries, press with a pointed object.)

(2) While pressing Ⓐ, slide the battery compartment lid in the direction of the arrow and remove it.



(3) Remove the old batteries.
(This will be easier if you tap the unit lightly with the battery compartment facing down.)



(4) Using a dry cloth, wipe off the new batteries and insert them with the ⊕ side facing up.

(5) While pressing the batteries down with the battery compartment lid, slide it closed.

(6) Replace the rear panel and tighten the screws and after turning the power switch on, press the ALL RESET button.

* Never throw the old batteries into a fire. This is very dangerous as they might explode.
* Be sure to position the ⊕ and ⊖ terminals correctly.
* Keep the batteries away from children. If swallowed consult your doctor immediately.

■ Auto power off

The auto power off function prevents wasted power consumption when you forget to turn off the power switch. The power automatically turns off about 6 minutes after the last key operation (excluding program calculation). In this case, the power is turned on again by turning the power switch off and then on, or by pressing the AC key.

* Although the memory contents are not erased when the power is turned off, the angle and mode specifications ("RAD", "WRT", "TR", "PRT", etc.) are released.

1-3. RAM EXPANSION PACK (OPTION)

The standard RAM area (memory) of this unit is 544 steps/26 memories. However, this can be increased to a maximum of 1,568 steps/222 memories by using the OR-1Ⓔ optional RAM pack. This expanded RAM area can be used the same as the standard area and permits step number increase and memory expansion (see page 21).

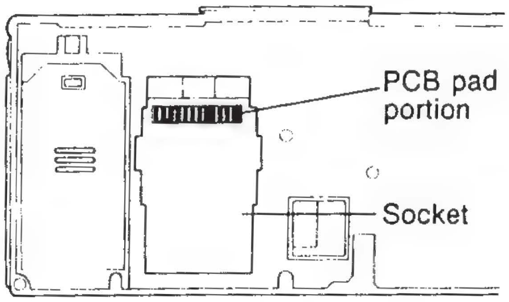
■ How to install the RAM pack

(preparation)
Since the internal circuitry may be damaged by static electricity, prior to handling the pack, be sure to ground yourself by touching some metallic object such as a doorknob so as to discharge any static electricity.

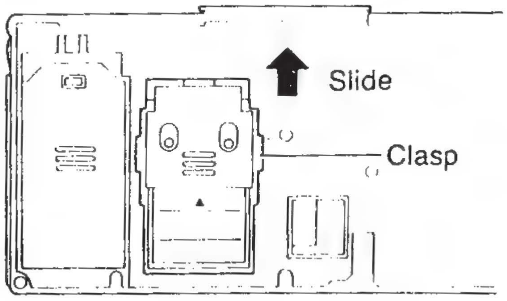
(procedure)

(1) Turn the power switch off.

(2) Loosen the two screws on the rear panel and remove it.



(3) Insert the pack into the socket on the computer body and slide the clasp into a locked position.
*Never touch the connector portion of the RAM pack or the PCB pad portion of the computer body.



(4) Replace the rear panel and tighten the screws.

- After installing or removing the RAM pack, be sure to press the ALL RESET button with a pointed object after turning the power switch on. If the ALL RESET button is not pressed, the memory contents may be changed or a meaningless display may be shown.
- Use care not to allow the connector portion of the pack or the PCB pad portion of the computer body to become dusty or dirty, and avoid getting fingerprints on them as this will cause poor contact.
- Be sure to place the removed pack in its case and store in a location where it is not subject to dust or dirt.

1-4. MEMORY EXPANSION

There are normally 26 memories (variables). The number of steps at this time is 544. The maximum number of standard memories is 94. Using a RAM pack, this can be expanded to 222. For memory expansion, program steps are converted to memory using 8 steps per memory.

Number of Memories	Number of Program Steps	
	Standard	Expanded
26	544	1568
27	536	1560
28	528	1552
⋮	⋮	⋮
46	384	1408
⋮	⋮	⋮
94	0	1024
⋮	⋮	⋮
200	—	176
⋮	⋮	⋮
222	—	0

Memory expansion is performed in units of 1 using a DEFM command.

Example)
Expand by 30 and make 56.

Operation)
Select the RUN mode (press MODE 0) or the WRT mode (press MODE 1).

DEFM 30 EXE

```
***VAR:56
```

*DEFM can be input by pressing D E F M or by pressing SHIFT V (DEFM).

A DEFM command is also used to confirm the number of memories which are currently designated.

Example)
A total of 56 memories are designated.

```
DEFM  EXE                    ***VAR:56
```

- When a large number of program steps are already in use, in order to protect the existing program, if a designation is attempted which would cause an insufficient number of steps, an error will occur. (ERR 1 ..... insufficient number of steps)
- The exclusive character variable ($) is not counted when designating since it is a special memory.

1-5. BEFORE CALCULATING

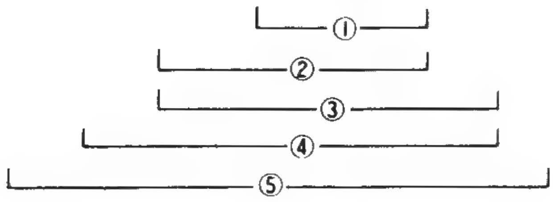
■ Calculation priority sequence

Calculations have "priority sequence" rules in which multiplication and division are performed prior to addition and subtraction. This computer is provided with a function that automatically distinguishes the priority sequence. This function is so convenient that a correct answer can be obtained by entering a calculation expression as it is.
The calculation priority sequence is determined as follows.

①Functions (SIN, COS, etc.)
②Power (↑)
③×(∗)、÷(/)
④+、−

Although calculation is performed according to this priority sequence, if two operations have the same priority, the left one has priority. If parentheses are used, operations inside parentheses have priority.

Example) 2 + 3 ∗ SIN (17 + 13) ↑2 = 2.75



■ Input/output number of digits and operation number of digits

The number of input digits are 12 digits for a mantissa and 2 digits for an exponent. Internal operations are also performed using 12 digits for a mantissa and 2 digits for an exponent.
Although the number of output digits is usually 10 digits for an mantissa, it differs depending on a displayed result of a manual calculation and that of a program calculation. In the manual calculation, the result is displayed up to 12 digits including mantissa, exponent and minus sign. While in the program calculation, 10 digit mantissa and 2 digit exponent are displayed. However, if 12 digits are exceeded, 12 digits from the beginning are displayed first, then the rest is displayed sequentially by shifting the display to the left.

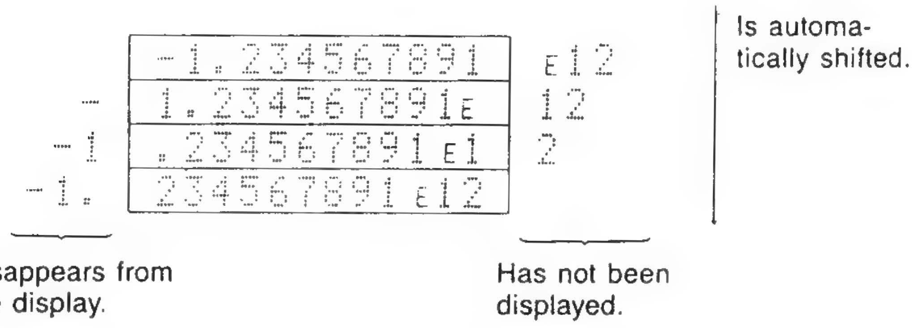
Example)

Manual calculation

1⊡2345678912 EXE	1.234567891
12345678912 E 100 EXE	1.2345678 E12
12345678912 E −100 EXE	−1.234567 E12

Program calculation

For PRINT 12345678912 E −100



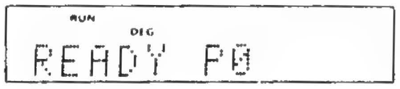
CHAPTER 2

LET'S OPERATE

You should operate this computer to become accustomed to it because it won't break even if it is wrongly operated. First, try a simple operation in accordance with the proverb "Practice makes perfect".

2-1. LET'S OPERATE THE COMPUTER

Learn how to operate the computer by actually using it.
To start, hold the computer and turn the power on by sliding the power switch to the right. Then the following is displayed.



Erase this display first by pressing the AC key. "READY P0" disappeared, didn't it? When this occurs, " _ " blinks on the extreme left of the display. This is called a "cursor" where a character can be written.

When the cursor blinks, this is called an "input wait state" in which the computer waits for the input of calculation or an instruction. While the cursor usually blinks as " _ ", it also blinks as " ▮ " while characters are continuously written. Up to 62 characters can be written on one line. When 56 or more characters are written, the " ▮ " sign appears as a warning signal. "RUN" and "DEG" on the display indicate the present status. "RUN" indicates the RUN mode in which manual calculations or program executions can be performed. "DEG" indicates that the angle unit is degree.
The angle unit also includes the Radian mode ("RAD" is on) which is specified by pressing MODE 5 , and the Grade mode ("GRA" is on) which is specified by pressing MODE 6 in addition to the above. The angle units are required when trigonometric functions are used. Whenever the power switch is turned on, "DEG" is displayed.
The status display also includes the program write-in mode ("WRT" is on) specified by pressing MODE 1 , the trace mode ("TR" is on, see page 61) specified by pressing MODE 2 , the print mode ("PRT" is on, see page 107) by pressing MODE 7, and the extension mode ("EXT" is on) by pressing MODE ·.
You will learn about these display modes as you continue to operate this computer.

Let's actually operate the computer to understand the display.
If a message stays displayed after using the corresponding mode, turn the power switch off and then turn it on again.

Start with a simple calculation.

Example) 123+456=579

```
_
```

Press AC.

Press appropriate keys to enter the numerical expression.

1 2 3 + 4 5 6

```
123+456_
```

After this, an answer is obtained by pressing EXE instead of = .

EXE

```
579
```

Next, try another calculation.

Example) 45×6+89=359

Consider that 45 was pressed as 46 by mistake.

4 6 × 6 + 8 9

```
46*6+89_
```

Now you notice that 46 was pressed by mistake. Place the cursor at the location where the wrong key was pressed by using a cursor key (←) calmly.

← ← ← ← ← ←

```
46*6+89
```

The cursor and 6 turn on and off.

Press the correct key, 5 .

5

```
45*6+89
```

Since the calculation expression is now corrected by performing the above procedure, press the EXE key to obtain the answer.

EXE

```
359
```

When a mistake was found in the middle as mentioned above, it can be easily corrected by using the cursor keys.
However, if the EXE key has been pressed, reenter the calculation expression from the beginning.

Now, let's enter characters by using the alphabet keys.
The array of the alphabet keys is the same as that of a QWERTY typewriter. Most personal computers now have QWERTY type array keyboards; remember the location of the characters even though it may not be easy for a beginner to do this.

Enter characters first.

Example) The characters used are "ABCXYZ".

Enter ABC.

$\boxed{A}\boxed{B}\boxed{C}$

```
ABC_
```

Enter XYZ next.

$\boxed{X}\boxed{Y}\boxed{Z}$

```
ABCXYZ_
```

Now put a space between ABC and XYZ.
Place the cursor on X.

$\boxed{\leftarrow}\boxed{\leftarrow}\boxed{\leftarrow}$

```
ABCXYZ
```

Make one character space.

$\boxed{SHIFT}\boxed{\overset{INS}{DEL}}$

```
ABC_XYZ
```

When a space is to be inserted between characters, place the cursor at the location where it is to be inserted, then press $\boxed{SHIFT}\boxed{\overset{INS}{DEL}}$.
When additional spaces are to be inserted, repeat this procedure.

This computer is provided with some special characters which are convenient for games or as scientific symbols in addition to alphanumeric characters. (See page 14 for the special characters.)

Let's practice displaying some of these characters.

Example) Display ♤♡◇♧ marks.

Specify the extension mode.

$\boxed{AC}\boxed{MODE}\boxed{\cdot}$

Displayed

```
EXT
_
```

For these marks, press alphabet keys after pressing the $\boxed{SHIFT}$ keys.

$\boxed{SHIFT}\boxed{♠}\boxed{SHIFT}\boxed{♥}\boxed{SHIFT}\boxed{♦}\boxed{SHIFT}\boxed{♣}$

```
♠♥♦♣_
```

Example) Display Σ, Ω, μ symbols.

Since the extension mode is used, just perform the following operations.

SHIFT $\overset{\Sigma}{\text{B}}$ SHIFT $\overset{\Omega}{\text{N}}$ SHIFT $\overset{\mu}{\text{M}}$

```
♠♥♦♣ΣΩμ_
```

Since these marks and symbols are provided, please try to use them. Also, to return to the mode in which upper case letters are displayed, press MODE · again to erase "EXT". During the continuous use of this computer, sometimes "ERR2" is displayed and it does not operate even if a key is pressed. This is not computer trouble, but is a message called an "error message". When this occurs, press the AC key, then the display clears and the computer operates again. See pages 56 and 166 for details.

2-2. SIMPLE CALCULATION, AT THE BEGINNING

Simple calculation is performed as follows. However, if you have never used a scientific calculator, please be careful because this computer is provided with True Algebraic Logic functions in which multiplication and division are performed before addition and subtraction.

Example 1) 23+4.5−53=−25.5

Operation) $\boxed{2}\boxed{3}\boxed{+}\boxed{4}\boxed{.}\boxed{5}\boxed{-}\boxed{5}\boxed{3}\boxed{\text{EXE}}$ $\boxed{-25.5}$

* Numeral keys are shown without a frame from now on.

Example 2) 56×(−12)÷(−2.5)=268.8

Operation) 56 $\boxed{\times}\boxed{-}$ 12 $\boxed{/}\boxed{-}$ 2.5 $\boxed{\text{EXE}}$ $\boxed{268.8}$

* To enter a negative number, press the $\boxed{-}$ key before a number.

Example 3) 7×8−4×5=36

Operation) 7 $\boxed{\times}$ 8 $\boxed{-}$ 4 $\boxed{\times}$ 5 $\boxed{\text{EXE}}$ $\boxed{36}$

* Multiplication is performed first, then subtraction is performed.

Example 4) $(4.5\times10^{75})\times(-2.3\times10^{-78})=-0.01035$

Operation) 4.5 $\boxed{\text{E}}$ 75 $\boxed{\times}\boxed{-}$ 2.3 $\boxed{\text{E}}\boxed{-}$ 78 $\boxed{\text{EXE}}$ $\boxed{-0.01035}$

* Press the $\boxed{\text{E}}$ key then enter the exponent.

There is another algebraic calculation which uses the memory. This memory is convenient when a certain numerical value is calculated in many different ways.

For example,

$3x+5=$

$4x+6=$

$5x+7=$

When the value of *x* is 123.456 in these calculations, it is troublesome to press the same numerical value repeatedly. Is there any way to perform these calculations without this trouble? The solution is to use a memory called a variable. In these examples, since *x* is used for algebraic calculations, the calculations are performed by using variable X.

First, assign 123.456 to variable X.

X = 123.456 EXE

"=" does not mean equal but means that 123.456 is assigned to variable X.
Let's perform these calculations.

3 * X + 5 EXE
4 * X + 6 EXE
5 * X + 7 EXE

```
375.368
499.824
624.28
```

They can be performed easily.

Since this computer is provided with 26 variables from A to Z, many different numerical values can be memorized.
In these examples, the numerical value for variable X is fixed and the calculation expressions are different.
However, how about a case in which the calculation expressions are fixed and the values of the variables are different?
When a calculation expression is determined to be "$3x + 5=$" and the value of x is changed to 123, 456, and 789, the operation is troublesome if the procedure mentioned above is used. Actually, the calculation expression is memorized by the computer and it is only necessary to change the value of variable X. This convenient calculation method is called "program calculation". A strong point of this computer is program calculation. Manual calculation, a previous step in which a program is used, is performed here. See Chapter 3 "BASIC" PROGRAMMING for a Program.

2-3. FUNCTION CALCULATION — A HIGHLIGHT OF THIS COMPUTER

This computer is provided with scientific functions as well the four basic functions.
Although these functions can be utilized in a program, manual utilization is explained here.
The functions provided by this computer are as follows.

Name of functions		Format
Trigonometric functions	$\sin x$	SIN x
	$\cos x$	COS x
	$\tan x$	TAN x
Inverse trigonometric functions	$\sin^{-1} x$	ASN x
	$\cos^{-1} x$	ACS x
	$\tan^{-1} x$	ATN x
Square root	$\sqrt{x}$	SQR x
Common logarithm	$\log x$	LOG x
Natural logarithm	$\ln x$	LN x
Exponential function	e^x	EXP x
Power	x^y	$x \uparrow y$
Decimal→sexagesimal		DMS $ (x)*
Sexagesimal→decimal		DEG (x,y,z)*
Integer		INT x
Integer removal		FRAC x
Absolute value	$\|x\|$	ABS x
Coding	Positive No.→1; 0→0; Negative No.→ −1	SGN x
Rounding off	x is rounded off at the 10^y position.	RND (x,y)*
Random number generation		RAN #

* For DMS $, DEG, and RND, the argument must be inside (   ).

Perform calculations with functions.

- **Trigonometric functions (sin, cos, tan), and inverse trigonometric functions ($\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$)**
  When trigonometric functions and inverse trigonometric functions are used, always be sure to specify the angle unit (DEG, RAD, GRA).

Example) sin 12.3456° = 0.2138079201

Operation) MODE 4 → "DEG"
S I N 12.3456 EXE

```
0.2138079201
```

* From now on, alphabet keys are shown without frames.

Example) cos 63°52′41″ = 0.4402830847

Operation) COS DEG SHIFT (63 SHIFT , 52 SHIFT , 41 SHIFT) EXE

```
0.4402830847
```

Example) 2·sin45° × cos65.1° = 0.5954345575

Operation) 2 × S I N 45 × COS 65.1 EXE

```
0.5954345575
```

Example) $\sin^{-1}0.5 = 30°$

Operation) ASN 0.5 EXE

```
30
```

Example) $\cos(\frac{\pi}{3}\text{rad}) = 0.5$

Operation) MODE 5 → "RAD"
COS SHIFT (SHIFT π / 3 SHIFT) EXE

```
0.5
```

Example) $\cos^{-1}\frac{\sqrt{2}}{2} = 0.7853981634\text{rad}$

Operation) ACS SHIFT (SQR 2 / 2 SHIFT) EXE

```
0.7853981634
```

Example) tan(−35gra) = −0.612800788

Operation) MODE 6 → "GRA"
TAN − 35 EXE

```
-0.612800788
```

• Logarithmic functions (log, ln), and exponential functions (e^x, x^y).

Example) $\log 1.23 (= \log_{10} 1.23) = 0.0899051114$
Operation) LOG 1.23 EXE

```
0.0899051114
```

Example) $\ln 90 (= \log_e 90) = 4.49980967$
Operation) LN90 EXE

```
4.49980967
```

Example) $e^5 = 148.4131591$
Operation) EXP5 EXE

```
148.4131591
```

Example) $10^{1.23} = 16.98243652$
(The anti-logarithm of common logarithm 1.23 is obtained.)
Operation) 10 SHIFT ⏏ 1.23 EXE

```
16.98243652
```

Example) $5.6^{2.3} = 52.58143837$
Operation) 5.6 SHIFT ⏏ 2.3 EXE

```
52.58143837
```

Example) $123^{\frac{1}{7}} (= \sqrt[7]{123}) = 1.988647795$
Operation) 123 SHIFT ⏏ SHIFT ⏏ 1 ◢ 7 SHIFT ⏏ EXE

```
1.988647795
```

* When $x < 0$, y is a natural number.

Example) $\log \sin 40° + \log \cos 35° = -0.278567983$
The anti-logarithm is 0.5265407845 (logarithmic calculation of sin 40° × cos 35°).
Operation) MODE 4 → " DEG "
LOG SIN 40 + LOG COS 35 EXE
10 SHIFT ⏏ SHIFT ANS EXE

```
-0.278567983
0.5265407845
```

• Other functions (√, SGN, RAN#, RND, ABS, INT, FRAC)

Example) $\sqrt{2} + \sqrt{5} = 3.65028154$
Operation) SQR 2 + SQR 5 EXE

```
3.65028154
```

Example) Gives "1" if it is a positive number, "−1" if it is a negative number, and 0 if it is "0".

Operation) SGN 6 [EXE]
SGN 0 [EXE]
SGN [−] 2 [EXE]

```
1
0
-1
```

Example) Random number generation (Pseudo random numbers with the range of 0 < RAN # < 1).

Operation) RAN [SHIFT] [#] [EXE]

```
0.7903739076
```

Example) Round off the result of 12.3 × 4.56 to one decimal place.

12.3×4.56=56.088

Operation) RND [SHIFT] [(] 12.3 [×] 4.56 [SHIFT] [,] [−] 2 [SHIFT] [)] [EXE]

* When RND (x,y) is used, $|y| < 100$.

```
56.1
```

Example) |−78.9÷5.6|=14.08928571

Operation) ABS [SHIFT] [(] [−] 78.9 [/] 5.6 [SHIFT] [)] [EXE]

```
14.08928571
```

Example) Integer of 7800/96 81

Operation) INT [SHIFT] [(] 7800 [/] 96 [SHIFT] [)] [EXE]

```
81
```

* The maximum integer that does not exceed the original numerical value is obtained by this function.

Example) The fraction of 7800/96 ..... 0.25

Operation) FRAC [SHIFT] [(] 7800 [/] 96 [SHIFT] [)] [EXE]

```
0.25
```

• Designation of number of significant positions, and designation of number of decimal positions.

The number of positions are designated by the "SET" command.

Designation of number of significant positions	SET E *n* (*n* =0 to 8)
Designation of number of decimal positions	SET F *n* (*n* =0 to 9)
Cancellation of designations of number of positions ...	SET N

* In a manual calculation, "SET E0" designates an 8 significant positions.
* Even if a designation is performed, the original numerical value remains in the memory.

Example) 100÷6=16.66666666········
Operation) SET E4 EXE (Designation of 4 significant positions.)
100 / 6 EXE

```
1.667E01
```

Example) 123÷7=17.57142857······
Operation) SET F2 EXE (Designation of 2 decimal positions)
123 / 7 EXE

```
17.57
```

Example) 1÷3=0.3333333333········
Operation) SET N EXE (Designation cancellation)
1 / 3 EXE

```
0.3333333333
```

• Decimal ↔ sexagesimal conversion (DEG, DMS $)

Example) 14°25'36"=14.42666667
Operation) DEG SHIFT (14 SHIFT , 25 SHIFT , 36 SHIFT) EXE

```
14.42666667
```

Example) 12.3456° = 12°20′44.16″

Operation) DMS [SHIFT] [$] [SHIFT] [(] 12.3456 [SHIFT] [)] [EXE]

```
12° 20' 44.16
```

Example) sin 63°52′41″ = 0.897859012

Operation) [MODE] [4]

SIN DEG [SHIFT] [(] 63 [SHIFT] [,] 52 [SHIFT] [,] 41 [SHIFT] [)] [EXE]

```
0.897859012
```

CHAPTER 3

"BASIC" PROGRAMMING

In this chapter, BASIC programs and programming are explained. Users who are not accustomed to programming should read this chapter to master the basics of programming.

3-1. WHAT IS A PROGRAM?

The word "PROGRAM" may sound like something difficult. However, there are very simple programs and more complicated ones. For example, calculation, in which algebraic expressions are memorized and numerical values are assigned to these expressions, is also a program.

3-1-1 A Program Is Convenient.

We are concerned with many different kinds of calculations such as those for financial business accounting, measurements, or for housekeeping and expenses. Although it is not so troublesome if these calculations are performed only once, it is tedious to perform calculations repeatedly with the same calculation expression while changing numerical values. Because of this, these calculations can be best performed by using your computer. For example, if the calculation expression $y=2x^2+5x+13$ is used, to obtain the value of y when the value of x is changed, the same calculation must be repeated. To eliminate this trouble, the following expression is placed in memory.

```
10 INPUT X
20 Y=2*X↑2+5*X+13
30 PRINT Y
```

In this program a calculation expression is memorized. A detailed explanation will be provided later. This program allows the calculation to be easily performed.
Simple programs can be conveniently used just by memorizing a calculation expression as mentioned above.
Next, the program will be sequentially explained.

3-1-2 Program Construction

Remember program construction.

```
10 INPUT X
20 Y=2*X↑2+5*X+13
30 PRINT Y
```

This program can be divided into 3 parts as follows.

```
10  INPUT  X ························· Input
20  Y=2*X↑2+5*X+13 ··········· Calculation
30  PRINT  Y ························· Output
```

At first, the input part is used to enter (input) data (such as numerical values for calculation) into the computer. Next, the calculation part is used to perform a calculation so that an answer can be provided. Last, the output part is used to provide (display) an answer.
A computer does everything required if correct commands (instructions) are provided. In this example, an input command (INPUT) and an output command (PRINT) are memorized.
These three parts can be further broken down as follows.

$$\underbrace{10}_{\text{Line No.}}\quad \underbrace{\text{INPUT}}_{\text{Command}}\quad \underbrace{X}_{\text{Operand}}$$

The line numbers indicate the sequence of the program flow. Since a computer reads and executes statements in ascending order of line numbers, place these line numbers according to the expected execution sequence. Also, it is advisable to assign these numbers in 10s (10, 20, 30, ...) becasue this is convenient if additions are required later. Decimal figures such as 1.5 or 12.3 cannot be used for line numbers.
The items that follow the line number are the commands to be performed by the computer. There are many different kinds of commands used for specification of instruction required.
Although it is desirable to remember all the commands, just memorize the minimum necessary commands at first, then the rest gradually. See "CHAPTER 4 COMMAND REFERENCE" on page113and after for the kinds of commands and their functions. The function of an operand next to a command is to supplement it. Some commands have an operand while other commands do not. In this example, the INPUT command indicates the entry of data. An operand specifies memory where entered data is placed; in this case, the entry to variable X.

In the following line,

$$\underbrace{\texttt{20}}_{\text{Line No.}}\ \underbrace{\texttt{Y=2*X}\uparrow\texttt{2+5*X+13}}_{\text{Assignment statement}}$$

20 is the line number. The assignment statement means that the value on the right of the equal sign (=) is entered (assigned) to the variable on the left. If the LET command is added to the assignment as follows,

```
    20 LET Y=2*X↑2+5*X+13
```

LET is a command and the assignment statement is an operand.
Line 30 consists of a line number, command, and an operand the same as line 10.

$$\underbrace{\texttt{30}}_{\text{Line No.}}\ \underbrace{\texttt{PRINT}}_{\text{Command}}\ \underbrace{\texttt{Y}}_{\text{Operand}}$$

Program construction is as mentioned above. Additional commands, operands, and line numbers are used to construct a large program.

3-1-3 Easy Program Preparation

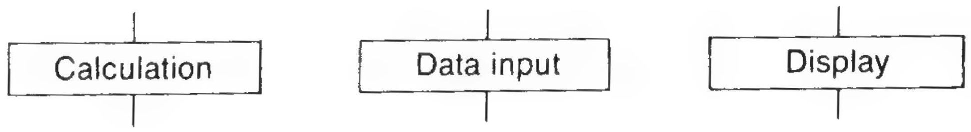
When program construction is understood, it is not so difficult to prepare a program.
After the three main parts (input, calculation, and output) are understood, they can be used to prepare a program. It is unnecessary to remember all the commands at one time. It is advisable to try simple calculation by just using "INPUT", "PRINT", and an assignment statement.
The secret of quick program mastery is not to just remember a program, but to actually prepare a program by selecting a problem that can be easily placed into a calculation expression from among those around you such as the calculation of bills or financial calculation repeatedly performed in a company, measurements, housekeeping and time calculation for cassette recording often performed at home. The best way to master programming is to first prepare a program for the subject to process, memorize the required commands, then improve the program.
Another secret is not to prepare the entire program at one time but to prepare its different parts and put them together later.
Prepare a program gradually and slowly by using this concept. After fundamental program construction has been understood, a program can be prepared by referring to each command function in the "CHAPTER 4 COMMAND REFERENCE" and by actually using them.

3-2. PROGRAM PREPARATION

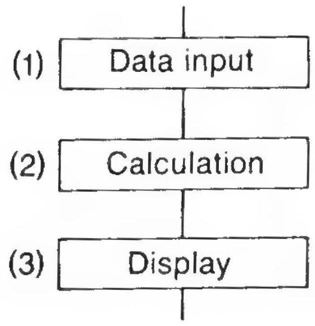
This section and after cover the main subject, BASIC Programming, in which a program is actually prepared.

3-2-1 Preparing A Flowchart

You may not be accustomed to a flowchart. It is a chart which describes a work sequence.
It is often heard that a flowchart is not required for BASIC; this is correct for a user who is accustomed to utilizing a computer. However, since the entire work procedure is hard to understand for a beginner, it is important to draw a simple flowchart to understand the program flow.
While there are formal and dedicated symbols for drawing flowcharts, it is unnecessary to remember these symbols. Just place each work item inside ☐ and connect them with lines.
Let's prepare a program to explain each item.
For example, make a program to obtain the square of a numeral by entering it. First, a calculation part is required. Since a numeral is entered, an input part is necessary, and since an answer is displayed, an output part is necessary. The three parts are placed inside ☐ as follows.

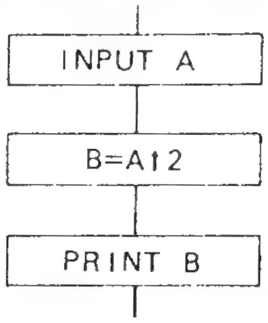


When these three items are sequentially connected, it is easily found that the last item is the display of the answer. Next, it is necessary to perform calculation to obtain an answer, and also to enter data to perform calculation. Connect these three items.



As the flowchart has been completed by this procedure, let's change it into a format that is more like a program.

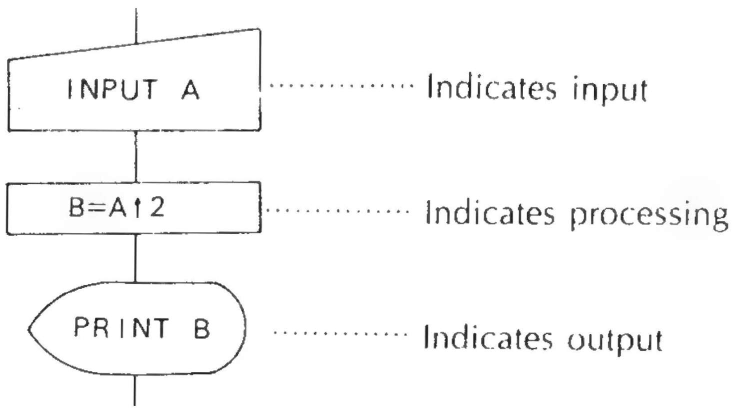
The first item is an instruction to enter data. Since data is entered into a variable by using an INPUT statement, determine the variable. If variable A is used, the content of (1) is "INPUT A".
In the calculation performed in the second item, the entered content of variable A is squared, and the answer is assigned to another variable. If this variable is B, then "B=A↑2". This calculation expression is called an assignment statement which is formally written as "LET B=A↑2". However, since LET can be omitted, it can be written as "B=A↑2".
The answer is displayed by the third item. Since the content of variable B which includes the answer is displayed by using a PRINT statement, it is written as "PRINT B". These three items are placed into a flowchart again.



This program is completed by placing line numbers for these three items.

```
10 INPUT A
20 B=A↑2
30 PRINT B
```

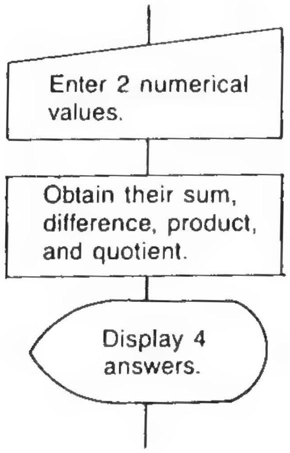
A program can be quickly and easily made by sequentially assembling a flowchart after preparing each item as mentioned above.
An actual flowchart has formal symbols which have special meanings. Draw the above example by using them.



Refer to the flowchart symbol at the end of this manual.

3-2-2 Preparing A Program

To provide a simple example, enter 2 numerical values and obtain their sum, difference, product, and quotient.
Prepare a flowchart based on the input, calculation, and output parts which are the three important items.

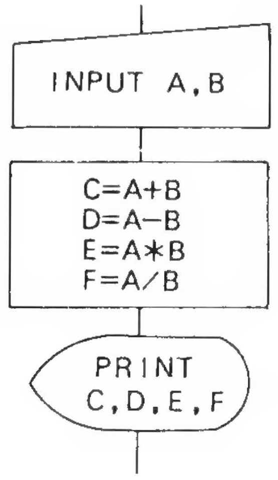

Enter 2 numerical values.
Obtain their sum, difference, product, and quotient.
Display 4 answers.

Use an INPUT statement (an instruction to enter data from the keyboard into a variable) to enter 2 numerical values. Items that should be noted are the variables where data or an answer are entered. The utilization of variables is quite complicated. Variables include those with alphabetical characters from A to Z, and array variables that have an item called a subscript such as A(3). Although a variable can be selected from among these variables, it is recommended that variables be selected in alphabetical order while you are not accustomed to programming.
Two numerical values are entered here. A and B are selected and "INPUT A, B" is written. Several variables can be handled in one INPUT statement by punctuating them with commas.
How about the calculation parts? Four items are calculated here; four answers will be obtained. The variables where the four answers are entered will be C, D, E and F.

First, For addition of A+B, C=A+B is used.
For subtraction of A−B, D=A−B is used.
For multiplication of A*B, E=A*B is used.
For division of A/B, F=A/B is used.

Since this completes the calculation items, the answers are displayed next. The display command is PRINT; "PRINT C, D, E, F" is realized.

The program format can be placed in a flowchart as follows.



Complete the program by placing line numbers.

```
10 INPUT A,B
20 C=A+B
30 D=A-B
40 E=A*B
50 F=A/B
60 PRINT C,D,E,F
```

Next add "END" to indicate program termination.

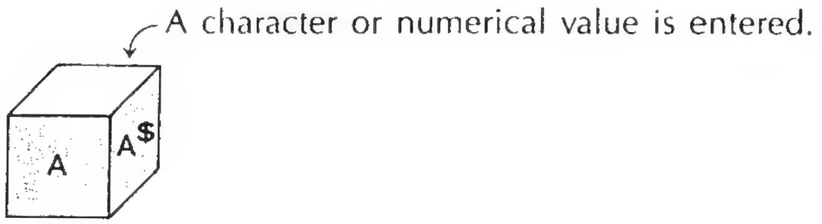
```
70 END
```

The program has been completed by the above procedure. It can be easily written if it is assembled sequentially.
First, prepare a simple and practical program instead of complicated one by using many different commands.

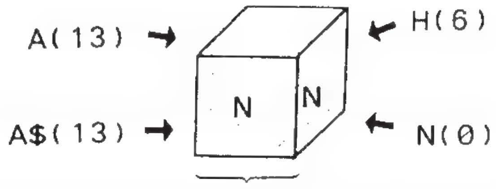
NOTE

VARIABLES
Variables are important elements for program preparation. Variables are just like boxes where entered data or calculated data are stored with each having a name. Variables include the standard ones from A to Z and those with a subscript attached to the name (A to Z). The latter ones are called array variables such as A(5) and B(50).

<Precautions when using variables>
If a numerical variable has the same name as a character variable when a program is used, data will be entered into the same place.



As a result, the numerical variable A and the character variable A$ cannot be used simultaneously in the same program.
Also, when an array variable is used (see page 84) precautions must be taken so as not to give several equivalent names to one container.



All names are for the same container.

```
A=A(0)=A$=A$(0)
B=A(1)=B(0)=B$=A$(1)=B$(0)
C=A(2)=B(1)=C(0)=C$=A$(2)=B$(1)=C$(0)
:
N=A(13)=B(12)=……=N(0)=N$=A$(13)=……N$(0)
:
Z=A(25)=B(24)=C(23)=……=Z$=A$(25)=…… Z$(0)
```

Same precautions must be taken when memory is expanded by a DEFM statement.

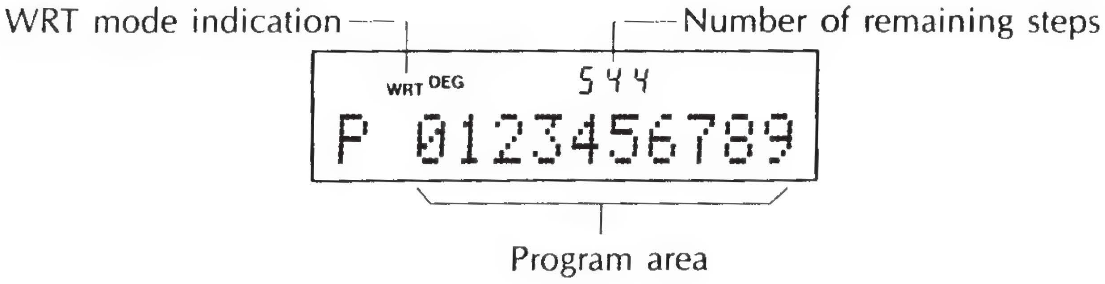
3-2-3 Program Input

Memorize (input) a program.
Use the previous program in which four basic calculations are performed after entering two numerical values.

Program

```
10 INPUT A,B
20 C=A+B
30 D=A-B
40 E=A*B
50 F=A/B
60 PRINT C,D,E,F
70 END
```

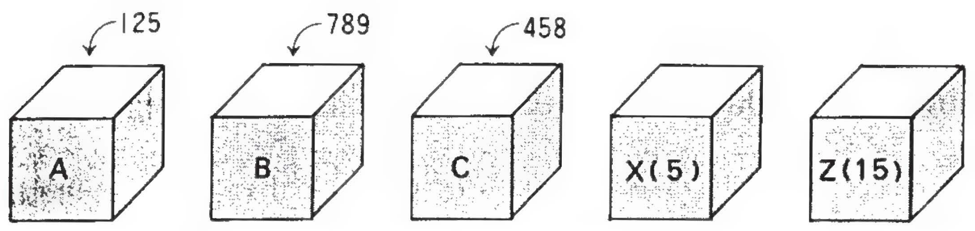
When the power switch is turned on, the RUN mode, in which manual calculations or program execution can be performed, is specified. Press MODE 1 to switch from this mode to the WRT mode in which a program can be written.



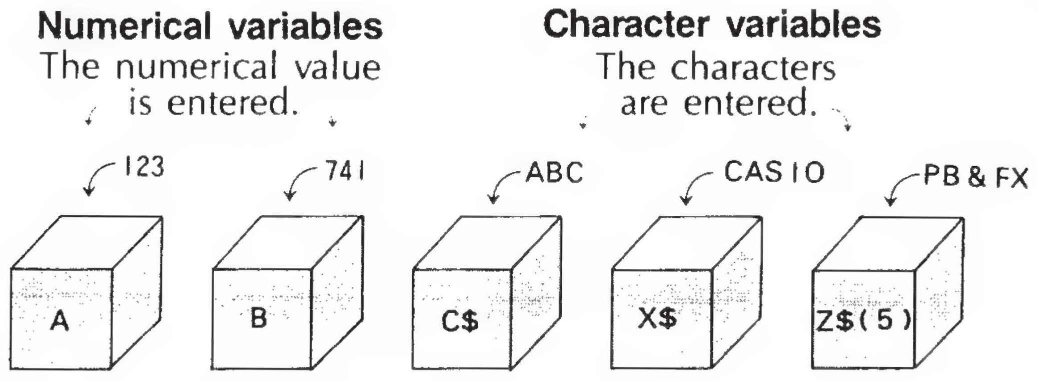
This display shows a status in which no program is stored. The above 3-digit numeral indicates the number of remaining steps. This number decreases when a program is stored or when the memory is expanded (see page 87). Numerals from 0 to 9 are program area numbers; the blinking one indicates the currently specified program area. When a program is stored during this status, it is stored in program area P0. Different programs can be written in 10 program areas from P0 to P9.
If a program is stored, the program area number is not displayed but the cursor, "–", is displayed. To erase all the programs and store a program in program area P0, enter

NEW ALL EXE

This is a command that erases all programs. Store a program with the following procedure.



Also, there are two different types of variables; numerical variables where numerical values are entered, and character variables where character strings are entered. The variables that were previously used are numerical variables where numerical values have been entered to perform a calculation. In addition to these, there are character variables with $ attached to the name (A to Z), such as A$, B$, C$, and a special character variable called "exclusive character variable", $.
A numerical value with up to 10 digits (10 digits in the mantissa part, 2 digits in the exponent) can be entered into a numerical variable, while a string with up to 7 characters can be entered into a character variable. Also, up to 30 characters can be entered into the exclusive character variable.



Since the items entered in these two kinds of variables are different, characters such as "ABC" cannot be entered into numerical variables while numerical values for calculations cannot be entered into character variables. The utilizations of these variables are different. Use numerical variables when numerical values are to be entered for calculation, and character variables when messages or symbols are to be entered.
Arrays are convenient when data are stored in many variables. They are distinguished by subscripts indicating the 1st variable, 2nd variable, etc. Array variables will be explained by utilizing them in a program.

1 0 SHIFT INPUT A SHIFT , B   EXE   Press this key at the end of each line.
This operation can also be performed by pressing I N P U T.
2 0 C = A + B   EXE
3 0 D = A − B   EXE
4 0 E = A × B   EXE
5 0 F = A / B   EXE
6 0 SHIFT PRINT C SHIFT , D SHIFT , E SHIFT , F   EXE
7 0 E N D   EXE

After the EXE key is pressed, a one character space is made after the line number to allow the display to be easily read.
Was the program correctly stored?
Press the keys slowly and firmly even if it is boring. When a wrong key was pressed, a correction can be made by the following operation.

• A mistake was noticed before the EXE key was pressed.

If this occurs, place the cursor at the location where the mistake occurred by using the ⇦ ⇨ keys and correct it.

Example 1) `10 INPUT S_` "S" was pressed instead of "A".
Place the cursor under the "S" by pressing the ⇦ key once.

⇦   `10 INPUT S`

Press the correct key.

A   `10 INPUT A_`

Complete the entry then press the EXE key.

SHIFT , B   EXE   `10 INPUT A,B`

Example 2) `40 EE=A*B_` An extra "E" was entered.
Press the ⇦ key 5 times and place the cursor under the second "E".

⇦ ⇦ ⇦ ⇦ ⇦   `40 EE=A*B`

Since 1 character is to be deleted, press the DEL key once.

DEL   `40 E=A*B`

After the deletion is made, press the EXE key.

`40 E=A*B`

Example 3) `RINT C, , E, F_` "D" on line 60 was skipped.

Press the ⇦ key 4 times to place the cursor next to the insertion location.

⇦ ⇦ ⇦ ⇦ `60 PRINT C, ,`

Provide one character space.

SHIFT INS `60 PRINT C, _`

Insert "D".

D `0 PRINT C, D,`

After the correction has been completed, press the EXE key.

EXE `60 PRINT C, D`

• **A mistake was noticed after the EXE key was pressed.**

Since a line is stored as part of a program after the EXE key is pressed, recall it by using the LIST command to correct.

Example) Line 50 was mistakenly entered as "50 F=A/N".

Recall line 50 by using the LIST command.

SHIFT LIST 50 EXE `50 F=A/N_`

Pressing L I S T provides the same result.

Press the ⇦ key once to place the cursor under the "N".

⇦ `50 F=A/N`

Press the correct key.

B `50 F=A/B_`

After the correction has been completed, press the EXE key.

EXE

If lines 60 and after are stored, line 60 is displayed. `60 PRINT C, D`

If no other correction is required, press the AC key to clear the display.

AC

After the EXE key is pressed, a correction can be performed by recalling the line with the LIST command. Also, the line can be rewritten with a new line number.

When a new line is stored with a number which has been already used, the line stored later has priority, and the old one is erased.

A program is stored as mentioned above. After the storage operation has been completed, press MODE Ø to return to the RUN mode. If the WRT mode is maintained, a stored program may be erased or changed by mistake. Therefore, be sure to return to the RUN mode after completion of the storage operation.

NOTE

PROGRAM AREAS

This computer is provided with 10 program areas, PØ to P9 where independent programs can be stored. All these program areas can be used in the same way. For example, if these areas were not provided and if 3 programs were to be used very often, they would have to be loaded from tape each time. This computer can store these 3 programs in 3 program areas such as PØ, P1 and P2.
Although this function is very convenient, precautions have to be taken concerning the number of steps used; the total number of steps used in all program areas must not exceed the maximum capacity.
A program area can be specified by pressing a key from Ø to 9 after pressing the SHIFT key. This specification can be done both in the RUN mode and WRT mode. In the RUN mode, the program stored in the specified area automatically starts. In the WRT mode, the program does not start but a program area, where a program is to be input or editing is to be performed, is specified.
The program areas must be correctly handled. When a program is executed, stored on cassette tape, or loaded from cassette tape, if a wrong area is specified, the operation cannot be performed correctly.
When the power is turned on, program area PØ is specified automatically. This can be confirmed by the numeral following "READY" after you press MODE Ø .

Example) MODE Ø→ READY P3 ······ Program area P3

3-2-4 Program Execution

Perform calculation with the program that was previously stored. Press MODE 0 and confirm that the computer is in the RUN mode ("RUN" is displayed).
There are two different ways to execute a program.

(1) Execution by specifying the program area.
After pressing the SHIFT key, press the numeral key from 0 to 9 which specifies a program area. If a program is stored, it will start.

Example) SHIFT P0

(2) Execution by the RUN command.

Example) SHIFT RUN (same as R U N) EXE

The difference in these two execution methods is as follows. In method (1), execution always starts from the beginning of the program area while in method (2), execution can be started from the beginning or from an arbitrary line number.

Execute a program with method (1).

Operation	Display
SHIFT P0	?

Enter 2 data.

Example)

Operation	Display
45 EXE	?
36 EXE	81 STOP

When the PRINT statement is executed, "STOP" is lit.

After 2 data were entered, the sum was displayed. Press the EXE key to display the next answer.

Operation	Display
EXE	9 STOP
EXE	1620 STOP
EXE	1.25 STOP

Next, execute a program by using the RUN command. If RUN EXE is entered, the result is the same as that obtained by method (1). Therefore, execute it from line 20.

Operation

SHIFT RUN 20 EXE (Same as RUN 20 EXE .)	81
EXE	9
EXE	1620
EXE	1.25

If no line number is specified when the RUN command is used, program execution starts from the beginning, and if a line number is specified, execution starts from that line number as mentioned above.
Actually there is another difference between these two methods.

When the RUN command is used, the program in the currently specified program area is executed. However, if P0 is to be executed while P5 is specified, what is the procedure?
The solution is to press SHIFT P0.

After a program has been prepared and stored, execute it. Even if an error (ERR is displayed) occurs after execution, don't be disappointed. In this case, find the cause of the error (debug) by referring to the following section.

NOTE

HOW TO COUNT THE NUMBER OF STEPS

This computer is provided with a memory capacity of 544 steps and 1568 steps with OR-1Ⓔ RAM expansion pack (option).
A step is the unit that indicates the memory capacity in which a program can be stored. As a program is stored, the number of remaining steps is reduced.
When the WRT mode is specified by pressing MODE 1, the current number of remaining steps is displayed.

Example)

MODE 1

```
WRT       544 ----
P 0123456789
```

Number of remaining steps

The number of steps is counted as follows.

- Line No. .... 2 steps per line No. from 1 to 9999.
- Command.............................. 1 step
- Function................................ 1 step
- Character............................... 1 step per character.
- In addition, each press of the EXE key during storage is counted as 1 step.

Example)

```
  1   INPUT   A  EXE  ········ 5 steps
  2     1     1   1
 10   B=SIN   A  EXE  ········ 7 steps
  2   1 1  1  1   1
100   PRINT  "B=" ;B  EXE ·········10 steps
  2     1     4   1 1  1
                                 Total: 22 steps
```

- When the memory is expanded, 8 steps are required for 1 memory expansion.

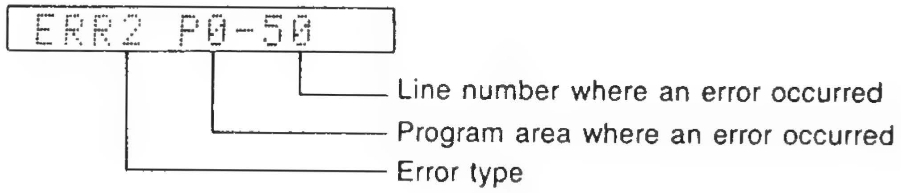
Example) Initial state..................................26 memories 544 steps
DEFM 10 EXE ....36 memories 464 steps

3-2-5 Debugging (error correction)

After a program has been prepared and executed, it often happens that an error is displayed and a result cannot therefore be obtained. Don't be disappointed since the cause of this error can be found.
Eliminating errors is called "debugging".
The debugging method depends on the cause of the errors. In some cases, an error is displayed during execution, and in other cases, an error is not displayed but a result cannot be obtained as it is supposed to be. When an error is displayed during execution, its location and its type are shown. As a result, the cause can be easily found. However, when a correct result is not obtained without displaying any error, this is troublesome.

(1) Debugging with the error display.

The error display provides "error message" which indicates the following three items.


ERR2 P0-50
Line number where an error occurred
Program area where an error occurred
Error type

The error type is indicated by a code number that follows "ERR". The code number from 1 to 9 is used to indicate type; "ERR1" indicates "memory overflow", and "ERR2" indicates "syntax error". See the "Error Message Table" on page 166 for the meaning of these code numbers.
The program area and line number where the error occurred are also indicated.
Where and what kind of error occurred can therefore be determined by these three items.
Let's take a look at an example.
An error that often occurs is "ERR2" which is a syntax error. It occurs when a program is incorrectly stored.

For example, the program used in the previous example is incorrectly stored.

Operation	**Display**
MODE 1	P _123456789
SHIFT LIST 20 EXE	2Ø C=A+B_
← ← DEL	2Ø C=AB
EXE	3Ø C=A-B_
AC MODE Ø	READY PØ

In this example, "C=A+B" on line 20 was entered as "C=AB" by mistake. Now, execute the program.

Operation	**Display**
SHIFT P0	?
45 EXE	?
12 EXE	ERR2 PØ-2Ø

An error message is displayed; it indicates that a syntax error occurred on line 20 in program area PØ.
Check line 20.

Operation	**Display**
AC ······Error release	_
MODE 1	P _123456789
SHIFT LIST 20 EXE	2Ø C=AB_

Check if the line is correctly written. Since "+" between A and B was left out, correct this.

Operation	**Display**
←	2Ø C=AB
SHIFT INS	2Ø C=A_B
+ EXE	3Ø C=A-B
AC MODE Ø	READY PØ

Since "ERR2" is mostly caused by erroneous program input, when "ERR2" occurs, check the line whose number is indicated in the error message. When data is read-in by a READ statement (see page 90), if character data is read into a numerical variable, "ERR2" is also displayed. When a READ statement exists at an "ERR2" location, check the data in the DATA statement, too.
Check points for various errors are as follows.

- ERR1: Shortage of memory. Stack over.
  Confirm the number of remaining steps. Check if the memory has been mistakenly expanded beyond the capacity by a DEFM statement. Check if the calculation expressions are too complicated.
- ERR2: Syntax error
  Check if there are any errors in the stored program.
- ERR3: Mathematical error
  Check if the arithmetic result of a calculation expression is more than 10^{99}, or if the input range of a function is exceeded. When variables are used, check their contents.
- ERR4: Undefined line number
  The specification of a line number in a GOTO, GOSUB, or RESTORE statement is not correct. Confirm this line number.
- ERR5: Argument error
  Check the value of an argument or parameter for a command or function. When variables are used, check their contents.
- ERR6: Variable error
  When an array variable is used, check if the memory is expanded by a DEFM statement. Also, check if the same variable is used for both character variables and numerical variables at the same time.
- ERR7: Nesting error
  If the line where an error occurred is a RETURN statement or NEXT statement, check if the GOSUB statement or FOR statement correspondence is correct or not. Also, if the line where the error occurred is a GOSUB statement or FOR statement, check if there are more than 8 nesting levels for a GOSUB statement, and more than 4 for a FOR statement.
- ERR8: Password error
  When a password is specified, check if another password was entered, or if LIST, NEW, NEW ALL, etc. were used.

- ERR9: Option error
  Check if the FA-3 cassette interface or FP-12S (or FP-12) character printer are connected properly or not. Check if the rechargeable battery of the FP-12S is charged or the FP-12S is clogged with paper. Also, adjust the volume or tone of the tape recorder connected to the FA-3, clean the tape recorder head and replace the tape with a new one. Or operate the tape recorder by using only the white plug when recording is performed, and only the black plug during tape playback.

The commands mentioned above will be explained later. See "COMMAND REFERENCE" on page 113 and after for details.

(2) Although no error is displayed, the desired result cannot be obtained.

Since this often happens when a calculation expression or variable in a program was incorrectly used, check the operation of calculation expressions and variables. Especially when a correct result is not obtained, compare the original expression with the expression used in the program.
When the program execution does not stop or stops without work being accomplished, check the operation of the variable that controls the program flow. In regard to a calculation expression, check its location in the WRT mode (press MODE 1).
The program flow can be checked by stopping it with a STOP statement after entering data into the control variable, or by displaying the value of a variable with a PRINT statement.
Store the following program.

```
10 INPUT A
20 B=1
30 FOR C=1 TO A
40 B=B*C
50 C=C+1
60 NEXT C
70 PRINT B
80 END
```

This program obtains factorial of data entered by an INPUT statement. Actually line 50 is not required, and the variable used for FOR loop should not be changed.
The FOR-NEXT statements on lines 30 and 60 form a loop in which calculation is repeatedly performed. (These statements will be explained later.)

Operation	Display
MODE 1	P _ 123456789
SHIFT P1	P _ 123456789
NEW EXE └ Command to erase a program.	P _ 123456789
10 SHIFT INPUT A EXE	10 INPUT A
20 B=1 EXE	20 B=1
30 SHIFT FOR C=1 SHIFT TO A EXE	30 FOR C=1 T
40 B=B*C EXE	40 B=B*C
50 C=C+1 EXE	50 C=C+1
60 SHIFT NEXT C EXE	60 NEXT C
70 SHIFT PRINT B EXE	70 PRINT B
80 END EXE	80 END

Press AC MODE 0 to specify the RUN mode.

Operation	Display
Example) SHIFT P1	?
12 EXE	10395

The correct answer is 479001600.
Check the calculation expression. It has no mistake. Next, check the FOR-NEXT loop flow.
Insert a STOP statement after line 50 to stop the program each time.

Operation	Display
MODE 1	P _ 23456789
55 STOP EXE	55 STOP
AC MODE 0	READY P1

Since a STOP statement is to be inserted after line 50, place a line number between 50 and 60. As line number, 55 was selected.
Execute the program.

Operation	Display
SHIFT, P1	?
12 EXE	* STOP

Check the value of loop control variable C.

C EXE	2 STOP

Continue execution.

EXE	* STOP

Check the value of variable C again.

C EXE	4 STOP

Although the value of variable C must be increased by one each time, it is increased by 2. Therefore, it was found that the FOR-NEXT loop operation (flow) is not correct. Check the increment of variable C again. It was found that the problem is line 50 which is not required. Therefore, delete line 50 and the STOP statement that was added on line 55.

Operation	Display
MODE 1	P *123456789
50 EXE	_
55 EXE	_
AC MODE 0	READY P1

Debugging has been completed.
There is another way to debug besides using the STOP statement. It is debugging in the trace mode (Press MODE 2 . "TR" is displayed.) In regard to debugging in the trace mode, a program is executed and stopped after each command. Since the value of a variable, etc. is checked while the program is stopped, debugging is performed by pressing the EXE key to advance to the next command.
Try this with the previous example. The program area and line number are displayed each time the EXE key is pressed.
Press MODE 3 to release the trace mode; "TR" is erased.

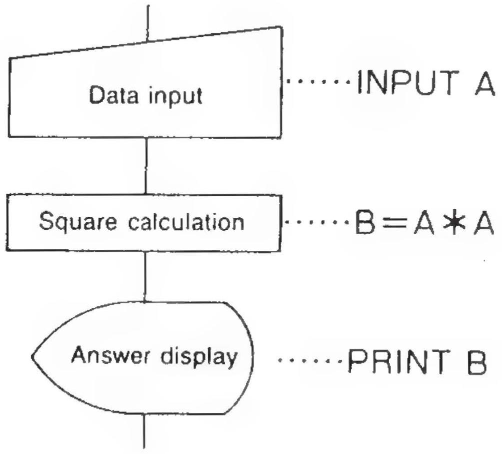
Since debugging can be performed as mentioned above, when an error is displayed or when the desired result cannot be obtained, don't be disappointed but try debugging.

3-3. PROGRAM DEVELOPMENT

It is certain that the outline of program has been understood by the previous explanation. The three parts of a program are input, calculation, and output. Many different programs can be prepared with these three items. However, a program can be more convenient and easier by using the commands explained in this section.

3-3-1 Changing The Program Flow (GOTO statements)

In addition to the three parts of a program, GOTO statements are very convenient when the same calculation must be repeated many times, or to transfer program flow to an arbitrary line instead of following line numbers sequence.
For example, let's prepare a program to obtain the square of a certain value. This program can be broken down in three parts which are "data input", "square calculation", and the "answer display". Make a flowchart.



Prepare the program in accordance with this flowchart.

```
10 INPUT A
20 B=A∗A
30 PRINT B
40 END
```

For example, square 15 and 43.

Operation	Display
RUN EXE	?
15 EXE	225
RUN EXE	?
45 EXE	1849

Since execution has to be performed each time as mentioned above, it is very inconvenient when lots of data exist.
Do you think it is very convenient that calculation can be repeatedly performed? It is the GOTO statement that makes this possible. The function of a GOTO statement is to transfer program flow to a line number or program area specified by the numerical value following GOTO. Replace the END statement on line 40 with a GOTO statement.

```
40 GOTO 10
```

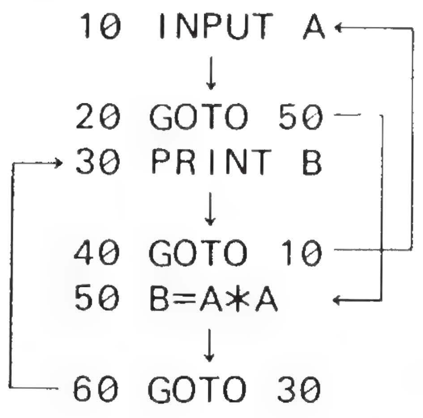
This means that the flow after line 40 is transferred (jumped) to line 10. Execute the modified program.

Operation	Display
RUN EXE	?
15 EXE	225
EXE	?
43 EXE	1849

A GOTO statement is convenient for the repetition of calculation as mentioned above.
Since a GOTO statement can cause a return to the beginning of a program to repeat execution, and can also cause a jump to an arbitrary location, there are many convenient ways to use it.
For example,

```
10 INPUT A
20 GOTO 50
30 PRINT B
40 GOTO 10
50 B=A*A
60 GOTO 30
```

The flow of this program is as follows.



```
10 INPUT A
20 GOTO 50
30 PRINT B
40 GOTO 10
50 B=A*A
60 GOTO 30
```

Since a GOTO statement unconditionally causes a jump to a specified line number as shown above, it is called an "unconditional jump."
A jump to a program area as well as to a line number can be performed by a GOTO statement. The program area is specified by adding "#" and a number from 0 to 9.

Example) GOTO #1 ······ Jumps to program area P1.
GOTO #9 ······ Jumps to program area P9.

When a jump is made to a program area, execution continues from the beginning of the program in this area.

NOTE

PRINT STATEMENTS

A PRINT statement is used for displaying the content of a variable, character string, or numerical value. Numerical variables and character variables can both be used.

Example) When A=123 ······PRINT A→123
When B$="ABC" ······PRINT B$→ABC

Since a character string placed inside " " (quotation marks) is displayed as it is, it can be used as a message.

Example) PRINT "CASIO" →CASIO

When two or more items are to be displayed, they can be written by punctuating them with commas (,) or semicolons (;).

Example)
```
PRINT A,B,Z$
PRINT "TOTAL=";T
```

Note that when a " , " is used, output is performed with line change; the execution stops after the first content is displayed ("STOP" appears). The following display is obtained by pressing the EXE key. However, when a " ; " is used, continuous display is performed.

Example) Try this by using the following program.

```
10 A=123
20 B$="ABC"
30 PRINT A,B$········ After displaying the content of A, the
                      content of B$ is displayed by press-
                      ing the EXE key.
40 PRINT A;B$········ The content of A and B$ are dis-
                      played continuously.
50 PRINT B$; ········ After displaying the content of B$,
                      the execution advances.
60 PRINT A   ········ After displaying the content of A, the
70 END                execution stops.
```

This program is executed as follows.

Operation	**Display**	
RUN EXE	` 123`	PRINT A,B$
EXE	`ABC`	
EXE	` 123ABC`	···PRINT A;B$
EXE	`ABC 123`	···PRINT B$; PRINT A

A one character space exists before the numerical value (123). This is the space for a sign (+ or −); since the positive sign is always omitted, a space is opened.

Line 20 uses an IF statement. If the condition is true, the item following THEN is executed. In this case, the program jumps to line 10.

The following relational operators are used for conditional expressions.

Left side > right side	...	Left side is larger than the right side.
Left side < right side	...	Left side is smaller than the right side.
Left side = right side	...	Left side is equal to the right side.
Left side ≧ right side	...	Left side is larger than or equal to the right side.
Left side ≦ right side	...	Left side is smaller than or equal to the right side.
Left side ≠ right side	...	Left side is not equal to the right side.

Since "THEN" includes the meaning of "GOTO", "THEN GOTO 10" can be written as "THEN 10".
Execute this program.

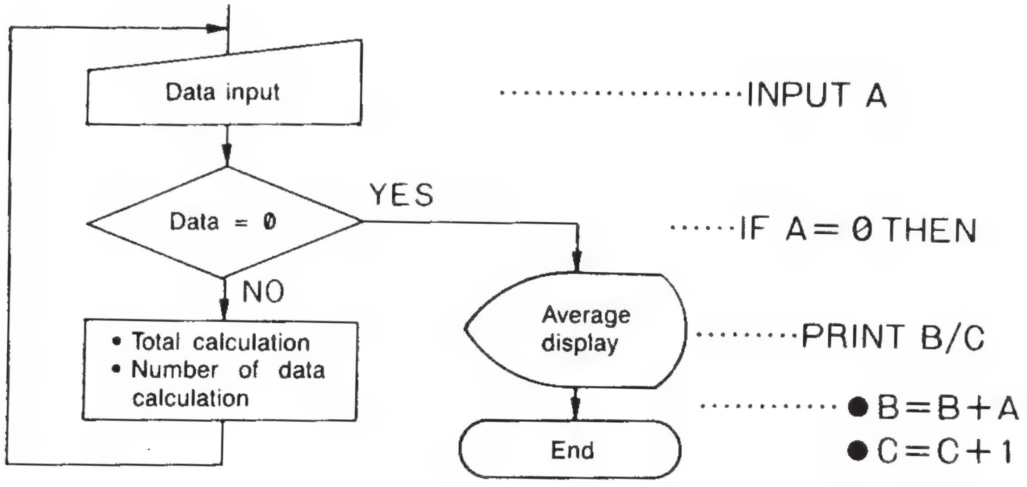
Operation	**Display**
RUN EXE	?
5 EXE	25
EXE	?
12 EXE	?
9 EXE	81

Data can be selected by an IF statement as mentioned above.

Example) When "0" is entered after entering several data, the average of these data is obtained.

This program can be divided into "Input", "Condition test", "Calculation" and "Display" parts. The "Calculation" part includes three procedures; obtaining the total, counting the number of items, and obtaining the average. Since the average calculation is only executed when "0" is entered, it will follow the "Condition test" part.

Prepare a flowchart based on this analysis.



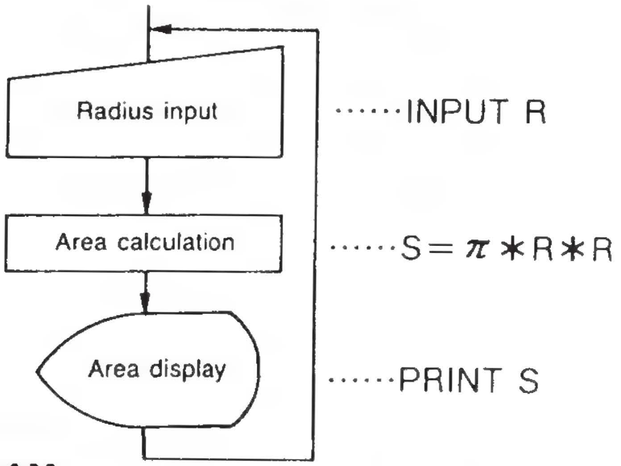
As can be seen in this flowchart, the IF statement checks if data entered to variable A is 0. If it is not 0, the total and number of data are obtained after which a return is made to data input status. If it is 0, the average is displayed and the execution terminates. Note that if the first input is 0, a division by zero causes an error.
This example is slightly difficult compared to the previous one. In regard to the calculation part, the input data are added to the variable for the total calculation. Since B is the variable for the total, the calculation is "B=B+A". (The content of variable A is added to the content of variable B.)
In regard to the number of data, when a data is entered, 1 is added to the counting variable (C in this case); "C=C+1" is realized.
In regard to the condition test part, which is the main part, if A is 0, the average is displayed by a PRINT statement. Since a statement can be written following THEN of an IF statement, "PRINT B/C" is written in this example; the result of the calculation expression (B/C) is displayed.

[EXERCISE]

Prepare a program to obtain the areas of circles by entering radiuses. Use a GOTO statement.

Expression: S= πr^2 (Press SHIFT π for Pi entry.)

The flowchart is as follows. S and R are used for variables according to the expression.



PROGRAM

```
10  INPUT R                 10  INPUT R
20  S=π∗R∗R        or      20  S=π∗R↑2      "↑2" is also used
30  PRINT S                 30  PRINT S      for square
40  GOTO 10                 40  GOTO 10      calculation.
```

3-3-2 Condition Test By A Program (IF-THEN statement)

If a size could be determined or control could be automatically performed in a program, it would be convenient.
An IF statement makes a test in a program; it makes a test on a conditional expression.

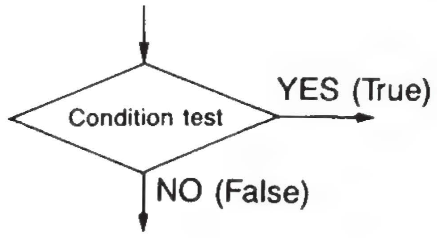
IF conditional expression THEN $\left\{\begin{array}{l}\text{Line No. or program area}\\ \text{Instruction statement}\end{array}\right\}$

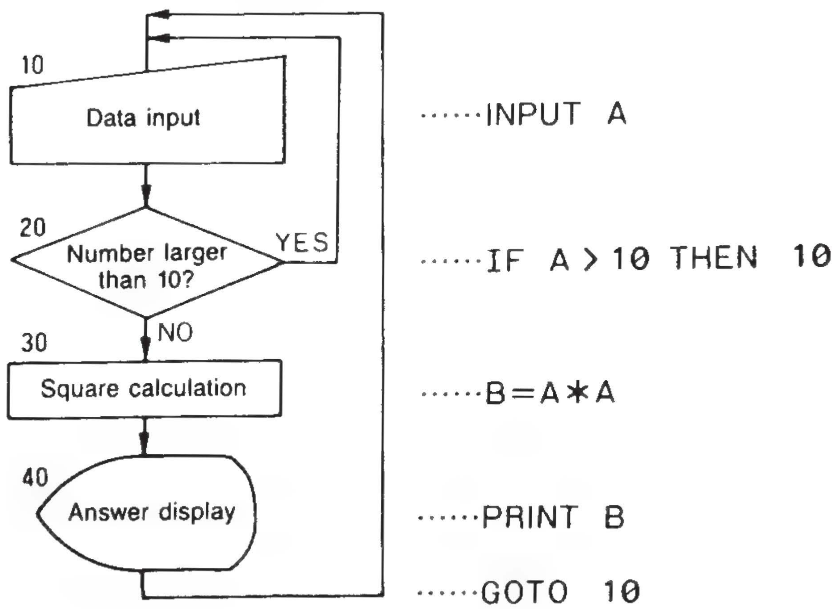
If the conditional expression is true, a jump is made to the line number or program area following "THEN", or the statement following "THEN" is executed. If the conditional expression is false, program execution advances to the next line.

Let's check the function of an IF statement.

Example) Enter an arbitrary number. If it is larger than 10, a return is made to the data input status. If it is 10 or smaller, its square is calculated and displayed then a return to the data input status.

This program consists of 4 parts (Input, Condition test, Calculation and Display). The following symbol is used for a condition test flowchart.





The numerals at the left of the flowchart are line numbers.

```
10 INPUT A
20 IF A>10 THEN 10
30 B=A*A
40 PRINT B
50 GOTO 10
```

In regard to variables B and C, if values were previously entered into them, they are continuously incremented and a wrong answer is obtained. As a result, zero must be entered into these variables ("B=0", "C=0"). Although separate line numbers can be used for these two assignment expressions, it will be easier to use a multistatement (written as "B=0:C=0" on one line with punctuation by a " : " (colon)).
Prepare the program.

```
10 B=0:C=0
20 INPUT A
30 IF A=0 THEN PRINT B/C
40 B=B+A
50 C=C+1
60 GOTO 20
```

Although the program input is completed, the program does not terminate after displaying the average. Therefore, add an END statement after the PRINT statement on line 30 by using a multistatement.

```
30 IF A=0 THEN PRINT B/C:END
```

Using an IF statement, a test is performed by the conditional expression as shown above.

• IF statement applications

In the above example, program progress was determined by one test. However, if there are several tests and all conditions must be satisfied, what is the solution?
For example, an arbitrary numerical value is to be entered and numerical values from 1 to 9 are to be selected. In other words, since the selected numerical values must be larger than 0 and smaller than 10, two conditions ("0 < variable" and "variable < 10") are required. This can be written on one line as follows.

IF 0 < variable THEN IF variable < 10 THEN .....

Although the same kind of statement can be used when there are three conditions or more, it is recommended that a maximum of two conditions be used since using more than two is too complicated and the line becomes too long.

NOTE

MULTISTATEMENT (:)

A multistatement is convenient when short assignment expressions are arranged on one line, or when there are several commands after THEN in an IF statement.

Example 1)

```
10 A=1 }
20 B=2 }  ⇒  10 A=1:B=2:C=3
30 C=3 }           :
   :               :
   :
```

Example 2)

```
50 C=A+B }
60 D=A-B }
70 E=A*B }  ⇒  50 C=A+B:D=A-B:E=A*B:F=A/B
80 F=A/B }           :
   :                 :
   :
```

When a multistatement is used after THEN in an IF statement, it is executed only when a condition expression is true. Therefore, precautions shall be taken.

Example 3)

```
IF A<0 THEN A=10:B=20:......
            ------------------
                    |
              Executed only when
              condition is true.
```

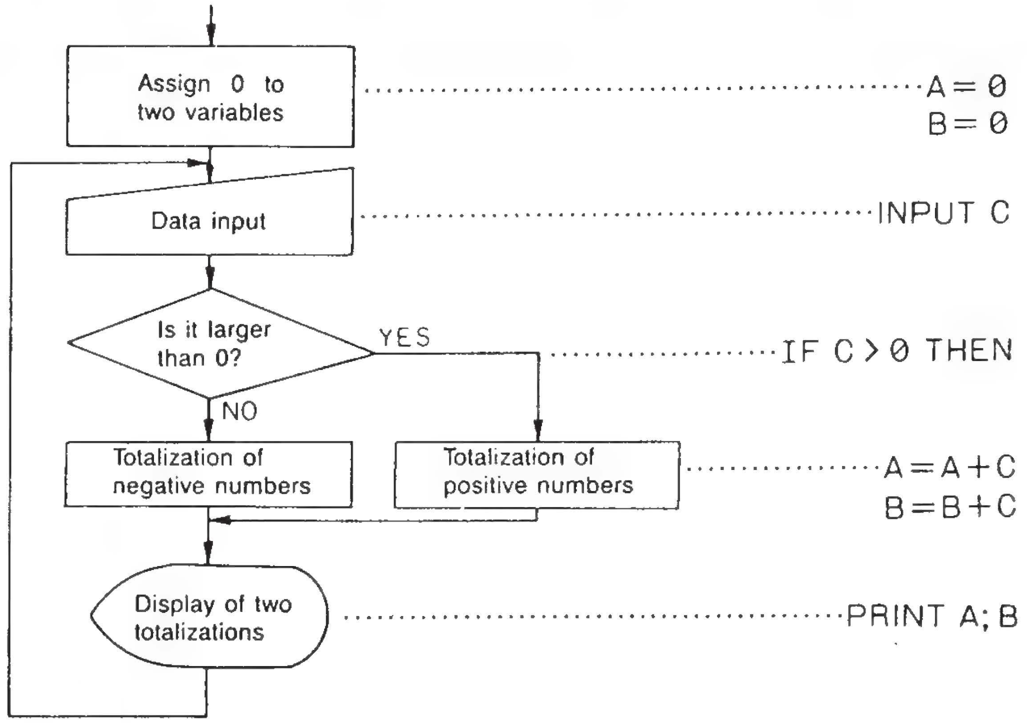
A multistatement is convenient when arranging a program. However, since one line is not easy to see if it is too long, make one line with an appropriate length, and write the remaining items on the next line.

[EXERCISE]

Prepare a program to separate entered numerical values into two groups (larger or smaller than 0) and obtain each total.

<HINT>

After a numerical value is input, an IF statement determines which variable is used for totalization.
0 (zero) must previously be assigned to the variables for totalization.


Assign 0 to two variables
A=0
B=0
Data input
INPUT C
Is it larger than 0?
YES
NO
IF C>0 THEN
Totalization of negative numbers
Totalization of positive numbers
A=A+C
B=B+C
Display of two totalizations
PRINT A;B

PROGRAM

```
10 A=0:B=0
20 INPUT C
30 IF C>0 THEN A=A+C:GOTO 50
40 B=B+C
50 PRINT A;B
60 GOTO 20
```

The IF statement on line 30 determines whether or not the input value (value of variable C) is larger than 0. If it is larger than 0, it is added to variable A after THEN, and if it is not larger than 0, it is added to variable B on line 40. On line 50, each total is displayed every time a value is entered.

3-3-3 Repeating A Program (FOR-NEXT statement)

If combined GOTO and IF statements are repeated a certain number of times, the program becomes too long and too complicated. When the number of repetitions is known, the program can be arranged in a more simple way. The command that performs this repetition is the FOR-NEXT statement. In the FOR-NEXT statement, the calculation between the FOR statement and the NEXT statement is repeated a specified number of times.
The format of a FOR statement is as follows.

FOR control variable = initial value TO final value STEP increment

The format of a NEXT statement is as follows.

NEXT control variable

A numerical variable with only one character, such as A or B, can be used as a control variable in a FOR statement, but an array variable cannot be used. The initial value, final value, and the increment can be a numerical expression or numerical variable. The control variable is repeatedly changed, by the increment, from an initial value to a final value. The increment can be omitted and becomes 1 when omitted.
Store and execute the following program to easily understand the function of a FOR-NEXT statement.

```
10 INPUT A
20 FOR B=1 TO 10 STEP A
30 PRINT B
40 NEXT B
50 GOTO 10
```

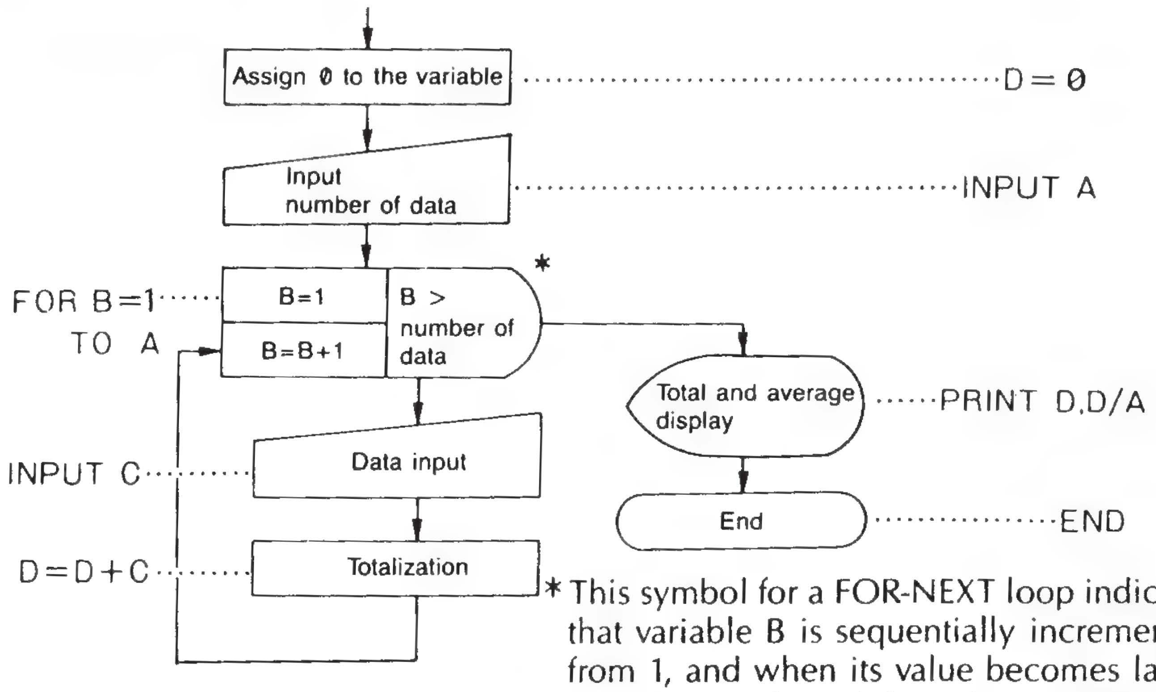
This program is executed as follows.

Operation	**Display**
RUN EXE	?
3 EXE	1
EXE	4
EXE	7
EXE	10
EXE	?
0.8 EXE	1
EXE	1.8
EXE	2.6
EXE	3.4
⋮	⋮

Example) Prepare a program to obtain a total and an average for data when a certain number of data is entered.

For this program, enter the number of data first, then enter each data by a FOR-NEXT statement and obtain the total. After the data input has been terminated, the total and average are displayed.

Prepare the flowchart.



* This symbol for a FOR-NEXT loop indicates that variable B is sequentially incremented from 1, and when its value becomes larger than the number of data, the program execution leaves the FOR-NEXT loop to make a transfer to the next work.

Prepare a program based on this flowchart.

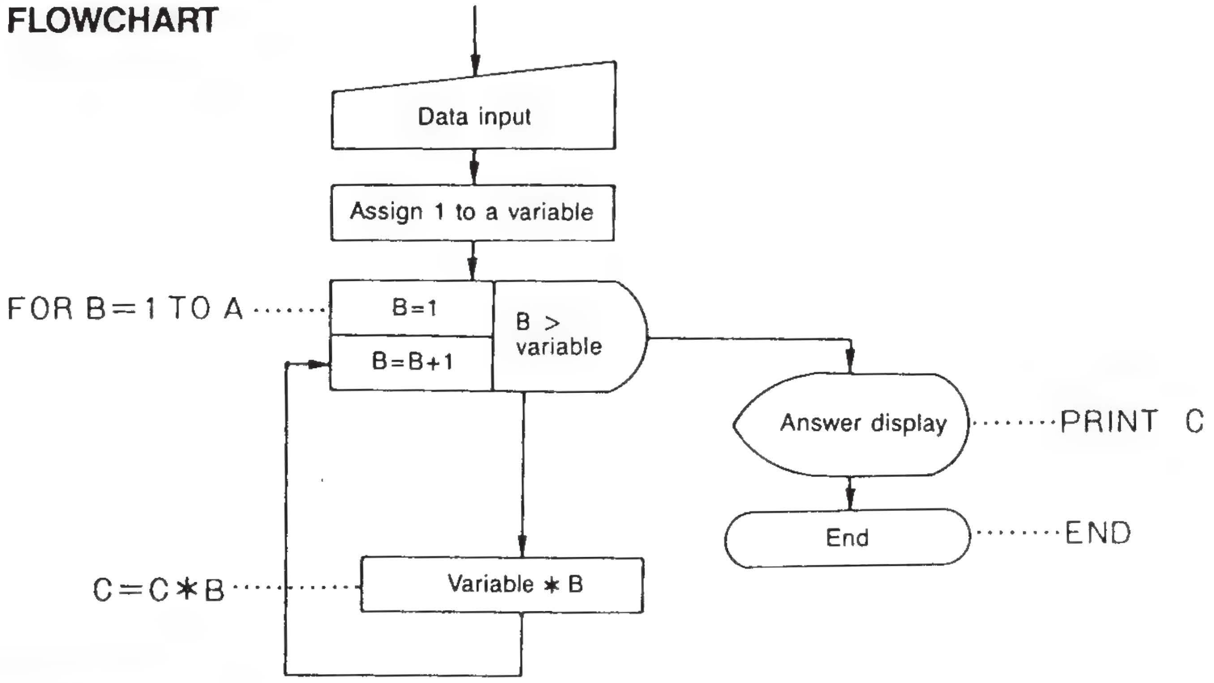
```
10 D=0
20 INPUT A
30 FOR B=1 TO A
40 INPUT C
50 D=D+C
60 NEXT B
70 PRINT D,D/A
80 END
```

When the number of data is known as shown above, data input and calculation can be repeated by a FOR-NEXT statement. The number of data can be directly written instead of A on line 30 without entering the number of data by an INPUT statement as shown on line 20. In this case, line 20 is not required.

[EXERCISE]
Prepare a program that calculates factorial.

<HINT>
Perform a factorial calculation (e.g., 5! = 1 × 2 × 3 × 4 × 5) by using a FOR-NEXT loop.

FLOWCHART



PROGRAM

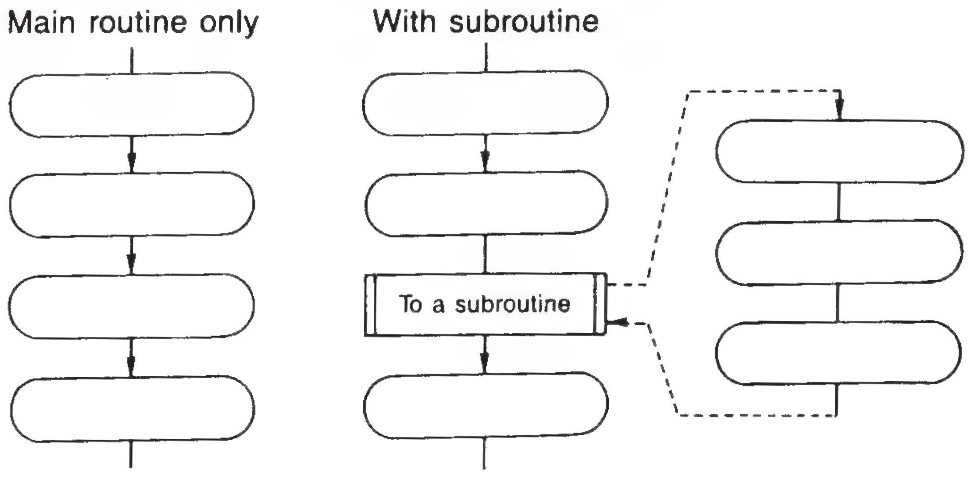
```
10 INPUT A
20 C=1
30 FOR B=1 TO A
40 C=C*B
50 NEXT B
60 PRINT C
70 END
```

On line 20, 1 is assigned to variable C, that obtains the factorial, because a wrong answer is obtained if this variable does not have 1 for initial value. Factorial calculation is performed by the FOR-NEXT statement (from lines 30 to 50) in which the value of variable B is incremented by 1; the factorial is obtained by calculating 1 ×2 × 3 × ... . The program is terminated by the END statement on line 70 after one calculation. However, if many factorial calculations are to be performed sequentially, line 70 should be "GOTO 10".

3-3-4 A Convenient Subroutine For Complicated Programs (GOSUB-RETURN statement)

You are now accustomed to preparing programs; they can become long and complicated. A program, which is convenient for performing repetitive processing and is especially helpful when composing lengthy programs, is called a "subroutine".

A subroutine recalled by a main routine performs part of the work.



Check its function by using an example.

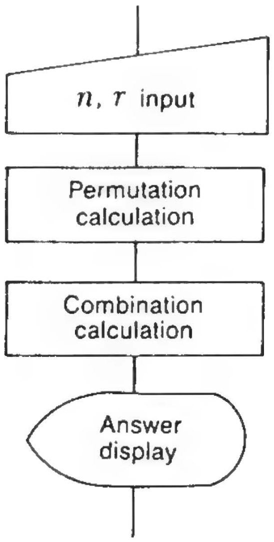
Example) Prepare a program to obtain permutations and combinations.

Expression: Permutations $_nP_r=\frac{n!}{(n-r)!}$

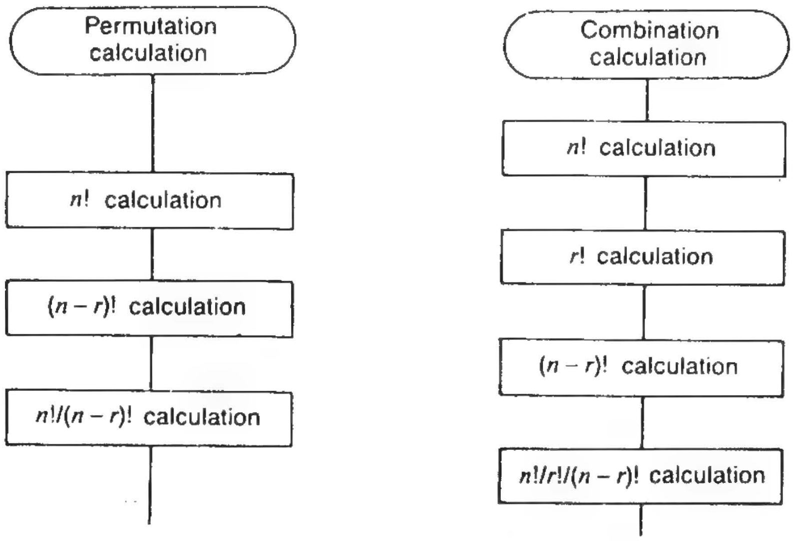
Combinations $_nC_r=\frac{n!}{r!(n-r)!}$

This program obtains permutations and combinations of two entered data items (*n*, *r*).

Prepare a simple flowchart.



Now prepare detailed flowcharts for the calculation of permutations and combinations.



The calculations of *n*! and (*n* − *r*)! are used in both of permutation and combination calculations. Also, three different factorial calculations are performed.

PROGRAM EXAMPLE (1)

```
 10 INPUT N,R          }
 20 A=1                }
 30 FOR B=1 TO N       }
 40 A=A*B              } n! calculation
 50 NEXT B             }
 60 E=A                }
 70 A=1                }
 80 FOR B=1 TO N-R     }
 90 A=A*B              } (n-r)! calculation
100 F=A                }
110 NEXT B             }
120 P=E/F       ....... n!/(n-r)! calculation
130 A=1                }
140 FOR B=1 TO R       }
150 A=A*B              } r! calculation
160 NEXT B             }
170 G=A                }
180 C=E/G/F     ........ n!/(r!(n-r)!) calculation
190 PRINT P,C
200 END
```

VARIABLE CONTENTS

N	: n
R	: r
P	: Permutation
C	: Combination
A	: For factorial
B	: For FOR-NEXT loop
E	: $n!$
F	: $(n-r)!$
G	: $r!$

In this program, three factorial calculations utilize a common program except for the final value in the FOR statements. If this final value can be controlled by a common variable, these three calculations can be made in common. Here using a subroutine is very effective.
Use the variable H for the final values in order to make these calculations common. It is necessary to sequentially enter the values of *n*, *n* − *r*, and *r* into variable H.
Now this program is changed and includes a subroutine system; a command that transfers the work to the subroutine, and a command to provide a return after work termination are necessary. These commands are "GOSUB" and "RETURN".

PROGRAM EXAMPLE (2)

```
 10 INPUT N,R
 20 H=N
 30 GOSUB 150
 40 E=A
 50 H=N-R
 60 GOSUB 150
 70 F=A
 80 P=E/F
 90 H=R
100 GOSUB 150
110 G=A
120 C=E/G/F
130 PRINT P,C
140 END
150 A=1              )
160 FOR B=1 TO H     |
170 A=A*B            } Subroutine
180 NEXT B           |
190 RETURN           )
```

A program that has a common part can be simply prepared by using a subroutine as shown above.
Although this program is shorter by only one line, it has an important function when a program is more complicated or longer.

[EXERCISE]

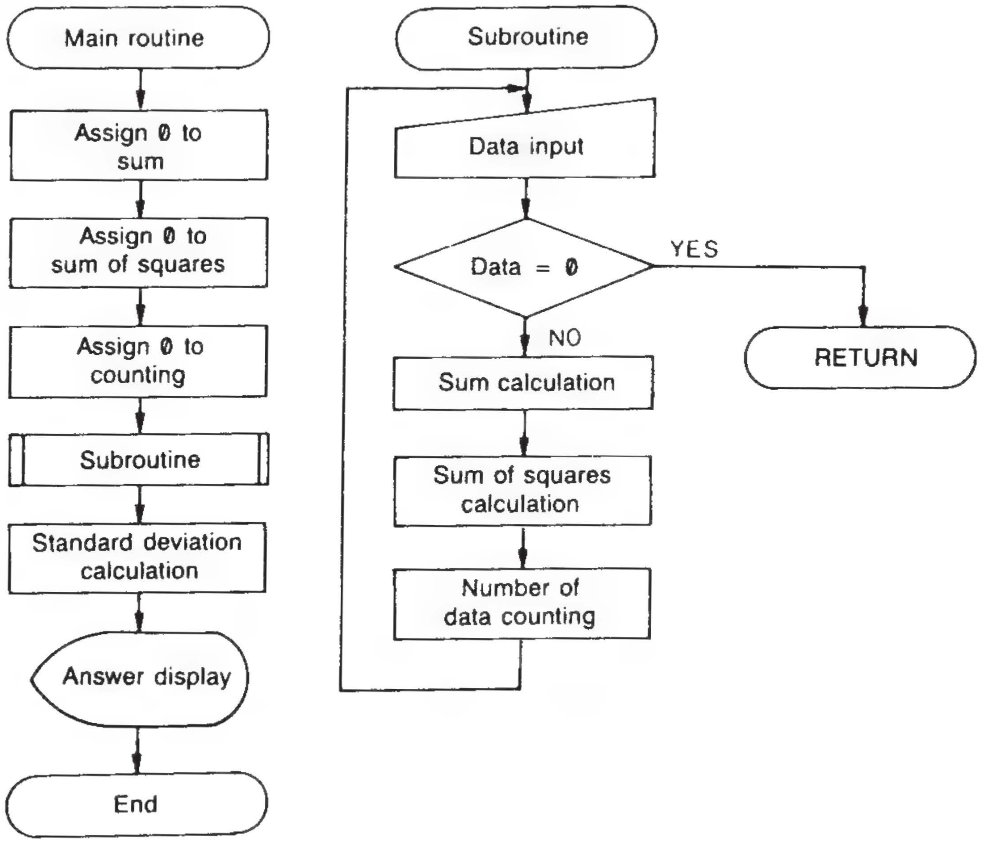
Prepare a program to obtain standard deviation. The data input, sum calculation, sum of squares calculation, and the number of data counting are included in a subroutine.

Expression:

$$\sigma n = \sqrt{\frac{\Sigma x^2 - (\Sigma x)^2/n}{n}}$$

Σx	Sum
Σx^2	Sum of squares
n	Number of data

< HINT >
After data input, if "0" is detected by an IF statement, a return is made to the main routine to obtain the standard deviation. SQR is used for square root.



PROGRAM

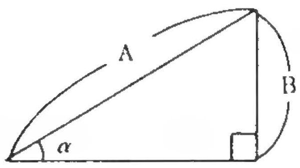
```
 10 B=0:C=0:D=0
 20 GOSUB 100                ······ To the subroutine
 30 E=SQR((C-B*B/D)/D)······ Standard deviation
 40 PRINT E
 50 END
100 INPUT A                                        }
110 IF A=0 THEN RETURN                             }
120 B=B+A               ··············· Sum        }
130 C=C+A*A             ··············· Sum of squares } Subroutine
140 D=D+1               ··············· Number of data }
150 GOTO 100                                       }
```

In this program, execution is transferred to the subroutine by line 20; data input, sum calculation, sum of squares calculation, and the number of data counting are performed from line 100 of this subroutine.
The IF statement on line 110 is the condition test for data input termination. If 0 is entered, the execution advances to the statement following "THEN" and returns to the main routine.
Also, be sure to add END at the end of the main routine as shown on line 50.

3-3-5 Using Functions

Functions can be used in a program. Some program examples are explained below.

Example 1) Trigonometric function



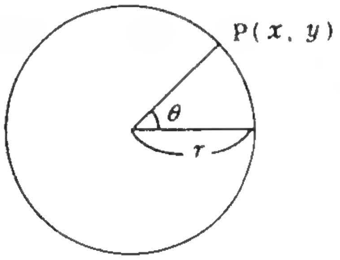
Obtain the length of side B of the triangle on the left by entering side A and angle α.

Expression: $B = A \cdot \sin\alpha$
Angle unit: Degree

PROGRAM EXAMPLE	Operation example	Display
10 MODE 4	RUN EXE	?
20 INPUT A,D	(Side A) 15 EXE	?
30 B=A∗SIN D	(Angle α) 30 EXE	7.5
40 PRINT B		
50 END		

The angle unit is specified on line 10. Since the calculation is performed in degree, "MODE 4" is specified. The trigonometric function is used to perform calculation on line 30.

Example 2) Trigonometric function



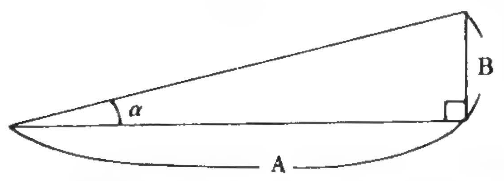
Obtain the coordinates (x, y) of point P of the circle on the left by entering the radius r and angle θ.

Expressions: $x = r \cdot \cos\theta$
$y = r \cdot \sin\theta$
Angle unit: Radian

PROGRAM EXAMPLE	Operation example	Display
`10 MODE 5`	RUN EXE	?
`20 INPUT R,T`	(Radius) 5 EXE	?
`30 X=R*COS T`	(Angle θ) π/3 EXE	2.5
`40 Y=R*SIN T`	EXE	4.330127019
`50 PRINT X,Y`		
`60 END`		

Since the angle unit is radian, "MODE 5" is specified on line 10. The *x*-coordinate and *y*-coordinate are obtained on lines 30 and 40.

Example 3) Inverse trigonometric function



Obtain angle α of the triangle on the left by entering side A and side B.

Expression: $\alpha = \tan^{-1}\frac{B}{A}$

Angle unit: Degree

PROGRAM EXAMPLE	Operation example	Display
`10 MODE 4`	RUN EXE	?
`20 INPUT A,B`	(Side A) 100 EXE	?
`30 D=ATN(B/A)`	20 EXE	11.30993247
`40 PRINT D`		
`50 END`		

Example 4) Decimal ↔ Sexagesimal conversion

Prepare a program that performs time calculation.

PROGRAM EXAMPLE	Operation example	Display
`10 T=0`	RUN EXE	?
`20 INPUT D,E,G`	(Hour) 1 EXE	?
`30 S=SGN D`	(Minute) 25 EXE	?
`40 D=ABS D`	(Second) 36 EXE	1°25'36
`50 T=T+S*DEG(D,E,G)`	EXE	?
`60 PRINT DMS$(T)`	(Hour) 2 EXE	?
`70 GOTO 20`	(Minute) 15 EXE	?
	(Second) 5 EXE	3°40'41

On line 20, enter the hour, minute, and second into 3 variables, D, E, and G respectively.
Lines 30 and 40 are used for subtraction. If the hour is entered with a negative sign (−), the entered time is subtracted from the previous result. If only addition is performed, these lines are not necessary.
The total is obtained on line 50. 1 is entered into variable S to perform addition, and −1 to perform subtraction. The DEG function converts sexagesimal (hours, minutes and seconds) to decimal, and totalization is performed in decimal.
Line 60 for the display uses the DMS$ function that converts decimal to sexagesimal.

Example 5) Random number generation
Generate random numbers from 1 to 99.

PROGRAM EXAMPLE
```
10 R=INT(RAN#*99)+1
20 PRINT R
30 GOTO 10
```

A random number is generated on line 10. The number being from 1 to 99 in this example, multiply the generated random number by 99 and add 1 to the result to obtain a number from 1 to 99 (see page 150).

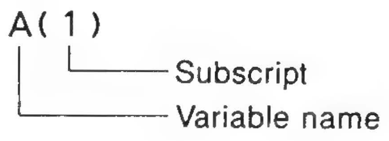
When the random number is from 5 to 9 → `INT(RAN#*5)+5`

When the random number is from 10 to 20 → `INT(RAN#*11)+10`

3-3-6 Using An Array

Being different from general variables from A to Z, an array variable is managed by a number (subscript).
An alphabetical character from A to Z can be used for the variable name with a subscript attached.

Example)



An array is convenient when a large amount of data is handled.
For example, to enter 10 data items:

```
10 FOR A=1 TO 10
20 INPUT N(A)
30 NEXT A
```

In this case, the array is arranged as follows.

General variables	O	P	Q	R	S	T	U	V	W	X
Array variables	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)

To select the largest data item from among 10:

```
100 A=0
110 FOR B=1 TO 10
120 IF N(B)>A THEN A=N(B)
130 NEXT B
140 PRINT A
```

When an array is used, precautions should be taken concerning its arrangement. Some array containers are commonly utilized as those for general variables. For example, the container for A(0) is the same as that for A, and A(1) is the same as B. Therefore if A(0) and A are used in the same program, data in the variable is changed.
The common parts are as follows.

```
   A      B      C      D            X      Y      Z
 A(0)   A(1)   A(2)   A(3)..........A(23)  A(24)  A(25)
        B(0)   B(1)   B(2)..........B(22)  B(23)  B(24)
                 :                           :
                 :                           :
                                    X(0)   X(1)   X(2)
                                           Y(0)   Y(1)
                                                  Z(0)
```

* For details, see page 170.

For example, perform the following operations.

Assign 7 to variable I.

`I=7` EXE	`_`

Confirm the content of variable I.

`I` EXE	`7`

Assign 10 to the 9th container of array variable A.

`A(8)=10` EXE	`_`

Confirm the content of variable I.

`I` EXE	`10`

The content of variable I has been changed as shown above. It is because the container for variable I is the same as that for array variable A(8). When an array is used in a program, keep variables for FOR-NEXT loop or assignment, and then determine the variable names of this array.

Example 1) When 10 array variables are used,
Array variables: A(0) — A(9)
Unused general variables: K — Z

Example 2) When 50 array variables and 15 general variables are used,
General variables: A — O
Array variables: P(0) — P(49) [Operate DEFM 39 EXE .]

Example 3) When variable A is used for FOR-NEXT loop, variable B for totalling and 50 array variables are used,

```
10 DEFM 26              ·········Increase variables by 26 to obtain
20 B=0                           52 variables.
30 FOR A=0 TO 49 )
40  INPUT C(A)    } ······Array variables C(0) to C(49) can be used.
50 B=B+C(A)       |
60 NEXT A         )
```

When a large amount of data is handled by an array, precautions shall be taken so that these array variables do not overlap other general variables.

Also, precautions shall be taken on memory expansion when an array is used. When the general variables A to Z are sufficient for the size of the array, expansion is not required. However, when these variables are not sufficient, be sure to expand the memory.
Memory expansion is performed by the DEFM command which specifies the amount of expansion.
For example, when an expansion of 10 variables is performed,

Manually: `DEFM 10` EXE
In a program: Line number `DEFM 10`

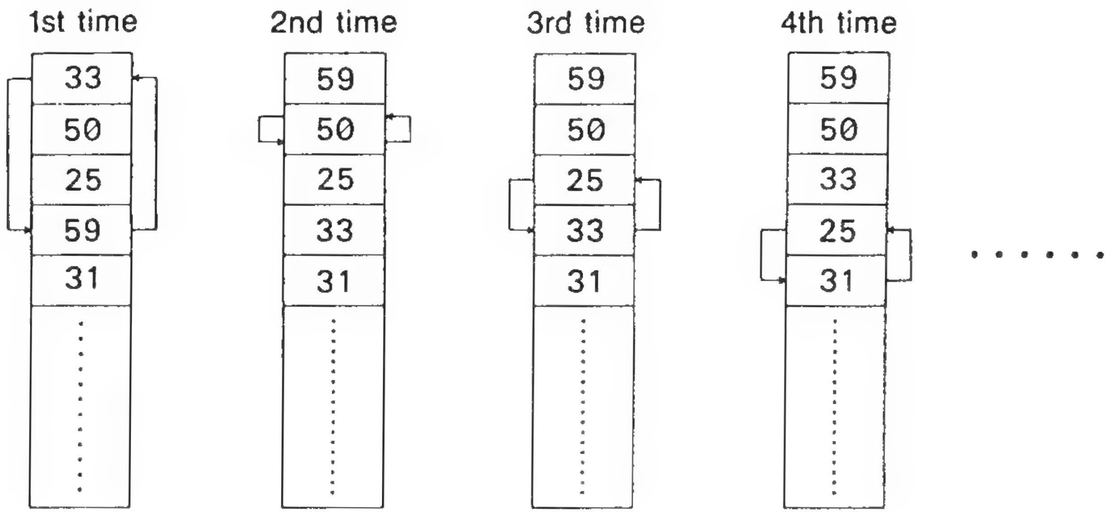
This command can be used manually or in a program. Write it at the beginning of a program when an array is used. Now, analyze a program that uses an array.

Example) Assign random numbers from 0 to 59 to array variables from G(0) to G(59) and arrange them in the descending order for display.

PROGRAM EXAMPLE

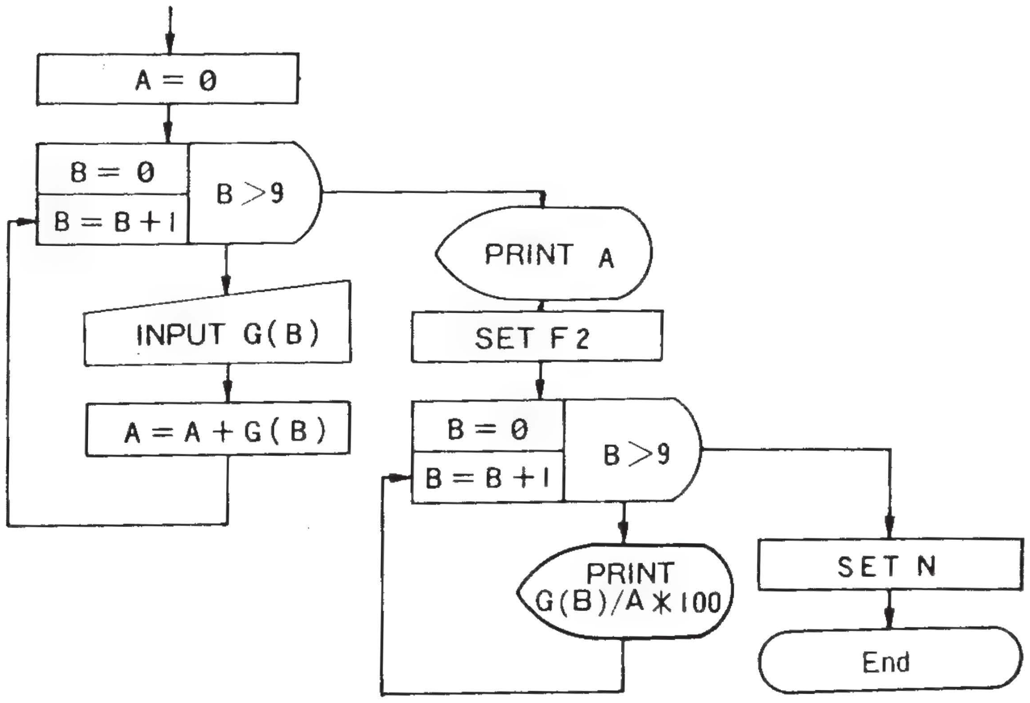
```
 10 DEFM 40
 20 FOR B=0 TO 59
 30 G(B)=INT(RAN#*60)              Assign random numbers
 40 NEXT B                         0—59 to G(0)—G(59).
 60 FOR B=0 TO 59
 70 A=-1
 80 FOR C=B TO 59
 90 IF G(C)>A THEN A=G(C):D=C      Arrange them in the
100 NEXT C                         descending order.
110 E=G(B):G(B)=G(D):G(D)=E
120 NEXT B
140 FOR B=0 TO 59
150 PRINT G(B)                     Display them sequentially.
160 NEXT B
170 END
```

This program consists of three parts. In the first part, numerical values from 0 to 59 are generated as random numbers which are assigned to array variables G(0) to G(59). In the second part, they are arranged in the descending order. In the third part, they are displayed in this order.
Lines 60 to 120 are repeatedly executed to arrange data in the descending order.



The DEFM statement on line 10 is used to expand the memory. Since 20 variables (G to Z) are remaining and 60 variables (G(0) to G(59)) are necessary, memory expansion for 40 of them is required.
An array is convenient when a large amount of data is handled.

[EXERCISE]
Enter 10 data items and obtain the component ratio for each of them. Obtain this ratio (percentage) with up to 2 decimal places.



PROGRAM

```
 10 A=0
 20 FOR B=0 TO 9
 30 INPUT G(B)
 40 A=A+G(B)
 50 NEXT B
 60 PRINT A
 70 SET F2
 80 FOR B=0 TO 9
 90 PRINT G(B)/A*100
100 NEXT B
110 SET N
120 END
```

Lines 20 to 50 are used to enter 10 data items into G(0) — G(9) by a FOR-NEXT statement. "SET F2" on line 70 specifies two decimal places for the component ratio. This ratio is displayed by lines 80 to 100. The specification of the decimal places is released on line 110.

3-3-7 Data Read-in (READ, DATA, RESTORE statements)

When an INPUT statement (data input method) is used, "?" is displayed during program execution to ask data input from the keyboard. A READ statement reads data written in a DATA statement and assign them to variables.

Example) Read 10 data items in a DATA statement and display them.

```
PROGRAM                            Operation
10 FOR B=1 TO 10                   RUN EXE    1
20 READ A                              EXE    2
30 PRINT A                             EXE    3
40 NEXT B                               :     :
50 DATA 1,2,3,4,5,                      :     :
   6,7,8,9,10                           :     :
60 END                                 EXE    10
```

A READ statement must be used together with a DATA statement since a data item fetched from a DATA statement is assigned to a variable following "READ".
Several variables can be written after a READ statement by punctuating them with commas (,).

Example)

```
PROGRAM                        Operation        Display
10 READ A$,B$                  RUN EXE          CASIOPB&FX
20 PRINT A$;B$
30 END
40 DATA CASIO,PB&FX
```

A DATA statement can be placed anywhere in a program. After program execution, data are assigned to variables sequentially from the first data of the DATA statement that has the smallest line number. When data written in a DATA statement are numerical values, use numerical variables for the variables in the READ statement, and when they are characters, use character variables.

Example)

PROGRAM

```
 10 RESTORE
 20 READ A$
 30 PRINT A$
 40 READ B
 50 PRINT B
 60 RESTORE 90
 70 READ C$
 80 PRINT C$
 90 DATA ABC
100 DATA 123456
110 END
```

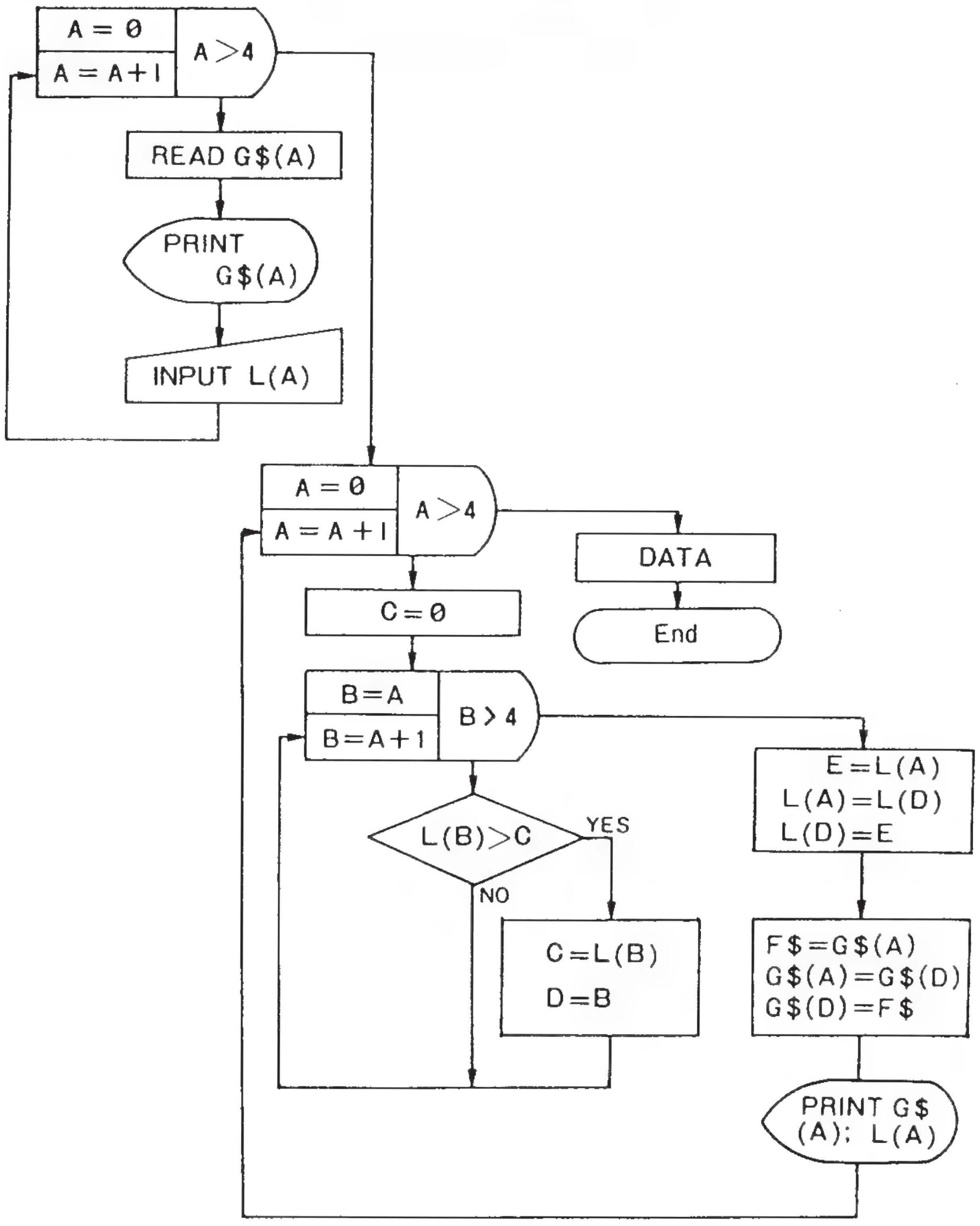
Operation	**Display**
RUN EXE	ABC
EXE	123456
EXE	ABC

When no line number is specified by a RESTORE statement, the first data in the first DATA statement in the program is read by the next READ statement. When a line number is specified, the first data in the specified DATA statement is read by the next READ statement.

[EXERCISE]

Read-in names CHICAGO, LONDON, PARIS, ROME, and TOKYO, and arrange them in the descending order of their related data.
Note that the names are assigned to G$(0)—G$(4) while data are entered into L(0)—L(4).

Flowchart example



PROGRAM EXAMPLE

```
 10 FOR A=0 TO 4
 20 READ G$(A)
 30 PRINT G$(A);
 40 INPUT L(A)
 50 NEXT A
 60 FOR A=0 TO 4
 70 C=0
 80 FOR B=A TO 4
 90 IF L(B)>C THEN C=L(B):
    D=B
100 NEXT B
110 E=L(A):L(A)=L(D):L(D)
    =E
120 F$=G$(A):G$(A)=G$(D):
    G$(D)=F$
130 PRINT G$(A);L(A)
140 NEXT A
150 DATA CHICAGO, LONDON, PARIS, ROME, TOKYO
```

MEMORY

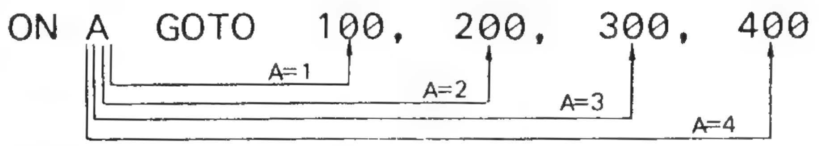
A	: FOR-NEXT statement use
B	: FOR-NEXT statement use
C	: Maximum value
D	: Number of maximum value
E	: Arrangement use
F$	: Arrangement use
G$(0)—G$(4)	: Names
L(0)—L(4)	: Data

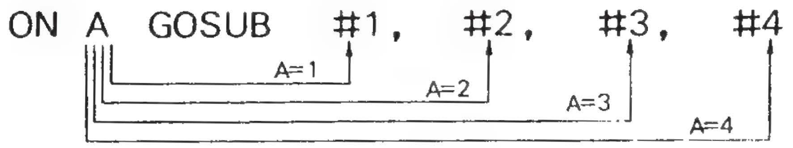
This program is divided into two parts which are an input part on lines 10 to 50, and an arrangement part on lines 60 to 140. In the input part, names are read by using a DATA statement while a loop is performed 5 times by a FOR-NEXT statement, and data are also entered at the same time. The PRINT statement on line 30 displays the name as a message before data is entered by the following INPUT statement. The names and data are both simultaneously arranged on lines 110 and 120.
The DATA statement on line 150 can be placed anywhere in the program.

3-3-8 Indirect Specification (ON-GOTO, ON-GOSUB statements)

Although GOTO statement and GOSUB statement specifications are performed by writing the line number or program area directly in a program, sometimes the branching depends on the arithmetic result or the data for processing convenience. In this case, testing the condition with an IF statement is not convenient.
The commands that determine and specify a branching in a program are indirect specifications (ON-GOTO and ON-GOSUB).
The function of an ON-GOTO statement is similar to that of an ON-GOSUB statement.

Example)





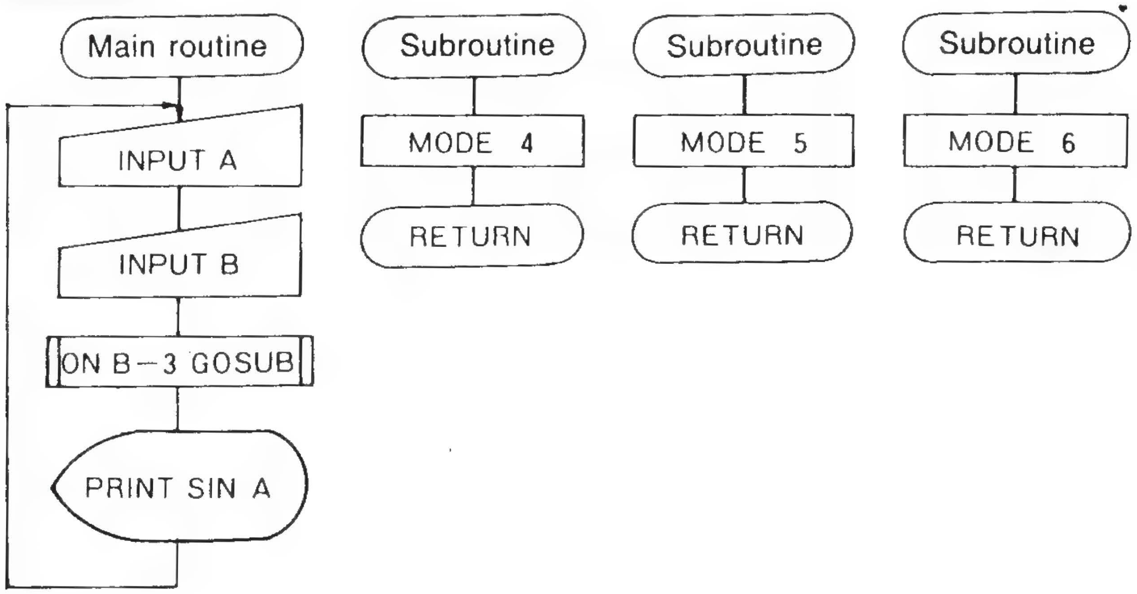
Branching is performed to the 1st location if the value of A is 1, and to the 2nd location if this value is 2, etc. It depends on the numerical variable or the result of the calculation expression that follows "ON". When the number of branching locations is less than the value of the variable or calculation expression, or when a branching location does not exist, program execution advances to the next line, or command in case of a multistatement.

Example)

```
 10 INPUT A
 20 ON A GOTO 100,200,300,400,500
 30 PRINT " NO"
 40 END
100 PRINT "LINE 100" :END
200 PRINT "LINE 200" :END
300 PRINT "LINE 300" :END
400 PRINT "LINE 400" :END
500 PRINT "LINE 500" :END
```

[EXERCISE]
Enter an angle and a numerical value from 4 to 6, branch to the subroutine that specifies the angle unit by an indirect specification, and obtain the sine of this angle.

Flowchart



PROGRAM

```
 10 INPUT "ANGLE",A
 20 INPUT "UNIT",B
 30 ON B-3 GOSUB 100,200,300
 40 PRINT SIN A
 50 GOTO 10
100 MODE 4
110 RETURN
200 MODE 5
210 RETURN
300 MODE 6
310 RETURN
```

Since two data items are entered, a message is added to each input statement so that input can be easily performed. On line 10, the angle is entered into variable A, and on line 20, either 4, 5 or 6 is entered into variable B to specify the angle unit (See page 139). On line 30, the branching location is determined by converting the 4—6 numerical value to 1—3 by using ON-GOSUB.
Each subroutine is used to specify an angle unit.

3-3-9 Character Handling Functions (LEN, MID$, VAL, STR$)

While SIN and COS are called numerical functions because they handle numerical values, there are character functions that handle characters. This computer is provided with "LEN", "MID$", "VAL" and "STR$" character functions.

• LEN

The LEN function counts the number of characters in a character variable.

Example) **Display**

```
A SHIFT $ = SHIFT " A B C SHIFT " EXE
L E N SHIFT ( A SHIFT $ SHIFT ) EXE
```

```
_
3
```

* An array variable cannot be used with the LEN function.

• MID$

The MID$ function fetches characters from among those stored in the exclusive character variable ($) by specifying the starting location and the number of characters to be fetched.

Example)

```
10 $="CASIO PB&FX"        Operation     Display
20 PRINT $                RUN EXE       CASIO PB&FX
30 PRINT MID$(1,5)            EXE       CASIO
40 PRINT MID$(7,5)            EXE       PB&FX
50 END
```

• VAL

The VAL function converts numerals stored in a character variable to a numerical value.

Example)

```
10 A$="123":B$="456"      Operation     Display
20 PRINT A$+B$            RUN EXE       123456
30 PRINT VAL(A$)              EXE        579
   +VAL(B$)
40 END
```

* An array variable cannot be used with the VAL function.

• STR$

The STR$ function converts numerical values stored in a numerical variable to a character string; this is the inverse of the VAL function.

Example)

```
10 A=123:B=456
20 PRINT A+B
30 PRINT STR$(A)+STR$(B)
40 END
```

Operation	Display
RUN EXE	579
EXE	123456

Example)

```
10 INPUT $
20 FOR A=1 TO LEN($)
30 PRINT MID$(A,1);
50 NEXT A
60 END
```

This program fetches a character entered in the exclusive character variable ($) with the MID$ function. One character is fetched each time. The starting location is specified by a FOR-NEXT statement,and the final value is determined by the LEN function.
" ; " is added at the end of line 30 so that the program does not stop because a continuous display is desired.

Example)

```
 10 A=1:B=0
 20 PRINT "<";STR$(A);">";
 30 INPUT $
 40 IF $="END" THEN 100
 50 B=B+VAL($)
 60 A=A+1
 70 GOTO 20
100 PRINT B/(A-1)
110 END
```

This program obtains the average for an unknown number of data. Data input is terminated by entering END, and the average is displayed by branching to line 100.
Line 20 provides a message that enables easier input.
On line 50, since data is entered into the exclusive character variable ($) as a character, totalization is performed after converting it to a numerical value. Also, since data input is terminated by entering END, an error (ERR2) occurs if anything else is entered.

3-3-10 Input/Output Control Functions (KEY$, CSR)

Although the KEY$ function is used to enter data, it differs from an INPUT statement as follows.

INPUT statement	KEY$ function
• Within a 12 digit mantissa and a 2 digit exponent for a numerical value. • Up to 7 characters for a character variable and up to 30 characters for the exclusive character variable ($).	• Reads the character of the key which has been pressed and assigns it to a character variable.
• Input waiting is indicated by a "?" display.	• No input waiting display occurs.

Example)

```
10 INPUT A
20 PRINT A
30 B$=KEY$
40 IF B$="" THEN 30
50 PRINT B$
60 END
```

Line 10 uses an INPUT statement while lines 30 and 40 utilize the KEY$ function. The KEY$ function accepts input of one character from the keyboard, but no input waiting display occurs and the execution does not stop. Therefore, this function is combined with an IF statement as shown on line 40, and if character input is not performed, a return is made to line 30.

Example)

```
 10 A$=KEY$
 20 IF A$="1" THEN 100
 30 IF A$="2" THEN 200
 40 IF A$="3" THEN 300
 50 GOTO 10
100 PRINT "LINE 100":END
200 PRINT "LINE 200":END
300 PRINT "LINE 300":END
```

In the previous example a check was made for key depression. In this program a check is made for the keyboard entry of 1, 2 or 3. If a condition is true, an advance is made to the next work.
When the KEY$ function is used at the beginning of a program like in this example, pay attention to program starting. There are two different program starting methods. When $\boxed{\text{SHIFT}}$ $\boxed{\text{P1}}$ is used for a program starting method, unless the $\boxed{1}$ key is released immediately, the numeral 1 is read by the KEY$ function; "LINE 100" will be displayed.
When the KEY$ function is used at the beginning of a program, add the following lines.

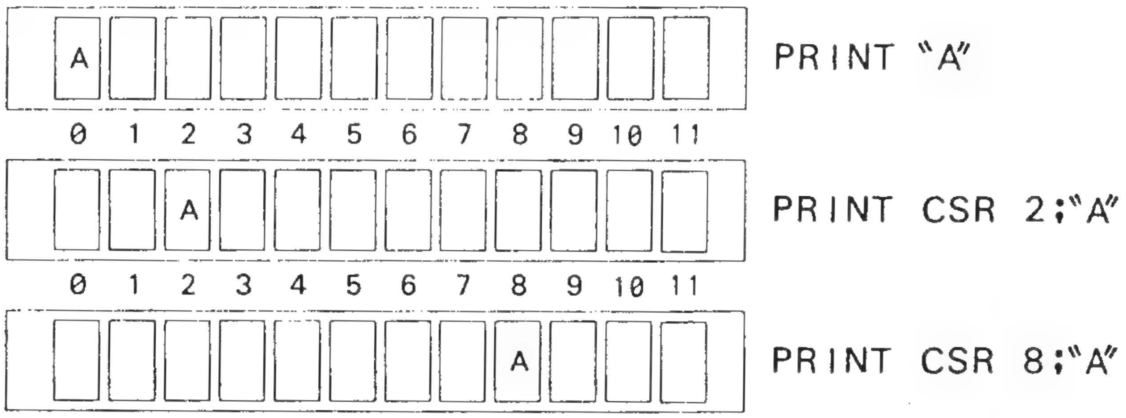
```
 5 A$=KEY$             }
 6 IF A$≠" " THEN 5    } Waits until the pressed key is released.
10 A$=KEY$             }
   .
   .
   .
```

The CSR function specifies a data display location; it is used in a PRINT statement.

Example)

```
10 PRINT "A"
20 PRINT CSR 2;"A"
30 PRINT CSR 8;"A"
40 END
```

The CSR function can be understood by executing this program.
When this function is used, display starts from the specified location (0, 1, 2, . . . or 11 from the left).
When it is not used, display starts from the extreme left.



Example)

```
10 A=INT(RAN#*12)
20 PRINT CSR A;"↑"
30 GOTO 10
```

This program generates a numerical value from 0 to 11 by using the RAN# function and displays "↑" with the CSR function. After a certain time, "↑" is displayed at different locations. An interesting game can be prepared by combining the KEY$ and CSR functions.

Example)

```
 10 D=0:$=" 0123456789"
 20 FOR B=1 TO 10
 30 IF KEY$≠"" THEN 30
 40 A=INT(RAN#*10)
 50 PRINT MID$(1,A+1);"↑";MID$(A+3);
 60 FOR C=1 TO 30
 70 E$=KEY$
 80 IF E$≠"" THEN 100
 90 NEXT C
100 IF E$<"0" THEN 140
110 IF E$>"9" THEN 140
120 IF VAL(E$)=A THEN D=D+1
140 PRINT
150 NEXT B
160 PRINT CSR 2;"RIGHT";D;
170 IF D≠10 THEN 210
180 FOR B=1 TO 10
200 NEXT B
210 END
```

This is a game program. Numerals from 0 to 9 are displayed. One of these is displayed as "↑". Press the numeral key ($\boxed{0}$ to $\boxed{9}$) which corresponds to the location of "↑".
On line 50, "↑" is displayed at the location corresponding to the numeral from 0 to 9 generated by the RAN# function on line 40.
Line 30 is used to wait until the pressed key is released.
Line 60 and after test if a key is pressed, in which case, repetition ceases and a test is made to see whether the correct key is pressed or not.
Lines 100 and 110 test if a numeral key is pressed.
Line 120 is used to test if the answer is correct. Since KEY$ reads a character, the condition test is performed after converting this character to a numerical value with the VAL function.
Whether a key is pressed or not is simply checked as shown on line 30, so the test can be performed without assigning KEY$ to a character variable. However, when several tests are performed to check if the answer is correct as shown on lines 70 to 120, KEY$ is assigned to a character variable. Store this program to play.

Operation example

Display	**Operation**
01↑3456789	Press $\boxed{2}$.
01234567↑9	Press $\boxed{8}$.
0123↑56789	Press $\boxed{4}$.
⋮	⋮

If the speed with which the display changes is too fast, set the final value of the FOR statement on line 60 to a numeral larger than 30.

3-4. CONVENIENT OPTIONAL EQUIPMENTS

This computer has optional equipments: a cassette tape recorder interface (FA-3), and a character mini printer (FP-12S or FP-12).
The FA-3 allows programs and data to be stored from the computer on a tape, or to be loaded from a tape to the computer.
The FP-12S (or FP-12) can print programs, data and arithmetic results.
The functions of these equipments are explained below.

3-4-1 Storing A Program Or Data

To store a program or data on cassette tape, the FA-3 is necessary. To connect the FA-3 to the computer and to a tape recorder, see the Instruction Manual that comes with the FA-3.

■ Program storing and loading

Since programs are memorized in the computer, sometimes the enxt program cannot be memorized because of memory capacity. If the previous program is erased, it cannot be used again. In this case, the FA-3 is very helpful.
Commands for storing programs on a tape are "SAVE" or "SAVE ALL". "SAVE" can only store a program located in one program area, while "SAVE ALL" can simultaneously store programs located in all program areas.

SAVE command

MODE 0

```
READY Pn
```

→The program located in this program area can be stored.

SAVE ALL command

Programs located in all program areas can be stored.

The SAVE and SAVE ALL commands are manually executed.

Example)

```
SAVE EXE
SAVE "CASIO" EXE
SAVE ALL EXE
SAVE ALL "PB" EXE
```

Characters enclosed with the quotation marks (") after SAVE and SAVE ALL are file names which are placed with stored programs. These programs can be loaded later by specifying these names. Up to 8 characters can be used for a file name.
LOAD and LOAD ALL commands are used to load programs from a tape to the computer. The proper use of these commands depends on whether programs were stored by SAVE or SAVE ALL.

Storing \ Loading	LOAD	LOAD "file name"	LOAD ALL	LOAD ALL "file name"
SAVE	○	×	×	×
SAVE "file name"	○	○	×	×
SAVE ALL	×	×	○	×
SAVE ALL "file name"	×	×	○	○

* Items marked with " o " can be loaded; those marked with " X " cannot be loaded.
* File names must be identical.

Example)

```
LOAD EXE
LOAD "file name" EXE
LOAD ALL EXE
LOAD ALL "file name" EXE
```

When programs are loaded by LOAD or LOAD ALL, a display depending on the storing format appears.

Storing format	Display
SAVE	PF:
SAVE "file name"	PF: file name
SAVE ALL	AF:
SAVE ALL "file name"	AF: file name

A program stored by a SAVE command can be loaded to any of the program areas by a LOAD command.

Example) Stores the program of P0.
↓
Loads it to P9.

Precautions:
Sometimes a program cannot be stored or loaded smoothly. If this happens, check the following items.

- "ERR9" is displayed during storing.
  [Check point]
  Check if the computer is properly connected to the FA-3.
- "ERR9" is displayed during loading.
  [Check points]
  If the tape is stretched, replace it with a new one.
  If the head of the tape recorder is dirty, clean it.
  Set the tone control of the tape recorder to medium.
- No error is displayed but loading is attempted without success.
  [Check points]
  If the tape recorder output volume is low, increase the volume near MAX.
  Check if the output standard of the tape recorder is in accordance with that of the FA-3.

■ Data storing and loading

A program always has data; it is troublesome to enter these data from the keyboard each time.
Try a method by which data in the memory are stored on tape and loaded again.
To store data on a tape, "PUT" is used.
Variables are specified in a PUT command. A file name can also be specified.

$$\text{PUT } \underbrace{\text{"file name"}}_{\text{Up to 8 characters.}}\ \ \$,\underbrace{\text{A,Z}}_{\text{Stores 26 variables from A to Z.}}$$

For a file name, up to 8 characters can be placed inside " " as for program storing.
If the exclusive character variable ($) is used, specify it first. Then next two variable names are specified to determine the beginning and end of the variables to be stored.

Example)
Store the content of the exclusive character variable ($) and 13 variables from A to M.

```
PUT $,A,M
```

Store the content of 36 variables from A to Z(10) with a file name, "CASIO".

```
PUT "CASIO" A,Z(10)
```

Since the variable names specify the beginning and end of the variables to be stored, place them in alphabetical order (e.g., "A, Z"). A specification such as "Z, A" cannot be performed.
When the variables are character variables, "A, Z" can be specified instead of "A$, Z$".
"GET" is used to load data from a tape to the computer. Variables are specified in a GET command. A file name can also be specified.

$$\text{GET } \underbrace{\text{"file name"}}_{\text{Up to 8 characters.}}\ \ \$,\underbrace{\text{M,W}}_{\text{Loads to variables from M to W.}}$$

Example)
Load data to the exclusive character variable ($) and 3 variables from X to Z.

```
GET $,X,Z
```

Load data with a "PB" file name to variables from G(0) to G(59).

```
GET "PB" G(0),G(59)
```

When data is being loaded by a GET command, a display depending on the storing format appears.

Storing format	Display
PUT $, A, Z	DF:
PUT "file name" G, P	DF: file name

3-4-2 Keeping A Record

If the content of a program can be printed after preparation, it is convenient to perform debugging or to modify its content. It is also convenient to be able to keep an arithmetic result after it is printed.
Perform printing with the FP-12S (or FP-12) character printer.
To perform printing the print mode ("PRT" is displayed) must be specified by pressing MODE 7. This mode can be released by pressing MODE 8.
To print the content of a program, execute the LIST command after pressing MODE 7.

Example)
Input the following program.

```
10 INPUT A
20 PRINT A*A
30 GOTO 10
```

3-5. USING A PB-100 PROGRAM

Programs prepared for the PB-100 and PB-300 can be utilized with this computer.
This computer is provided with more commands than the PB-100/PB-300; its utilization is more convenient.

The BASIC language used by this computer is almost the same as that used by the PB-100/PB-300.

■ Different points

- **Additional commands**
  PASS (Program protection)
  READ (Reads data from a DATA statement)
  DATA (Writes data)
  RESTORE (Specifies data to be read)
  ON-GOTO (Indirect specification of a GOTO statement)
  ON-GOSUB (Indirect specification of a GOSUB statement)
  REM (Comment statement)
- **Additional functions**
  DEG (Sexagesimal → decimal conversion)
  DMS$ (Decimal → sexagesimal conversion)
  STR$ (Converts a numerical value to a character string)
- **Modified commands**

This computer	PB-100/PB-300
NEW (NEW ALL)	CLEAR (CLEAR A)
CLEAR	VAC
IF–THEN	IF–;
SAVE ALL	SAVE A
LOAD ALL	LOAD A
VERIFY	VER
DEFM (Can be written in a program)	DEFM (Can only be performed manually.)

- **Modified functions**

This computer	PB-100/PB-300
KEY$ MID$	KEY MID

In spite of these different points, a program prepared by the PB-100/PB-300 can be fundamentally utilized with this computer.
However, it is better that programs be rewritten for this computer so that it can be easily used or can be easily reconsidered later.

Example 1)

PB-100 program

```
10 VAC
20 FOR A=1 TO 20
30 INPUT Z(A)
40 IF Z(A)>80;B=B+1:GOTO 90
50 IF Z(A)<60;C=C+1:GOTO 90
60 IF Z(A)>40;D=D+1:GOTO 90
70 IF Z(A)>20;E=E+1:GOTO 90
80 F=F+1
90 NEXT A
     :
     :
```

This example is part of a program to enter data and distribute them according to their size. Although the program could be used as it is, correct the following items.
Change "VAC" on line 10 to "CLEAR".

```
10 CLEAR
```

Change " ; " on lines 40 to 70 to "THEN".

```
40 IF Z(A)>80 THEN B=B+1:GOTO 90
     :
     :
```

Since memory expansion is necessary in this program, write the DEFM command, manually executed in the PB-100/PB-300, at the beginning.

```
5 DEFM 20
```

Example 2)
PB-100 program

```
10 INPUT "I=1/O=2/P=3",N
20 IF N<1 THEN 10
30 IF N>3 THEN 10
40 GOTO N*100
        :
        :
```

This program is used to determine branch locations according to the work. To adapt it for this computer, modify it as follows by using an ON-GOTO statement.

```
10 INPUT "I=1/O=2/P=3",N
20 ON N GOTO 100,200,300
30 GOTO 10
      :
      :
```

The program is simplified by utilizing an ON-GOTO statement as mentioned above; testing the data N is deleted.
Programs and data stored on tape by CASIO'S handheld computers can be loaded as they are to this computer. However, the reverse operation is not always possible. Therefore precautions shall be taken. The relationships are as follows.

This computer → FX-710P

SAVE / LOAD	PF	AF	MF	with password		
				PF	AF	MF
LOAD	○	⧄	⧄	○	⧄	⧄
LOAD ALL	⧄	○	⧄	⧄	○	⧄

This computer → PB-100, PB-300, FX-700P, FX-802P

SAVE / LOAD	PF	AF	MF	with password		
				PF	AF	MF
LOAD	○	⧄	⧄	⧄	⧄	⧄
LOAD ALL	⧄	○	⧄	⧄	⧄	⧄

○ : Can be loaded.
⧄ : Cannot be loaded.

[PRECAUTIONS]

- KEY$ and MID$ should be changed to KEY and MID for the PB-100, PB-300, FX-700P and FX-802P.
- When a program prepared by other CASIO's computers is executed with this computer, sometimes it cannot be properly executed as shown below.
  * If a numerical expression is used for a branch location in an IF - THEN statement, an error occurs. In this case, change it to an IF - THEN - GOTO statement.

CHAPTER 4
COMMAND REFERENCE

* The following descriptions apply symbols and terms frequently used in the syntax.

- $\left\{\begin{matrix}\times\times\times\times\\ \text{OOOO}\end{matrix}\right\}$ .... One of the elements inside { } must be selected.
- [OOOO] .... The element inside [ ] can be omitted
- OOOO* .... The element with * on the top right can be repeatedly used.
- Numerical expression .....
  Numerical value, calculation expression, and numerical variable such as 10, 2+3, A, S*Q.
- Character expression .....
  Character constant, character variable, and character expression such as "ABC", X$, N$+M$.
- Parameter .... An element that accompanies a command.
- Ⓟ .... Can only be executed in a program.
- Ⓜ .... Can only be executed manually.
- Ⓐ .... Can be executed both manually and in a program.
- Ⓕ .... Function instruction that can be executed both manually and in a program.

⟨Example⟩ DATA [data] [,[data]]*

Since all data are provided with a bracket [ ], it will also be possible to write "DATA" only. Since ,[data] is provided with [ ]*, this element can be written repeatedly. This can therefore be written "DATA data, data, ..." If we omit the first [data], this can also be written "DATA, data, data, ..."

$$\textbf{GOTO}\left\{\begin{matrix}\textbf{Line No.}\\ \#\ \textbf{program area No.}\end{matrix}\right\}$$

There are two different ways to write this statement as shown below.

(1) GOTO line No.

(2) GOTO # program area No.

NEW [ALL]

Ⓜ

Function

Program erase. Erases programs and variables.

Parameter

When ALL is specified, all P0 ~ P9 programs and variables are erased.

Explanation

(1) If ALL is not specified, the program in the presently specified program area is erased. Variables are not erased.
(2) If ALL is specified, the programs in all program areas and variables are erased. The DEFM setting is released and the number of memories is initialized to 26.
(3) Cannot be executed while a password is specified.
(4) Cannot be used in a program.
(5) Can only be executed in the WRT mode.
* NEW ALL can be abbreviated as NEW A.

Example MODE 1 NEW EXE

RUN [Execution start line]

line No.

Ⓜ

Function

Program execution.

Parameter

When a line is specified, execution starts from the line.

Explanation

(1) Executes a program from a specified line (when the line number is omitted, execution starts from the beginning of the program).
(2) When a specified line number does not exist, execution starts from the line with the closest larger number.
(3) Variables are not cleared.

Example

```
10 PRINT "LINE 10"
20 PRINT "LINE 20"
30 END
```

```
RUN EXE                LINE 10
RUN 20 EXE             LINE 20
```

LIST $[\left\{\begin{matrix}\text{line No.}\\\text{ALL}\end{matrix}\right\}]$ Ⓜ

Function

Displays the content of a program.

Parameter

Line No.: No. of the first line to be displayed.
ALL: Displays the content of all P0—P9 programs sequentially.

Explanation

I. RUN mode

(1) Sequentially displays the content of a program from a line number if it is specified, or from the beginning if it is omitted.
(2) Since the content of a program is automatically displayed sequentially, press the STOP key to stop this. Press the EXE key to display the next line and after.
(3) In the PRINT mode (when "PRT" is displayed) with a printer connected, the display is not stopped but is made sequentially at high speed.

II. WRT mode

(1) Displays the content of a program from a line number if it is specified, and from the beginning if it is omitted.
(2) Since each line is displayed for edit in the WRT mode, if edit is not required, press the EXE key to advance to the next line. Also, if the SHIFT key is pressed before the EXE key, the previous line is displayed.

- When ALL is specified, the content of all P0—P9 programs are sequentially displayed. In this case they are sequentially advanced even in the WRT mode, so edit cannot be performed.
- This command cannot be used while a password is specified.

* LIST ALL can be abbreviated as LIST A.

Example

```
LIST EXE
LIST 30 EXE
```

PASS "Password" / Character string

Ⓜ

Function

Specifies or releases a password.

Parameter

Password: String with 1—8 characters.

Explanation

(1) If this command is executed when a password is not specified, a password is specified for all program areas (P0—P9).
(2) If this command is executed while a password is specified, this password is released only when entering the corresponding password. When passwords do not correspond, a protect error (ERR8) occurs.
(3) A password consists of a 1—8 character string in which spaces, alphabetical characters, numerals, special symbols, etc. can be used. However, (") cannot be used.
(4) While a password is specified, commands such as LIST, LIST ALL, LIST#, NEW, NEW ALL and NEW# cannot be used. Also no writing (WRT mode) can be made; if it is attempted, an error (ERR8) occurs.
(5) Cannot be used in a program.
(6) A password can be maintained while the power switch is off.
(7) If a program is stored on a cassette tape by a SAVE or SAVE ALL command while a password is specified, this password is also stored. When a program with a password attached is loaded from a cassette tape by a LOAD or LOAD ALL command, the password is also loaded. Also, when a currently specified password in the mainframe and the password of a program loaded from a cassette tape are different, the program cannot be loaded from a cassette tape (ERR8).

Precaution

If a password was forgotten after it was specified, press the ALL RESET button on the back of the computer and clears all the programs and memory.

Example PASS "CASIO" EXE

SAVE [ALL] [“File name”] Ⓜ

Character string

Function

Stores a program on a cassette tape.

Parameter

ALL: Stores the programs in all the program areas.
File name: String with 1—8 characters. Can be omitted.

Explanation

(1) When ALL is omitted, the content in the presently specified program area is stored.
(2) When ALL is used, the contents of all P0—P9 program areas are stored.
(3) When a password is specified, the storing is performed with that password. Therefore, the password is the same as that stored when the program is loaded by the LOAD command.

* SAVE ALL can be abbreviated as SAVE A.

Example

```
SAVE EXE
SAVE "CASIO"EXE
SAVE  ALL "PB"EXE
```

LOAD [ALL] ["File name"]

Character string

M

Function

Loads a program from a cassette tape.

Parameter

ALL: Loads the programs in all program areas.
File name: String with 1—8 characters. Can be omitted.

Explanation

(1) When ALL is omitted, a program stored by "SAVE" is read into the presently specified program area.
(2) When ALL is used, programs stored by "SAVE ALL" are read into the P0—P9 program areas.
(3) When a program stored with a password attached is loaded from a cassette tape, this password is also loaded.
* Load ALL can be abbreviated as LOAD A.

SAVE and LOAD Relationship

	LOAD	LOAD "File name"	LOAD ALL	LOAD ALL "File name"
	○	×	×	×
SAVE "File name"	○	○	×	×
SAVE ALL	×	×	○	×
SAVE ALL "File name"	×	×	○	○

* File names are identical.
○... Can be loaded.
×... Cannot be loaded.

VERIFY ["File name"]

Character string

Ⓜ

Function

Checks the status of a program and data stored on a cassette tape.

Parameter

File name: String with 1—8 characters. Can be omitted.

Explanation

(1) When a file name is specified, the file with this name is checked.
(2) When the file name is omitted, checks the first file that appears on the cassette tape.
(3) The parity check system is used to check a storing format.

Example

```
VERIFY EXE
VERIFY "PROG1" EXE
```

CLEAR

Ⓐ

Function

Clears all variables.

Explanation

(1) Clears all variables; all numerical variables are cleared to 0 and all character variables to a null.
(2) This command can be used both in a program and manually.
(3) Since control variables are also cleared in a FOR-NEXT loop (see page 130), an error occurs during NEXT statement execution.

* The CLEAR command functions the same as VAC.

END Ⓟ

Function

Terminates program execution.

Explanation

Since program execution is terminated, the next program is not executed even if it exists.

STOP Ⓟ

Function

Temporarily suspends program execution.

Explanation

(1) Temporarily suspends program execution and displays "STOP" after which input waiting occurs.
(2) After suspension, execution is resumed by pressing the EXE key.
(3) If the STOP key is pressed while execution is stopped by a STOP statement, the program area number and line number are displayed.
(4) During execution suspension by STOP, manual calculations can be performed.

[LET] $\left\{\begin{array}{l}\text{Numerical variable = numerical expression}\\ \text{Character variable = character expression}\end{array}\right\}$ Ⓟ

Function

Assigns the value of the expression on the right to the variable on the left.

Explanation

(1) A numerical expression corresponds to a numerical variable, and a character expression corresponds to a character variable.
(2) LET can be omitted.

Example

```
10 LET X=12
20 LET Y=X↑2+2∗X-1
30 PRINT Y
40 A$="CASIO"
50 B$="PB&FX"
60 PRINT A$;B$
70 END
```

REM $\underbrace{\text{Comment}}_{\text{Character string}}$ Ⓟ

Function

Statement that expresses a comment.

Explanation

(1) Written in a program. Content after REM is treated as comment statement and is threfore not executed.
(2) When a command to be executed is written on the same line, write a multistatement sign (:) before the REM statement.

Example

```
10 INPUT "R",R
20 S=π∗R↑2:REM AREA
30 PRINT S
40 END
```

Ⓟ

INPUT $\left[\underbrace{\text{"Message statement",}}_{\text{Character string}}\right]$ variable name $\left[,\ \left[\underbrace{\text{"Message statement",}}_{\text{Character string}}\right]\text{variable name}\right]^*$

Function

Inputs data from the keyboard to a variable.

Parameter

Message: Character string
Variable name: Numerical variable name or character variable name.

Explanation

(1) Input data from the keyboard to a specified variable.
(2) When a message exists, it is displayed followed by "?".
(3) When there is no message, only "?" is displayed.
(4) Press the EXE key after data input.
(5) When character data are entered into a numerical variable, an error (ERR2) occurs and data input is requested again by the display of "?" after the AC key is pressed. When a numerical expression is entered, the result of this expression is assigned. When one alphabetical character is entered, the value of the variable corresponding to this character is assigned.
(6) When the EXE key is pressed during input waiting, it becomes null input. So, an error (ERR2) occurs if the variable is a numerical variable.

Example

```
10 INPUT A
20 INPUT "B$=",B$
30 INPUT "C$=",C$,"D$=",D$
```

KEY$

Ⓟ

Function

A function that enters one character from the keyboard.

Explanation

(1) The input of only one character is accepted from the keyboard.
(2) Numerals, alphabetical characters, and symbols can be input.
(3) Since "?" is not displayed and input waiting does not occur, KEY$ is usually combined with an IF statement.
* KEY$ can be abbreviated as KEY.

Example

```
10 PRINT "INPUT<6>";
20 A$=""
30 K$=KEY$
40 IF K$="" THEN 30
50 A$=A$+K$
60 IF LEN(A$)<6 THEN 30
70 PRINT A$
80 END
```

* Six characters are accepted from the keyboard.

PRINT [Output element] $\left[\begin{Bmatrix} , \\ ; \end{Bmatrix}\right.$ [Output element]$\left.\right]^*$ Ⓟ

Function

Displays an output element.

Parameter

Output element: Output control function (CSR), numerical expression, character expression.

Explanation

(1) Displays an output element. When an output control function is added, the element is displayed at the location determined by this function.
(2) Values are displayed for numerical expressions and character expressions.
(3) When an output element is a numerical expression, a position for sign (+, −) is placed before the value. However, the + sign is displayed as a blank.

- **Character display**........................ $\underbrace{\sqcup\sqcup\sqcup\cdots\cdots\sqcup}_{\text{Output element}}$
- **Numeral display**............................ $\underset{\text{Sign}}{\sqcup}\underbrace{\sqcup\sqcup\sqcup\cdots\cdots\sqcup}_{\text{Output element}}$

(4) When an output element is a numerical expression and the mantissa is more than 10 digits, the 11th digit is rounded off. When an exponent exists besides the mantissa, an exponent sign (E) and a two digit exponent are displayed.
(5) " , " and " ; " can be used as punctuation between output elements. When " , " is used, the execution stops (STOP is displayed) after the first output element is displayed, then the next output element is displayed by pressing the EXE key. When " ; " is used, the next output element is displayed continuously after the first one.
(6) When no output element is specified (only PRINT is written), the display is cleared and is not stopped.
(7) The display is not stopped during printing in the print mode (MODE 7).
(8) The output format of the numerical value can be specified by a SET statement.

Example

```
10 PRINT 1/3
20 PRINT "A="; A
30 PRINT "SIN 30", SIN 30
40 PRINT "END";
50 PRINT
60 END
```

CSR Output location specification Ⓕ

Numerical expression

Function

Displays an output element from a specified location.

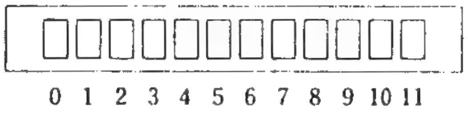
Parameter

Output location specification: Numerical expression. Values below decimal point are discarded.

0 ≦ specification < 12

Explanation

(1) Used in a PRINT statement to specify the location of an output element.
(2) The output location of the left end is 0.



Example

```
10 FOR I=0 TO 11
20 PRINT CSRI;"A";CSR 11-I;"B"
30 NEXT I
40 END
```

• A and B characters are shifted from the left and right respectively each time the EXE key is pressed.

GOTO $\left\{ \begin{array}{c} \underline{\text{Branching line No.}} \\ \text{line No.} \\ \underline{\text{\# program area No.}} \\ \text{Number 0 to 9} \end{array} \right\}$ Ⓟ

Function

Unconditionally branches to a specified location.

Parameter

Branching line No.: Line No. from 1 to 9999.
Program area No.: A number from 0 to 9.

Explanation

(1) Branches to a specified location.
(2) When a branching location is a line number, branches to the specified line in the current program area and executes the program. When the branching line number does not exist, an error (ERR4) occurs.
(3) When the branching location is a program area number, branches to the specified program area and executes the program from the beginning.
* A numerical expression can be used for the branching line number and the program area number.

Example

```
 10 PRINT "START";
 20 GOTO 100
 30 PRINT "LINE 30"
 40 END
100 PRINT "LINE 100"
110 GOTO 30
```

ON $\frac{\text{Branching condition}}{\text{Numerical expression}}$ GOTO [Branching location] [, [Branching location]]* Ⓟ

★ Branching location { Branching line No. / # program area No.

Function

Branches to a specified location according to the branching condition.

Parameter

Branching condition: Numerical expression. Values below the decimal point are discarded.
Branching line No.: Line No. from 1 to 9999.
Program area No.: A number from 0 to 9.

Explanation

(1) Branches according to the integer part of the value in a branching condition expression. Branching locations are allocated sequentially according to

ON A GOTO $\underset{A=1}{\underline{100}}$, $\underset{A=2}{\underline{200}}$, $\underset{A=3}{\underline{300}}$, ·········

(2) When the value of the expression is smaller than 1, or when an appropriate branching location does not exist, the next statement is executed without branching.
(3) As many branching locations that can fit on one line can be written.

Example

```
 10 INPUT A
 20 ON A GOTO 100,200,300
 30 PRINT "OTHER"
 40 GOTO 10
100 PRINT "LINE 100":GOTO 10
200 PRINT "LINE 200":GOTO 10
300 PRINT "LINE 300":GOTO 10
```

• When 1—3 is entered, branchings to 100—300 are performed respectively, otherwise "OTHER" is displayed.

IF $\frac{\text{Branching condition}}{\text{Conditional expression}}$ THEN $\begin{Bmatrix}\text{Statement [: statement]*} \\ \text{Branching location}\end{Bmatrix}$ Ⓟ

★ Branching location $\begin{cases}\text{Branching line No.} \\ \text{\# program area No.}\end{cases}$

Function

When a branching condition is true, the statements after THEN are executed. Also, when a statement after THEN is a branching location, branching is performed.

Parameter

Branching condition: Conditional expression
Branching line No.: Line No. from 1 to 9999.
Program area No.: A number from 0 to 9.

Explanation

(1) When the branching condition is true, the statements after THEN are executed or branching is performed.
(2) When the branching condition is false, the next line is executed.
(3) The branching condition is tested by a conditional expression (=, ≠, <, >, ≦, ≧).

=	The item on the left is equal to the item on the right.
≠	The item on the left is not equal to the item on the right.
<	The item on the right is larger than that on the left.
>	The item on the right is smaller than that on the left.
≦	The item on the right is larger than or equal to that on the left.
≧	The item on the right is smaller than or equal to that on the left.

(4) When two or more branching conditions exist, several IF-THEN statements can be written sequentially.

IF — THEN IF — THEN ....

* When a statement exists after THEN, " ; " can be used instead of THEN.

Example

```
10 N=6
20 PRINT CSR N;"↑";
30 K$=KEY$
40 IF K$="4"THEN N=N-1:IF N<0THEN N=0
50 IF K$="6"THEN N=N+1:IF N>11THEN N=11
60 PRINT
70 GOTO 20
```

● " ↑ " is shifted to the left when the $\boxed{4}$ key is pressed and is shifted to the right when the $\boxed{6}$ key is pressed.

FOR $\frac{\text{Control variable name} = \text{Initial value}}{\text{Numerical expression}}$ TO $\frac{\text{Final value}}{\text{Numerical expression}}$ [STEP $\frac{\text{Increment}}{\text{Numerical expression}}$] NEXT Control variable name Ⓟ

Function

Repeats process contained between FOR and NEXT statements a number of times specified by the control variable. The value of this variable is changed, from the initial to the final one, by the increment for each repetition of the process.

Parameter

Control variable name: Simple variable name.
An array variable can not be used.
Initial value: Numerical expression
Final value: Numerical expression
Increment: Numerical expression
The value 1 is taken in default of this.

Explanation

(1) Repeats process contained between FOR and NEXT statements a number of times specified by the control variable. The value of this variable is changed, from the initial to the final one, by the increment for each repetition of the process. When the value of the control variable exceeds the final value, repetition is terminated.
(2) When the initial value is larger than the final value, the execution between FOR-NEXT is performed only once.
(3) A negative number can be used for an increment.
(4) A NEXT statement must always correspond to a FOR statement and must be written after it.
(5) FOR-NEXT loops can have the following nested structure.

```
10 FOR I=1 TO 10      ─┐
20 FOR J=11 TO 20─┐    │
30 PRINT I;":";J  │    │
40 NEXT J        ─┘    │
50 NEXT I             ─┘
60 END
```

(6) Nesting can be performed with up to 4 levels.

(7) When a FOR-NEXT loop is terminated, the value of the control variable exceeds the final value by the value of the increment.
(8) A branching out of a FOR-NEXT loop can be performed. If branching inside a FOR-NEXT loop by an IF statement or GOTO statement is attempted, an error occurs.

GOSUB

$$\mathbf{GOSUB}\ \left\{\begin{array}{c}\dfrac{\text{Branching line No.}}{\text{Line No.}}\\ \dfrac{\#\ \text{program area No.}}{\text{A character from 0 to 9}}\end{array}\right\}$$

Ⓟ

Function

Performs a branching to a specified subroutine.

Parameter

Branching line No.: Line No. from 1 to 9999.
Program area No.: A character from 0 to 9.

Explanation

(1) Performs a branching to a subroutine. A return from this subroutine is performed by executing RETURN.
(2) To make a subroutine inside a subroutine is called nesting which can be performed with up to 8 levels.
(3) Return to the statement next to the GOSUB statement is performed by RETURN.
(4) Return to the main routine cannot be performed by an IF statement or GOTO statement. Therefore, be sure to perform return by a RETURN statement.
(5) When the branching line No. does not exist, an error (ERR4) occurs.
* A numerical expression can also be used for a branching line number and a program area number.

Example

```
 10 PRINT "MAIN 10"
 20 GOSUB 100
 30 PRINT "MAIN 30"
 40 END
100 PRINT "SUB 100"
110 GOSUB 200
120 RETURN
200 PRINT "SUB 200"
210 RETURN
```

RETURN Ⓟ

Function

Provides a return from the subroutine to the main program.

Explanation

Returns to a statement located just after the statement which called the subroutine.

ON $\frac{\text{Branching condition}}{\text{Numerical expression}}$ GOSUB [Branching location] [, [Branching location]]* Ⓟ

★ Branching location $\begin{cases}\text{Branching line No.}\\ \text{\# program area No.}\end{cases}$

Function

Branches to a subroutine according to a branching condition.

Parameter

Branching condition: Numerical expression.
Values below the decimal point are discarded.
Branching line No.: Line No. from 1 to 9999.
Program area No.: A character from 0 to 9.

Explanation

(1) Performs a subroutine branching by the integer part of the value in a branching condition expression. Branching locations are allocated sequentially according to the value of the expression.

$$\text{ON B GOSUB } \underbrace{1000}_{B=1},\ \underbrace{2000}_{B=2},\ \underbrace{3000}_{B=3}\cdots\cdots$$

(2) When the value of the expression is smaller than 1 or an appropriate branching location does not exist, the next statement is executed without branching.
(3) As many branching locations as can fit in one line can be written.

Example

```
 10 INPUT A
 20 ON A GOSUB 100,200,300
 30 GOTO 10
100 PRINT "SUB 100":RETURN
200 PRINT "SUB 200":RETURN
300 PRINT "SUB 300":RETURN
```

• When 1—3 is entered, a branching to the corresponding subroutine occurs.

DATA

$$[\underbrace{\text{data}}_{\text{Constant}}]\ \ [,[\underbrace{\text{data}}_{\text{Constant}}]]^*$$ Ⓟ

Function

Stores data.

Parameter

Data: Character constant or numerical constant.

Explanation

(1) Used to write data that is read by a READ statement.
(2) Plural data can be written by punctuation with " , ".
(3) If only a DATA statement is executed without a READ statement, no function is performed.
(4) When a character constant includes " , ", place it inside " ".

$$\text{DATA } \underbrace{\text{ABC}}_{\text{1st}},\ \underbrace{\text{DEF}}_{\text{2nd}},\ \underbrace{\text{"GHI, JKL"}}_{\text{3rd}},\cdots\cdots$$

(5) When data is omitted, a character string with a length of 0 is taken by default.

```
DATA A, ,B  → DATA A,"",B
DATA ,      → DATA "",""
DATA        → DATA ""
```

READ Variable name [, [variable name]]* Ⓟ

Function

Reads the content of a DATA statement.

Parameter

Variable name: Numerical variable or character variable. An array variable can be used.

Explanation

(1) Allocates data in the currently specified DATA statement sequentially to a specified variable.
(2) Only numerical type data can be read for a numerical variable.
(3) Data in DATA statements are read sequentially with the smallest line number first, and sequentially from the beginning in a statement.
(4) After the necessary data are read by a READ statement, the following data are read by the next READ statement.
(5) The first data in the program area where a READ statement exists is read by the first execution of this statement after which data in the program area at that time are read sequentially.
(6) The specification of data to be read can be changed by a RESTORE statement.
(7) When the number of data in a DATA statement is smaller than the number of variables in a READ statement, an error (ERR4) occurs.
(8) When a space exists at the beginning of data, it is skipped.

Example

```
10 DATA 1,2,3
20 READ A,B
30 PRINT A;B
40 DATA 4,5
50 READ C,D,E
60 PRINT C;D;E
70 END
```

• Reads data sequentially from a DATA statement and displays them.

RESTORE [Line No.] (P)

Numerical expression

Function

Specifies the location of data to be read by a READ statement.

Parameter

Line No.: Numerical expression. Values below the decimal point are discarded.

1 ≦ line No. ≦ 9999

Explanation

(1) Specifies a DATA statement where data to be read by a READ statement exist.
(2) When a line number is omitted, the data specification is cancelled. After this, the first data in the program area where a READ statement exists are specified and read by the first READ statement that is executed.
(3) When a line number of the program area is specified by a RESTORE statement, data of the DATA statement with this line number are read sequentially by the READ statement.
(4) When a specified line number does not exist or a DATA statement does not exist on a specified line number and after, an error (ERR4) occurs.

Example

```
10 DATA 1,2,3
20 DATA 4,5
30 READ A,B,C,D,E
40 RESTORE 10
50 READ F,G
60 RESTORE 20
70 READ H,I
80 PRINT A;B;C;D;E;F;G;H;I
90 END
```

PUT

[" File name "] variable 1 [, Variable 2]* Ⓐ
Character string

Function

Stores data on a cassette tape.

Parameter

File name: A string with 1—8 characters. Can be omitted.
Variable 1, variable 2: Specification of the variable to be stored.

Explanation

(1) Stores the contents of variables on a cassette tape.
(2) Variable specifications are written as follows.

PUT A	Content of variable A.
PUT A,Z	Content of variables A—Z.
PUT A,A (50)	Content of variables A—A(50).
PUT $,D,W	Content of the exclusive character variable $ and of variables D—W.

When the content of the exclusive character variable $ must be stored, write $ first.
(3) Can be executed both manually and in a program.

GET

[" File name "] variable 1 [, Variable 2]* Ⓐ
Character string

Function

Loads data stored on a cassette tape into a variable.

Parameter

File name: A string with 1—8 characters. Can be omitted.
Variable 1, variable 2: Specification of the variable to be loaded.

Explanation

(1) Loads data stored on a cassette tape into a specified variable.
(2) Variable specifications are written as follows.

GET A	Loads in variable A.
GET A,Z	Loads in variables A—Z.
GET A,A (50)	Loads in variables A—A(50).
GET $,D,W	Loads in the exclusive character variables $, and in variables D—W.

(3) A variable name stored by PUT can be different from the name read by GET.
(4) When the number of stored data is smaller than the number of variables to be loaded, only the data are loaded sequentially in the first variables.
(5) It can be executed both manually and in a program.

DEFM

[Size of memory expansion]
Numerical expression

Ⓐ

Function

Provides memory expansion.

Parameter

Size of memory expansion: Numerical expression. Values below the decimal point are discarded.
Can be omitted.

$0 \leqq$ Size of memory expansion < 69

Explanation

(1) Expands the memories (variable area).
(2) An arbitrary number can be specified according to the remaining number of program steps.
(3) Since 8 steps are required for each memory expansion, the number of remaining steps is reduced.
(4) When the size of memory expansion is omitted, the number of currently specified memories is displayed.
(5) It can be executed both manually and in a program. When it is manually executed, the status (number of expanded memories + 26 basic memories) is displayed. When executed by writing it in a program, the status is not displayed.
(6) When an attempt is made to perform expansion larger than the number of remaining program steps, an error (ERR1) occurs.
(7) Specify DEFM 0 to cancel the memory expansion and to return to the 26 basic memories.

Example

```
DEFM 10 EXE      ***VAR:36
DEFM EXE         ***VAR:36

10 DEFM 10
20 FOR I=1 TO 10
30 INPUT Z(I)
40 NEXT I
      ⋮
```

MODE Numerical expression Ⓟ

Function

Sets the state of the computer.

Parameter

Numerical expression: Values below the decimal point are discarded.

4 ≦ numerical expression < 9

Explanation

(1) Sets the angle unit, print mode or releases this mode depending on the numerical expression used.

(2) Settings are as follows.

MODE4 ········· Sets the angle unit to degrees.
MODE5 ········· Sets the angle unit to radians.
MODE6 ········· Sets the angle unit to grades.
MODE7 ········· Displays "PRT" and sets the print mode.
MODE8 ········· Releases the print mode.

(3) Same setting as by the MODE key. However, the RUN mode and WRT mode cannot be set using this command. Also, input cannot be performed with the MODE key, but by pressing the M O D E keys.

Example

```
10 MODE 4
20 A=SIN 30
30 MODE 7
40 PRINT A
50 MODE 8
60 END
```

SET $\left\{\begin{matrix} Fn \\ En \\ N \end{matrix}\right\}$

Ⓐ

★ *n* is an integer from 0 to 9.

Function

Specifies the output format for numerical data.

Parameter

F*n*: Specifies the number of decimal places.
E*n*: Specifies the number of significant digits.
N: Releases a specification.

Explanation

(1) Specifies the number of decimal places or significant digits.
(2) For specifying the number of decimal places (F*n*), a value from 0 to 9 is used.
(3) For specifying the number of significant digits (E*n*), a value from 0 to 9 is used. Also "SET E0" indicates a 10-digit specification.
(4) Both specifications are released by " SET N ".
(5) It can be executed both manually and in a program.

Example

```
10 INPUT N
20 SET F5:PRINT N
30 SET E5:PRINT N
40 SET N:GOTO 10
```

LEN (Simple character variable) Ⓕ

Function

Gives the length of the character string in a simple character variable.

Parameter

Simple character variable: An array variable can not be used.

Explanation

(1) Counts the number of characters in a simple variable.
(2) The character variable used is a simple character variable (A$, Y$, etc.); an array character variable (B$ (3), etc.) cannot be used.

Example

```
10 INPUT A$
20 PRINT LEN(A$)
30 GOTO 10
```

MID$ ($\underset{\text{Numerical expression}}{\underline{\text{Location}}}$ [, $\underset{\text{Numerical expression}}{\underline{\text{Number of characters}}}$]) Ⓕ

Function

Fetches the specified number of characters from a specified location of the exclusive character variable ($).

Parameter

Location: Numerical expression. Values below the decimal point are discarded.

1 ≦ location < 101

Number of characters: Numerical expression. Values below the decimal point are discarded.

1 ≦ number of characters < 101

When omitted, all characters after the specified location are fetched.

Explanation

(1) Fetches a specified number of characters from a specified location of the exclusive character variable ($).
(2) When the specified location is out of the character string, a null is obtained.
(3) When the length of the character string after the specified location is smaller than the specified number of characters, all the characters after the specified location are fetched.

* MID$ can be abbreviated as MID.

Example

```
10 $="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 INPUT M,N
30 PRINT MID$(M,N)
40 END
```

VAL (Simple character variable) Ⓕ

Function

Converts characters in a simple character variable into a numerical value.

Parameter

Simple character variable: An array variable cannot be used.

Explanation

(1) Converts characters in a simple character variable into a numerical value.
(2) When the content of a character variable includes +, −, •, E or E⁻, it is converted into a numerical value as it is.

When A$ = " −12.3 ", VAL(A$) → −12.3

(3) When the content of a character variable starts with a character other than a numeral, +, −, or •, an error occurs.

When A$ = " A45 ", VAL(A$) → − error (ERR2)

(4) When a character other than a numeral is inserted in the middle, only the part before this character is converted to a numerical value.

When A$ = " 78A9 ", VAL(A$) → 78

Example

```
10 INPUT A$
20 PRINT VAL(A$)
30 END
```

STR$ (Numerical expression) Ⓕ

Function

Converts the value of a numerical expression into a character string.

Parameter

Numerical expression: Numerical value, calculation expression, numerical variable, numerical array variable.

Explanation

(1) Converts the value of a numerical expression into a character string.
(2) When the numerical expression is a calculation expression, the calculation result is converted into a character string.
(3) When a numerical expression is positive, the sign digit is deleted and only the numerals are converted.

Example

```
10 PRINT STR$(123)
20 PRINT STR$(45+78)
30 A=963
40 PRINT STR$(A)
50 END
```

NUMERICAL FUNCTIONS

SIN $\frac{\text{Argument}}{\text{Numerical expression}}$ **COS** $\frac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

TAN $\frac{\text{Argument}}{\text{Numerical expression}}$

Function

Obtains the value of a trigonometric function for a given argument.

Parameter

Argument: Numerical expression
$-1440° < \text{argument} < 1440°$ (degrees)
$-8\pi < \text{argument} < 8\pi$ (radians)
$-1600 < \text{argument} < 1600$ (grades)
However, for TAN, "$|\text{Argument}| = (2n-1) * 1$ right angle" is excluded.
1 right angle $= 90° = \frac{\pi}{2}$ rad $=$ 100 grad.

Explanation

(1) Obtains the value of a trigonometric function for a given argument.
(2) The value depends on the angle unit setting (by the MODE key or MODE command).

ASN $\frac{\text{Argument}}{\text{Numerical expression}}$ ACS $\frac{\text{Augument}}{\text{Numerical expression}}$ Ⓕ

ATN $\frac{\text{Argument}}{\text{Numerical expression}}$

Function

Inverse trigonometric function that obtains an angle for a given argument.

Parameter

Argument: Numerical expression.

For ASN, ACS, $-1 \leqq$ argument $\leqq 1$.

Explanation

(1) Inverse trigonometric function that obtains an angle for a given argument.
(2) The value depends on the angle unit setting (by the MODE key or MODE command).
(3) The values of the functions are given within the following range.

$-90° \leqq$ ASN X $\leqq 90°$
$0° \leqq$ ACS X $\leqq 180°$
$-90° \leqq$ ATN X $\leqq 90°$

LOG $\frac{\text{Argument}}{\text{Numerical expression}}$ LN $\frac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function

Gives the value of a logarithmic function.

Parameter

Argument: Numerical expression.

0 < argument

Explanation

Gives the value of a logarithmic function.

- **LOG Common logarithmic function** $\log_{10}x$, $\log x$
- **LN Natural logarithmic function** $\log_e x$, $\ln x$

EXP $\frac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function

Gives the value of an exponential function.

Parameter

Argument: Numerical expression.

$-227 \leqq$ argument $\leqq 230$

Explanation

Gives the value of an exponential function.

EXP e^x

SQR $\frac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function

Gives the square root of an argument.

Parameter

Argument: Numerical expression.

$0 \leqq$ argument

Explanation

Gives the square root of an argument.

SQR $\sqrt{x}$

ABS $\dfrac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function
Gives the absolute value of an argument.

Parameter
Argument: Numerical expression.

Explanation
Gives the absolute value of an argument.

ABS $|x|$

SGN $\dfrac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function
Gives a value that corresponds to the sign of an argument.

Parameter
Argument: Numerical expression.

Explanation
Gives a value that corresponds to the sign of an argument.

When an argument is positive,	1
When an argument is 0,	0
When an argument is negative,	– 1

INT $\dfrac{\text{Argument}}{\text{Numerical expression}}$ Ⓕ

Function
Gives the maximum integer that does not exceed an argument.

Parameter
Argument: Numerical expression.

Explanation

Gives the maximum integer that does not exceed an argument.

INT 12.56 → 12
INT −78.1 → −79

FRAC $\underset{\text{Numerical expression}}{\underline{\text{Argument}}}$ Ⓕ

Function

Gives the decimal part of an argument.

Parameter

Argument: Numerical expression.

Gives the decimal part of an argument. The sign is in accordance with the sign of the argument.

RND $\underset{\text{Numerical expression}}{\underline{\text{(Argument}}}$, $\underset{\text{Numerical expression}}{\underline{\text{digit location)}}}$ Ⓕ

Function

Gives the value of an argument which is rounded off at the specified location.

Parameter

Argument: Numerical expression.
Location: Numerical expression. Values below the decimal point are discarded.

−100 < location <100

Explanation

(1) Gives the value of an argument which is rounded off at the specified location.

(2) The argument is rounded off at the 3rd decimal place (10^{-3}).

→ **RND (*x*, −3)**

The argument is rounded off at the place of 100s (10^2).

→ **RND (*x*, 2)**

RAN # Ⓕ

Function

Gives a random number from 0 to 1.

Explanation

(1) Gives a random number from 0 to 1.

0 < random number < 1

(2) The random number has 10 digits.

Example

Provides a random number with 1 digit from 0—9.

```
INT (RAN# * 10)
```

Provides a random number with 1 digit from 1—5.

```
INT (RAN# * 5) + 1
```

Provides a random number with 2 digits from 10—99.

```
INT (RAN# * 90) + 10
```

DEG (Degree [, Minute [, Second]]) Ⓕ

Degree: Numerical expression; Minute: Numerical expression; Second: Numerical expression

Function

Converts sexagesimal to decimal.

Parameter

Degree: Numerical expression.
Minute: Numerical expression.
Second: Numerical expression.

|DEG (degree, minute, second)| < 10^{100}

Explanation

Converts sexagesimal expressed by degree, minute, and second to decimal.

Example

```
DEG(12,34,56) EXE        12.58222222
10 INPUT A,B,C
20 PRINT DEG(A,B,C)
30 END
```

DMS$ Argument / Numerical expression Ⓕ

Function

Converts decimal to sexagesimal.

Parameter

Argument: Numerical expression.

| numerical expression | < 10^{100}

Explanation

(1) Converts decimal to sexagesimal.
(2) The converted result is provided as a character string.

Example

```
DMS$(45.678) EXE         45°40'40.8
10 INPUT A
20 $=DMS$(A)
30 PRINT$
40 END
```

CHAPTER 5
PROGRAM LIBRARY

1. CONVERSION OF DECIMAL NUMBERS INTO HEXADECIMAL NUMBERS

This program can be used to convert decimal numbers into hexadecimal numbers and vice versa so that it is best for address calculation.

Examples)

1. What is the decimal equivalent of the hexadecimal 14AF?
2. What is the decimal equivalent of the hexadecimal A540?
3. What is the hexadecimal equivalent of the decimal 4582?
4. What is the hexadecimal equivalent of the decimal 18657030?

	Operation	**Display**
•Start the program first.	RUN EXE	`10→1-0-2→16`
•The menu will be displayed. To change a hexadecimal number to the decimal equivalent, press the 1 key. To convert a decimal number to the hexadecimal equivalent, press the 2 key. To return the menu, enter 0. Now the 1 key must be pressed.	1	`HEX?`
•Enter the hexadecimal data.	14AF EXE	`5295`
•Enter the next data.	EXE	`HEX?`
	A540 EXE	`42304`
•If "HEX" appears after the conversion of a hexadecimal number into the decimal equivalent, enter 0 to return the menu.	EXE	`HEX?`
	0 EXE	`10→1-0-2→16`
•Next, press the 2 key to convert the decimal number into the hexadecimal equivalent.	2	`DEC?`
•Enter the decimal data.	4582 EXE	`11E6`

•Enter the next data.

```
           EXE      DEC?
18657030   EXE      11CAF06
```

■ **Program List**

```
 10 FOR I=0 TO 5: R
    EAD A$(I): NEXT
     I
 20 DATA A,B,C,D,E,
    F
 30 PRINT "10→1-0-2
    →16";
 40 $= KEY$: IF $="
    " THEN 40
 50 PRINT
 60 ON VAL($) GOTO
    80,220
 70 GOTO 30
 80 INPUT "HEX",$
 90 IF $="0" THEN 3
    0
100 O= LEN($):R=0
110 FOR I=0 TO O-1
120 P$= MID$(O-I,1)
130 IF P$≤"9" THEN
    Q= VAL(P$): GOT
    O 180
140 FOR J=0 TO 5
150 IF A$(J)=P$ THE
    N Q=J+10: GOTO
    180
160 NEXT J
170 GOTO 100
180 R=R+16↑I*Q
190 NEXT I
200 PRINT R
210 GOTO 80
220 INPUT "DEC",$
230 IF $="0" THEN 3
    0
240 S= VAL($):P=S:Q
    =0:$=""
250 O= FRAC(S/16)*1
    6:S=S-O
260 IF P<16 THEN 29
    0
270 P= INT(P/16):Q=
    Q+1
280 GOTO 260
290 IF Q=0 THEN 350
300 FOR I=Q TO 1 ST
    EP -1
310 P= INT(S/16↑I)
320 Z=P: GOSUB 500
330 S= INT( FRAC(S/
    16↑I)*16↑I+.5)
340 NEXT I
350 Z=O: GOSUB 500
360 PRINT $
370 GOTO 220
500 IF Z>9 THEN $=$
    +A$(Z-10): RETU
    RN
510 $=$+ STR$(Z): R
    ETURN
```

Total 485 steps

2. LOAN CALCULATIONS (EQUAL MONTHLY INSTALLMENTS)

This program is used to calculate one of the following three data, monthly payment, amount of loan, and number of installments, when annual interest rate and two other data are known.

Input		Display
•Annual interest rate, monthly payment, and number of installments	$\longrightarrow$	Amount of loan
•Annual interest rate, monthly payment, and amount of loan	$\longrightarrow$	Number of installments
•Annual interest rate, amount of loan, and number of installments	$\longrightarrow$	Monthly payment

To obtain the desired data, enter 0 when input of desired data is requested.
Enter annual interest rate in percent.
The monthly payment and amount of loan are displayed in dollars, and the number of installments is in months. The calculation results are rounded.

Formulas)

P: Monthly payment PV: Amount of loan
i: Monthly interest rate (Actually the annual interest rate is entered)
n: Number of installments

$$P = PV\frac{i}{1-(1+i)^{-n}}$$

$$PV = P\frac{1-(1+i)^{-n}}{i}$$

$$n = -\frac{ln\left(1-\frac{i \cdot PV}{P}\right)}{ln(1+i)}$$

Examples)

1. If a loan is given on condition that payment is made in monthly installments of \$200 for 24 months at an annual interest rate of 7.5%, what will be the amount of the loan?
2. If a loan of \$5,000 is given at an annual interest rate of 6.5% in 36 (three years) installments, what will be the monthly payment?
3. If a loan of \$25,000 is given at an annual interest rate of 5.8% and in installments of \$250 per month, what will be the number of installments?

	Operation	**Display**
•Start the program first.		
	RUN EXE	Payment?
•Enter the monthly payment. In Example 1 above, the amount is $200.		
	200 EXE	Loan?
•In this example, the purpose of the problem is to obtain the amount of loan. So enter 0.		
	0 EXE	Rate?
•Enter the annual interest rate.		
	7.5 EXE	Months?
•Enter the number of installments.		
	24 EXE	Loan 4444
		* The amount of loan is displayed.
•Let's try Example 2.		
	RUN EXE	Payment?
	0 EXE	Loan?
•The amount of loan is $5,000.		
	5000 EXE	Rate?
•The annual interest rate is 6.5%.		
	6.5 EXE	Months?
•The number of installments is 36.		
	36 EXE	Payment 153
		*The monthly payment is displayed.
•What about Example 3?		
	RUN EXE	Payment?

•The monthly payment is $250.

250 EXE

```
Loan?
```

•The amount of loan is $25,000.

25000 EXE

```
Rate?
```

•The annual interest rate is 5.8%.

5.8 EXE

```
Months?
```

•To obtain the number of installments, so enter 0.

0 EXE

```
Months 137
```

*The number of months is displayed.

■ **Program List**

```
 10 CLEAR
 20 RESTORE
 30 DATA Payment,Lo
    an,Rate,Months
 40 FOR I=0 TO 3
 50 READ E$(I): PRI
    NT E$(I);
 60 INPUT X
 70 IF X≠0 THEN A(I
    )= X:Y=Y+1
 80 NEXT I
 90 IF Y<3 THEN 10
100 K=C/1200
110 FOR I=0 TO 3
120 IF A(I)=0 THEN
    150
130 NEXT I
140 GOTO 10
150 ON I GOTO 170,1
    40,180
160 A= INT(B*K/(1-(
    1+K)↑-D)+.5): G
    OTO 200
170 B= INT(A*(1-(1+
    K)↑-D)/K+.5): G
    OTO 200
180 D= INT(- LN(1-K
    *B/A)/ LN(1+K)+
    .5)
200 PRINT E$(I);A(I
    )
```

Total 272 steps

3. ARITHMETIC EXERCISE

This program presents arithmetical problems of addition, subtraction, multiplication, and division.
In four operations, it is necessary to select to do calculations either in one digit or in two digits.
Ten problems makes a set and a point of 10 is given per problem.
Quotients will be scored whether they are correct, up to two decimal places, or not.

Operation	Display	
RUN EXE	Push + - * /	······Press +, -, × or ÷ for addition, subtraction, multiplication or division.
+	1 or 2	······Select either 1 digit or 2 digits.
1	<1>3+9=?	······Problem 1
12 EXE	<2>2+7=?	······Problem 2
9 EXE	<3>2+6=?	······Problem 3
	⋮	
	<10>6+7=?	······Problem 10
13 EXE	SCORE:100	······Points indication

■ Program List

```
10 E=0
20 PRINT "Push + -
   * /";
30 K$= KEY$
40 IF K$="+" THEN
   M=1: GOTO 100
50 IF K$="-" THEN
   M=2: GOTO 100
60 IF K$="*" THEN
   M=3: GOTO 100
70 IF K$="/" THEN
   M=4: GOTO 100
80 GOTO 30
100 PRINT : PRINT "
   1 or 2";
110 K$= KEY$
120 IF K$="1" THEN
   N=9: GOTO 160
130 IF K$="2" THEN
   N=99: GOTO 160
140 GOTO 110
160 FOR I=1 TO 10
170 PRINT
180 A= INT( RAN#*N+
   1)
190 B= INT( RAN#*N+
   1)
200 ON M GOSUB 300,
   400,500,600
210 PRINT "<"; STR$
   (I);">";$;: INP
   UT D
220 D= INT(D*100)/1
   00
230 IF C=D THEN E=E
   +1
240 NEXT I
250 PRINT "SCORE:";
   E*10;
260 IF E=10 THEN FO
   R I=1 TO 10: NE
   XT I
270 END
300 $= STR$(A)+"+"+
   STR$(B)+"="
310 C=A+B
320 RETURN
400 IF A<B THEN F=A
   :A=B:B=F
410 $= STR$(A)+"-"+
   STR$(B)+"="
420 C=A-B
430 RETURN
500 $= STR$(A)+"x"+
   STR$(B)+"="
510 C=A*B
520 RETURN
600 IF A<B THEN F=A
   :A=B:B=F
610 $= STR$(A)+"÷"+
   STR$(B)+"="
620 C= INT(A/B*100)
   /100
630 RETURN
```

Total 519 steps

4. QUADRATIC REGRESSION ANALYSIS

This program is used to obtain an estimate of y through a quadratic regression analysis of data x and y.
Calculation is performed on the basis of $y=ax^2+bx+c$ using the following formulas:

$$a=\frac{\Sigma(x^2\cdot y)\cdot\Sigma(x\cdot x)-\Sigma(x\cdot y)\cdot\Sigma(x\cdot x^2)}{\Sigma(x\cdot x)\Sigma(x^2\cdot x^2)-\{\Sigma(x\cdot x^2)\}^2}$$

$$b=\frac{\Sigma(x\cdot y)\Sigma(x^2\cdot x^2)-\Sigma(x^2\cdot y)\Sigma(x\cdot x^2)}{\Sigma(x\cdot x)\cdot\Sigma(x^2\cdot x^2)-\{\Sigma(x\cdot x^2)\}^2}$$

$$c=\frac{\Sigma y_i}{n}-b\frac{\Sigma x_i}{n}-a\frac{\Sigma x_i{}^2}{n}$$

The program in P0 is used for data input. Enter data x and y, and obtain Σx, Σy, Σx^2, Σxy, Σx^3, Σx^4, and $\Sigma x^2 \cdot y$.
The program in P1 is used to calculate a, b and c from the Σx, Σy, Σx^2, Σxy, Σx^3, Σx^4, and $\Sigma x^2 \cdot y$ obtained using the program in P0. Then, obtain an estimate of y corresponding to entered x.

Example)

	1	2	3	4	5	6	7
x	1	5	8	11	15	18	22
y	21	30	41	54	70	①	②

Through a regression analysis of the above data, obtain the estimates of ① and ②.

Operation	**Display**
SHIFT P0	x 1?
1 EXE	y 1?
21 EXE	x 2?
5 EXE	y 2?
Enter the following data in order.	⋮
	y 5?
70 EXE	x 6?

After entering all data, execute the program in P1.

Operation	Display	
SHIFT P1	a= 0.08967	.... a
EXE	b= 2.14288	.... b
EXE	c= 18.23781	 c
EXE	x=?	
18 EXE	y= 85.86245	.... Estimate of y
EXE	x=?	
22 EXE	y= 108.78103	.... Estimate of y

Using the program in P1 in this way, a, b, c, and the estimate of y can be obtained.
Here, the calculation results are rounded off to five decimal places, but if rounding is unnecessary or rounding-off to another decimal place is required, line No. 90 in P1 must be changed.

■ Program List

```
P0
  10 CLEAR
  20 PRINT "x";M+1;:
      INPUT X
  30 PRINT "y";M+1;:
      INPUT Y
  40 N=N+X:O=O+Y
  50 P=P+X↑2:R=R+X*Y
  60 A=A+X↑3:B=B+X↑4
     :C=C+X↑2*Y
  70 M=M+1
  80 GOTO 20
```

```
P1
  10 D=P-N↑2/M
  20 E=R-N*O/M
  30 F=A-N*P/M
  40 G=C-P*O/M
  50 H=B-P↑2/M
  60 I=(G*D-E*F)/(D*
     H-F↑2)
  70 J=(E*H-G*F)/(D*
     H-F↑2)
  80 K=O/M-J*N/M-I*P
     /M
  90 SET F5
 100 PRINT "a=";I,"b
     =";J,"c=";K
 110 INPUT "x=",X
 120 Y=I*X↑2+J*X+K
 130 PRINT "y=";Y
 140 GOTO 110
```

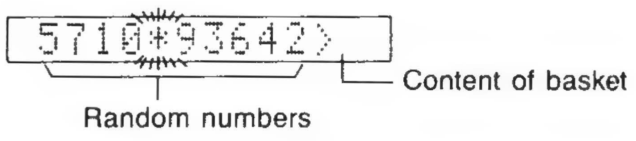
Total 309 steps

Variable contents	
A	Σx^3
B	Σx^4
C	$x^2 \cdot y$
I	a
J	b
K	c
N	Σx
M	Number of data
O	Σy
P	Σx^2
R	$\Sigma x \cdot y$

5. REARRANGEMENT GAME

This game offers amusement in arranging random numbers 0 to 9 in numerical order (0123456789) as quickly as possible.
To play the game, the $\boxed{4}$, $\boxed{6}$ and $\boxed{+}$ keys are used. Pressing the $\boxed{4}$ key moves the flashing asterisk one position to left, and pressing the $\boxed{6}$ key moves it one position to right. When the $\boxed{+}$ key is pressed, the number on the flashing asterisk is replaced by the number in the basket and in turn, the number on the flashing asterisk is moved into the basket.

Reading the display



Since the score represents the time taken for rearrangement, the smaller the figure, the better the score. When you play the game for the first time or after other program was executed, be sure to enter CLEAR $\boxed{\text{EXE}}$.

Operation	Display	
CLEAR $\boxed{\text{EXE}}$		
RUN $\boxed{\text{EXE}}$	Hi-Sc:0	········ High score is displayed.
	<<WAIT!>>	········ Problem is being prepared.
	3721*94650>	
$\boxed{+}$	3721*94650>8	········ Move 8 into the basket.
$\boxed{6}\boxed{6}\boxed{6}\boxed{6}$	3721 946*0>8	········ Move the asterisk to where it should be.
$\boxed{+}$	3721 946*0>5	········ Replace 5 with 8.
$\boxed{4}\boxed{4}\boxed{4}$	3721 *4680>5	········ Move the asterisk to 9.
$\boxed{+}$	3721 *4680>9	········ Replace 5 with 9.
Repeat the above operations.	⋮	
	0123*56789>4	········ Move 4 to where it should be.
$\boxed{+}$	[0123456789]	········ End with success
	SCORE 132	········ Score

When continuing the game, enter RUN $\boxed{\text{EXE}}$.

***What is your level of skills?**

Score	Evaluation
Under 50	You are genius! Also very lucky.
50—69	You have excellent reflexes.
70—99	About the average. Just a bit of more practice is needed for higher levels.
100—149	You are still out of practice.
150 or over	Something wrong with you. Try once more.

■ Program List

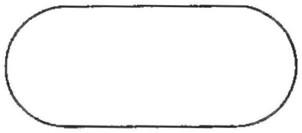
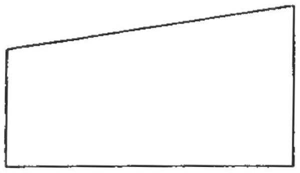
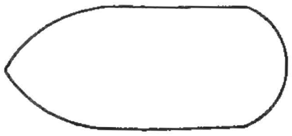
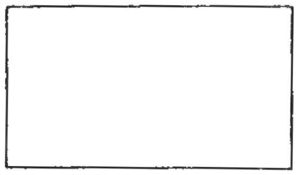
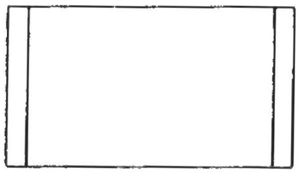
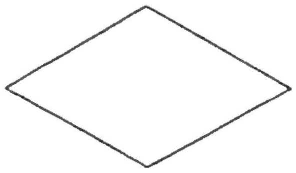
```
 10 PRINT "Hi-Sc:";
    H; : FOR I=1 TO
    200: NEXT I
    T I
 20 $=""
 30 PRINT : PRINT "
    << WAIT ! >>";
 40 A$= STR$( INT(
    RAN#*10))
 50 FOR I=1 TO LEN(
    $)
 60 IF A$= MID$(I,1
    ) THEN 40
 70 NEXT I
 80 $=$+A$
 90 IF LEN($)<10 TH
    EN 40
100 PRINT
110 X=5:B$=" ":N=0
120 PRINT CSR0;$;">
    ";B$;
130 PRINT CSRX;"*";
140 K$= KEY$
150 IF K$="4" THEN
    X=X-1: IF X<0
    THEN X=0
160 IF K$="6" THEN
    X=X+1: IF X>9
    THEN X=9
170 IF K$="+" THEN
    GOSUB 510
180 N=N+1
190 IF $≠"012345678
    9" THEN 120
200 IF H=0 THEN H=N
210 IF H>N THEN H=N
220 PRINT : PRINT "
    [";$;"]";
230 FOR I=1 TO 200
250 NEXT I
260 PRINT
270 PRINT "SCORE";N
    ;
280 END
510 IF X=0 THEN C$=
     MID$(1,1):$=B$
    + MID$(2): GOTO
     540
520 C$= MID$(X+1,1)
530 $= MID$(1,X)+B$
    + MID$(X+2)
540 B$=C$
550 RETURN
```

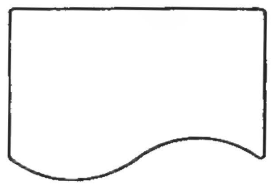
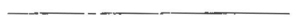
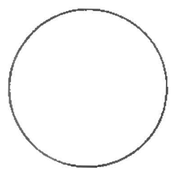
Total 424 steps

CHAPTER 6

REFERENCE MATERIAL

6-3. FLOWCHART SYMBOLS

Symbol	Meaning
	Entry or exit point (start, return, end, etc.)
	Data input from the keyboard
	Output function
	General processing
	Processing in a subroutine
	Test (condition)

Symbol	Meaning
	Output to a printer
	Flowline
	Transfer or continuation point

COMMAND/FUNCTION INDEX

6-4. ARRAY VARIABLE TABLE

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
B	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
C	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
D	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
E	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
F	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
G	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
J	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12	13
N	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11	12
O	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10	11
P	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9	10
Q	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8	9
R	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7	8
S	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6	7
T	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5	6
U	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4	5
V	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3	4
W	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2	3
X	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1	2
Y	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	1
Z	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

This table indicates the relations between variables.

Example) H(0)~H(9)→H~Q

SPECIFICATIONS

- **Type**
  PB-100F
- **Fundamental calculation functions**
  Negative numbers, exponents, parenthetical addition, subtraction, multiplication and division (with priority sequence judgement function (true algebraic logic))
- **Built-in functions**
  Trigonometric/inverse trigonometric functions (angular units — degree/radian/grade), logarithmic/exponential functions, square roots, powers, conversion to integer, deletion of integer portion, absolute value, symbolization, designation of number of significant digits, designation of number of decimal digits, random numbers, π, decimal ↔ sexagesimal conversion.
- **Commands**
  INPUT, PRINT, GOTO, ON-GOTO, FOR-NEXT, IF-THEN, GOSUB, ON-GOSUB, RETURN, READ, DATA, RESTORE, STOP, END, REM, LET, PASS, RUN, LIST, LIST ALL, MODE, SET, CLEAR, NEW, NEW ALL, DEFM, SAVE, SAVE ALL, LOAD, LOAD ALL, PUT, GET, VERIFY.
- **Program functions**
  KEY$, CSR, LEN, MID$, VAL, STR$
- **Calculation range**
  $\pm 1 \times 10^{-99}$ to $\pm 9.999999999 \times 10^{99}$ and 0 (internal calculations use 12-digit mantissa)
- **Program system**
  Stored system
- **Number of steps**
  Maximum 544 steps (maximum 1,568 steps when optional RAM pack is loaded)
- **Program capacity**
  Maximum 10 programs (P0 through P9)
- **Number of variables**
  Standard 26, expandable to 94 (maximum 222 variables when optional RAM pack is loaded) and exclusive character variable ($)
- **Nesting**
  Subroutine — 8 levels
  FOR-NEXT loop — 4 levels
  Numerical value — 6 levels
  Operators — 12 levels
- **Display system and contents**
  10-digit mantissa (including minus sign) or 8-digit mantissa (7 digits for negative number) and 2-digit exponent.

- **Display elements**
  12-digit dot matrix display (liquid crystal)
- **Main components**
  C-MOS VLSI and others
- **Power supply**
  2 lithium batteries (CR2032)
- **Power consumption**
  Maximum 0.02 W
- **Battery life (Continuous use)**
  Mainframe only — approximately 170 hours
  With options connected — approximately 100 hours
- **Auto power-off**
  Power is turned off automatically approximately 7 minutes after last operation.
- **Ambient temperature range**
  0°C to 40°C (32°F to 104°F)
- **Dimensions**
  9.8H × 165W × 71mmD (3/8"H × 6 1/2"W × 2 3/4"D)
- **Weight**
  119g (4.2 oz) including batteries

CASIO®

025B SA ※ Printed in Japan