PROGRAMMABLE CALCULATOR

CASIO FX-700P

INSTRUCTION MANUAL

Ⓑ FC ㊮

First of all, we would like to thank you very much for purchasing this product. This instrument is a high performance, programmable calculator which incorporates microcircuitry to provide for repetitious or complex calculation. The most important feature of this instrument is that it uses BASIC program language. This provides a conversation type language for problem solving. Operation is easy, even for a beginner. Additionally, programming is simple using one key commands which permit highly efficient keying.

This instrument's calculation management functions are generally separated as follows.

1. Manual Calculation
2. Program Calculation

It not only performs high level program calculations like a computer but is also designed for easy operation as a scientific calculator.

CONTENTS

This instruction manual explains how to use the calculator. Please read it thoroughly before using. Master each function and use it with care to provide long life.

Prior to Use

This calculator is brought to you as a result of our highly developed electronic technology. Strict quality control procedures and a rigid inspection process were employed in its production. Please follow the precautions below to ensure long equipment life.

■ Use Precautions

- ● This calculator is composed of precision electronic parts. Never try to take it apart. Avoid dropping or throwing. Do not permit it to undergo extreme temperature variations. Do not store or leave in any location where there is dampness, high temperature or dust. During periods of low temperature, the display response may be slower or there may be no display. Normal display will resume when the temperature returns to normal.
- ● Do not use any other equipments than the optional ones.
- ● While the calculator is performing calculations, a "–" will be displayed. Key operation during this time will not be effective except for one section so please pay attention to the display at all times and press the keys carefully.
- ● Concerning the batteries, even when the calculator is not used to any great extent, please change the batteries every 2 years. If worn out batteries are used, they may leak and cause damage to the instrument. Never leave worn out batteries in the instrument.
- ● To clean the calculator, use a soft, dry cloth or a damp cloth and mild detergent to wipe it off. Never use paint thinner or benzine.
- ● In case of malfunction, contact the store where it was purchased or a nearby dealer.
- ● Prior to seeking maintenance, please read the instruction manual again and also check the power supply as well as program or operational error.

■ Power Supply and Battery Replacement

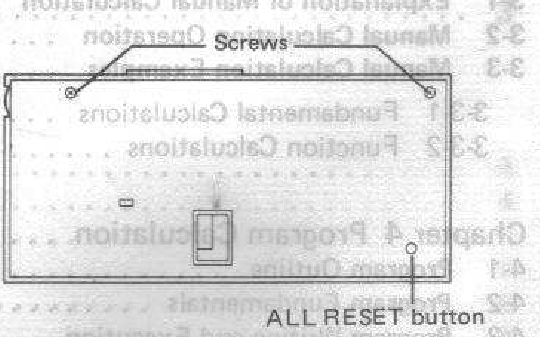
This instrument uses two lithium batteries (CR2032) for a power supply.

If the contrast is weak even when the contrast control is adjusted for maximum contrast (refer to page 8), this means that the batteries are worn out. Therefore, please replace the batteries at the earliest opportunity using the following procedure.

Furtheremore, even though the instrument is functioning normally, be sure to replace the batteries every 2 years.

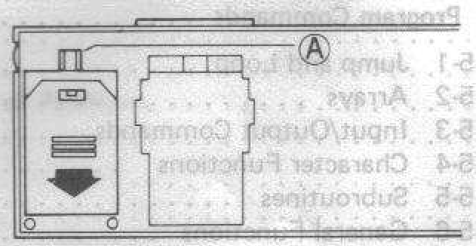
● How to Replace the Batteries

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



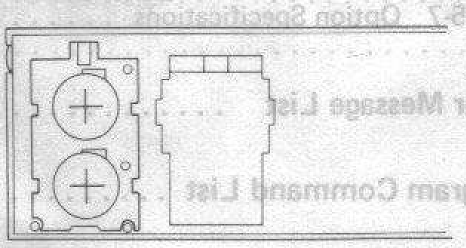
(After replacing the batteries, press with a pointed object.)

(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 instrument 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.



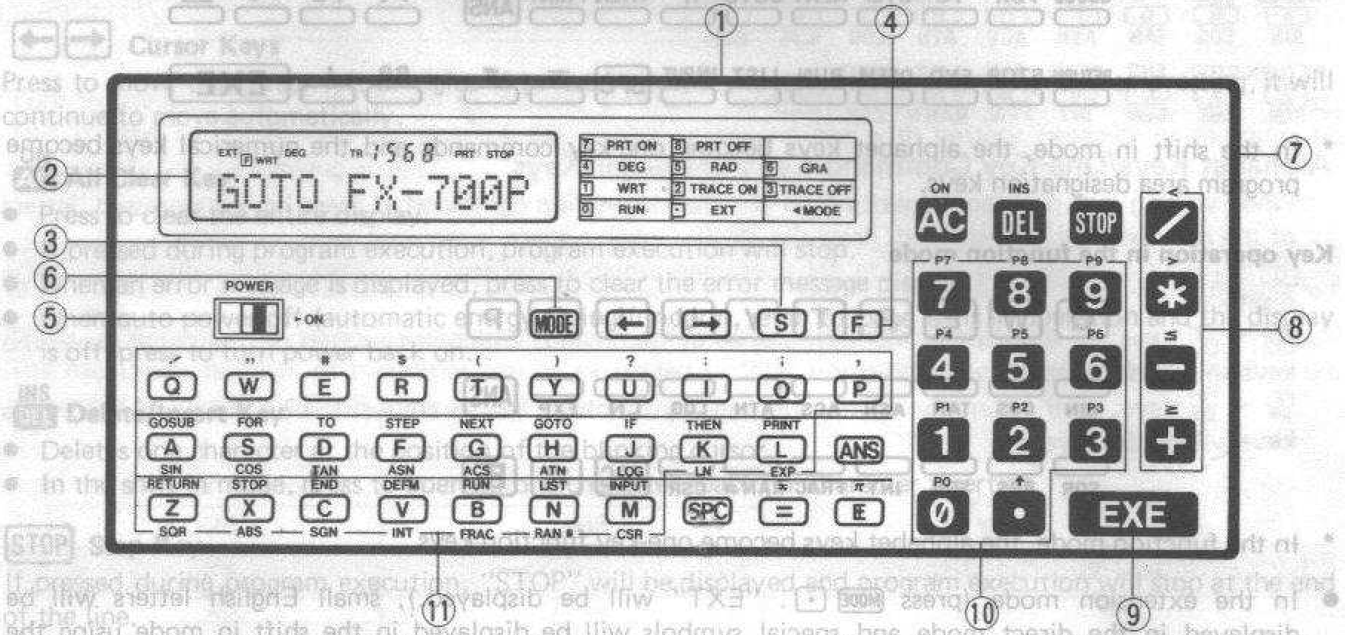
(5) While pressing the batteries down with the battery compartment lid, slide it closed.

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

* Be sure to replace both batteries.
* Never throw the old batteries into a fire. This is very dangerous as they might explode.
* Be sure to position the ⊕ and ⊖ terminals correctly.

Chapter 1

Each Section's Nomenclature and Operation



①Adaptor connector
②Display window
③Shift key
④Function key
⑤Power switch
⑥Mode key
⑦Display contrast control
⑧Calculation command keys
⑨Execute key
⑩Numerical keys and decimal point key
⑪Alphabet keys

1-1 Each Section's Nomenclature

Each key has 3 separate operations.

Press the keys directly for the function printed on the key. Press Ⓢ and then the key for the function printed above the key. Press Ⓕ and then the key for the function printed below the key.

Example

GOSUB . . .Shift in mode

A . . .Direct mode

SIN . . .Function mode

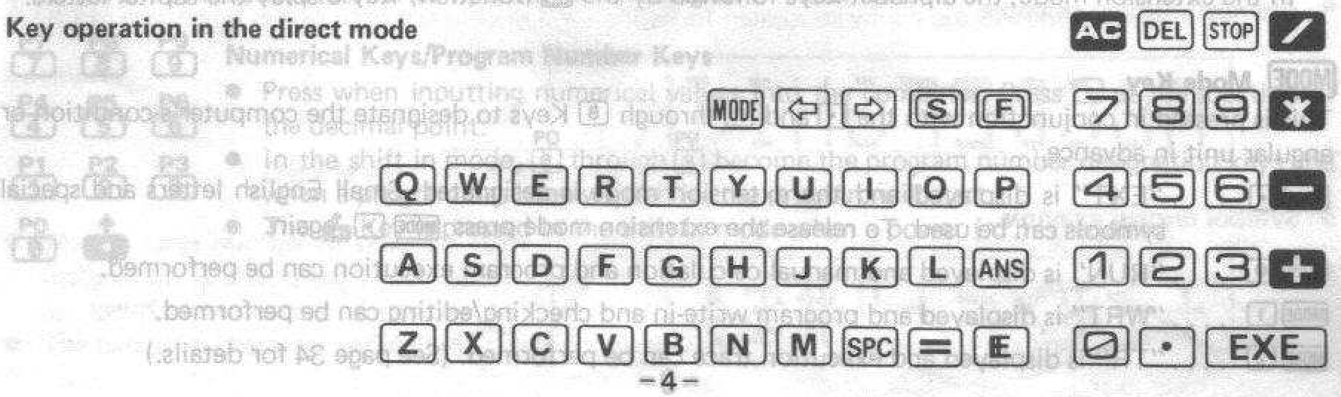
S Shift Key (symbolized by Ⓢ hereafter)

If this key is pressed, the shift in mode is selected ("Ⓢ" is displayed) and the function printed above each key can be used.

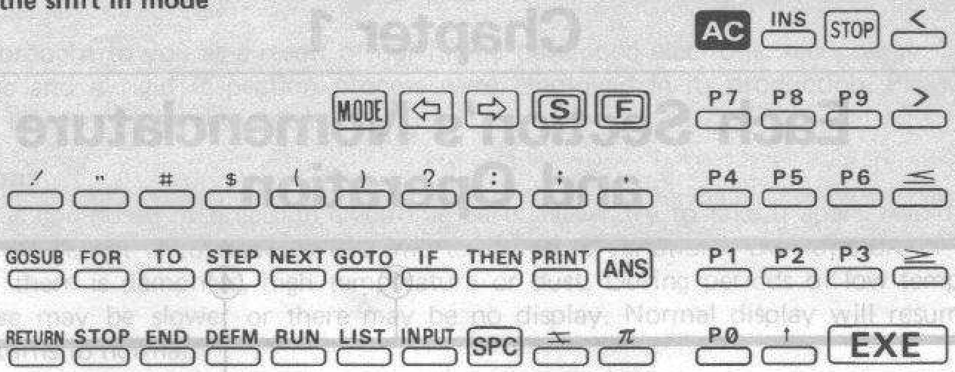
F Function Key (symbolized by Ⓕ hereafter)

If this key is pressed, the function mode is selected ("Ⓕ" is displayed) and the function printed below each key can be used.

Key operation in the direct mode

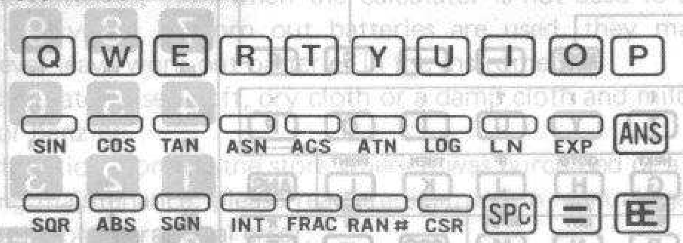


Key operation in the shift in mode



* In the shift in mode, the alphabet keys become one-key commands and the numerical keys become program area designation keys.

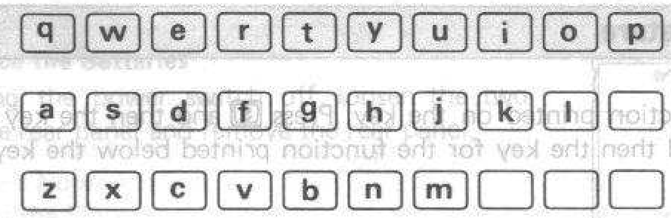
Key operation in the function mode



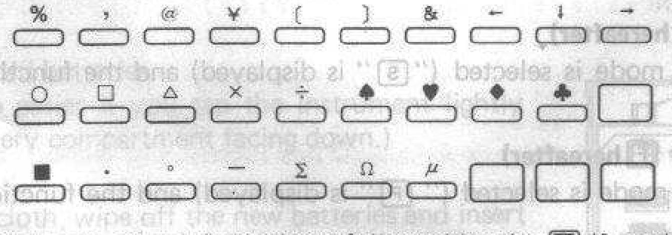
* In the function mode, the alphabet keys become one-key function keys.

- In the extension mode (press MODE ·. "EXT" will be displayed.), small English letters will be displayed in the direct mode and special symbols will be displayed in the shift in mode using the alphabet keys.

Direct mode using the extension mode



Shift in mode using the extension mode



* In the extension mode, the alphabet keys followed by the F (function) Key display the capital letters.

MODE Mode Key

This is pressed in conjunction with the · and 0 through 8 Keys to designate the computer's condition or angular unit in advance.

MODE · . . . . "EXT" is displayed and the extension mode is designated. Small English letters and special symbols can be used. To release the extension mode press MODE · again.

MODE 0 . . . . "RUN" is displayed and manual calculation and program execution can be performed.

MODE 1 . . . . "WRT" is displayed and program write-in and checking/editing can be performed.

MODE 2 . . . . "TR" is displayed and execution trace can be performed. (See page 34 for details.)

[MODE] [3] ....When "TR" is displayed, it will be extinguished and the execution trace function will be released.

[MODE] [4] ...."DEG" is displayed and the angular unit will be designated as "degree".

[MODE] [5] ...."RAD" is displayed and the angular unit will be disignated as "radian".

[MODE] [6] ...."GRA" is displayed and the angular unit will be designated as "gradient".

[MODE] [7] ...."PRT" is displayed and if a printer is connected, printout can be performed.

[MODE] [8] ....When "PRT" is displayed, it will be extinguished and the printout function will be released.

[←] [→] Cursor Keys

Press to move the cursor left or right. If pressed once, it moves one character. If you keep pressing, it will continue to move automatically.

ON [AC] All Clear Key

- Press to clear the entire display.
- If pressed during program execution, program execution will stop.
- When an error message is displayed, press to clear the error message display.
- When auto power off (automatic energy saving function, refer to page 9) is in operation and the display is off, press to turn power back on.

INS [DEL] Delete/Insert Key

- Deletes one character at the position of the blinking cursor.
- In the shift in mode, press to open up one character space for character insertion.

[STOP] Stop Key

If pressed during program execution, "STOP" will be displayed and program execution will stop at the end of the line.

During execution trace with "STOP" on the display, this key will display the program area number and the line number.

[EXE] Execute Key

- When the result of a manual calculation is required, press instead of "=".
- In the "WRT" mode, when writing in a program, press to write (store) each line in the computer. If this key is not pressed, nothing will be written in.
- In the "RUN" mode, press for data input during program execution or press to continue program execution while "STOP" is displayed.

[ANS] Answer Key

For manual calculation, press to call out the calculation result (answer) of the previous calculation.

π [E] Exponent/Pi Key

When inputting exponential value, press after inputting the mantissa portion.

Example: 2.56×10^{34} → [2] [.] [5] [6] [E] [3] [4]

* The exponential portion may be a maximum of ±99. If this is exceeded, an error will occur.

≒ [=] Equal Key/Comparison Key

- Press when using a substitution statement or for comparison when using an IF statement (equal sign).
- In the shift in mode, press for comparison when using an IF statement.

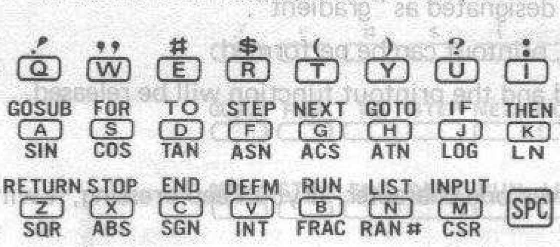
P7 [7] P8 [8] P9 [9]
P4 [4] P5 [5] P6 [6]
P1 [1] P2 [2] P3 [3]
P0 [0] [↑]

Numerical Keys/Program Number Keys

- Press when inputting numerical values into the computer. Press [.] at the location of the decimal point.
- In the shift in mode, P0 [0] through P9 [9] become the program number designation keys and when a program has been written in, the program will start.
- The [↑] Key is pressed in the shift in mode when a power (x^y) is required.

≥ ≤ > < + − * / Calculation Command Keys/Comparison Keys

- When performing addition, subtraction, multiplication and division, press at the respective locations.
  * is used for multiplication (corresponds to "x").
  / is used for division (corresponds to ÷).
- In the shift in mode, press for comparison of a judgement in an IF statement.



' " # $ ( ) ? : ; ,
Q W E R T Y U I O P
GOSUB FOR TO STEP NEXT GOTO IF THEN PRINT
A S D F G H J K L
SIN COS TAN ASN ACS ATN LOG LN EXP
RETURN STOP END DEFM RUN LIST INPUT
Z X C V B N M SPC
SQR ABS SGN INT FRAC RAN# CSR

Alphabet Keys/One-Key Command Keys/Character Keys

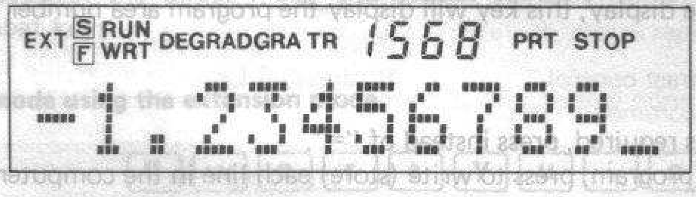
When writing in a program or writing an command/function command, if these keys are pressed, letters of the alphabet will be displayed. Press the SPC Key when a space is required.

Q ~ P Keys: In the shift in mode, the characters which are written on the panel above the keys will be displayed.

GOSUB A ~ INPUT M Keys: In the shift in mode, the one-key commands which are written on the panel above the keys will be displayed.

A SIN ~ M CSR Keys: In the function mode, the one-key functions which are written on the panel below the keys will be displayed.

1-2 How to Read the Display



Displays the calculation value or result. The respective display positions are composed of 5 horizontal and 7 vertical dots. Up to a maximum of 12 positions are available for display of numbers or characters. (Zero is displayed as 0.) If a formula or statement exceeds 12 positions, the numbers or characters will move to the left and up to a maximum of 62 characters can be input.

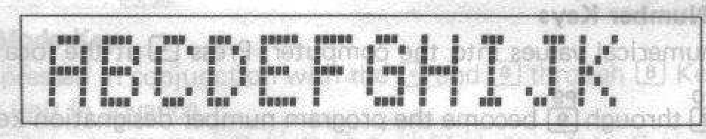
The blinking cursor is displayed until 55 characters have been input. From the 56th character on, a blinking "■" will be displayed instead.

A 4-position numerical display is available on the upper portion of the display to indicate the number of remaining steps.

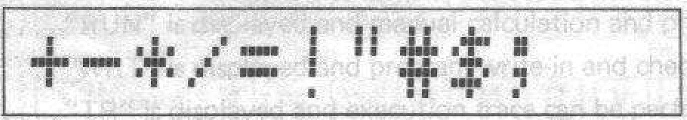
Furthermore, during operation, a "–" will be displayed in the rightmost position of the 4-position display on the upper portion of the display.

Also, various symbols such as "DEG", "RAD" and "GRA" for angular units, "S" (when the S Key is pressed), "F" (when the F Key is pressed), "RUN" (RUN mode), "WRT" (WRT mode), "TR" (TR mode), "PRT" (PRT mode) and "STOP" will be displayed to indicate the respective situation.

- **Alphabet display example**

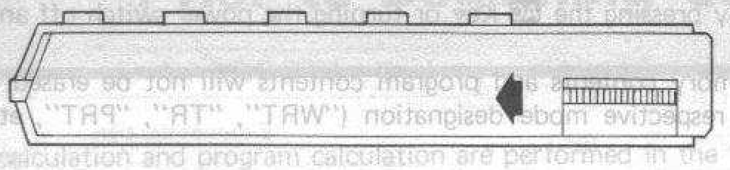


- **Symbol display example**



1-3 Contrast Adjustment

Adjustment of display contrast can be performed using the adjustment control located on the right side of the instrument.



Turn in the direction of the arrow to increase contrast. Turn in the opposite direction to reduce contrast. This is used to compensate contrast of the display in accordance with battery capacity or to adjust to compensate for the viewing angle.

1-4 Memory Expansion

There are normally 26 memories (variables). The number of steps at this time is 1568.
The maximum number of memories 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
26	1568
27	1560
28	1552
⋮	⋮
46	1408
⋮	⋮
94	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).

```
DEFM30 EXE     ***VAR:56
```

* DEFM can be input by pressing D E F M or by pressing S DEFM V .

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 Auto Power Off

This is an automatic energy-saving function which prevents wasted power consumption when you forget to turn off the power switch. Approximately 7 minutes after the last key operation (except during program execution), power will go off automatically.

In this case, power can be resumed by pressing the AC Key or turning the power switch off and then on again.

* Even if power is turned off, memory contents and program contents will not be erased. However, angular unit designation and the respective mode designation ("WRT", "TR", "PRT", etc.) are all released.

Chapter 2

Prior to Beginning Calculation

Manual calculation and program calculation are performed in the "RUN" mode. (Press $\boxed{\text{MODE}}\boxed{0}$ and RUN will be displayed.)
Furthermore, with respect to "DEG", "RAD" and "GRA", since these only apply to angular unit, the display of these has no effect for a calculation which has nothing to do with angular unit.

2-1 Calculation Priority Sequence (True Algebraic Logic)

This unit determines the calculation priority sequence internally and will perform calculations based on that sequence.
The calculation priority sequence is determined as follows.

① Functions (SIN, COS, TAN, etc.)
② Power
③ Multiplication and division (✱and /)
④ Addition and subtraction (+ and −)

When the priority is the same, calculation will begin from the left.
Furthermore, when parentheses are used, these have the highest priority.

Example:

$$\underbrace{2+\underbrace{3*\underbrace{\underbrace{\text{SIN}(\underbrace{17+13}_{①})}_{②}\uparrow 2}_{③}}_{④}}_{⑤}=2.75$$

2-2 Input/Output Number of Positions and Operation Number of Positions

The number of input positions for this unit are 12 positions for the mantissa portion and 2 positions for the exponential portion. Internal operations are also performed using 12 positions for the mantissa portion and 2 positions for the exponential portion.
The range is 1×10^{-99} — $\pm 9.99999999999 \times 10^{+99}$ and 0.
The number of output positions is 10 positions for the mantissa portion and 2 positions for the exponential portion. However, if an exponential portion is attached, the mantissa portion will be 8 positions.

* For function results, etc., when the number of display positions (12 positions) is exceeded, up to 12 positions will be displayed, including the 0 and the decimal point.

Example:

$(1\times10^5)\div7=14285.71429$
$(1\times10^5)\div7-14285=0.7142857$

$1\boxed{\text{EXP}}5\boxed{\div}7\boxed{\text{EXE}}$	14285.71429
$1\boxed{\text{EXP}}5\boxed{\div}7\boxed{-}14285\boxed{\text{EXE}}$	0.7142857

When the calculation result exceeds 10^{10} (10,000,000,000) or is below 10^{-3} (0.001), it is automatically displayed using an exponential display.

Example:

$1234567890 \times 10 = 12345678900$
$(= 1.23456789 \times 10^{10})$

1234567890 ✱ 10 EXE | 1.2345678E10

Exponential sign

* The exponential portion is displayed along with an exponential sign following the mantissa portion.

Example:

$1.234 \div 10000 = 0.0001234$
$(= 1.234 \times 10^{-4})$

1.234 ⁄ 10000 EXE | 1.234E⁻04

2-3 Basic Calculation

2-3-1 Calculation Symbols and Function Commands

Using BASIC calculation symbols, + and − symbols are the same as standard symbols for addition and subtraction but ✱ and / are used for multiplication and division symbols.

For instance:

2+3−4×5÷6

is expressed as

2+3−4✱5/6

Also, this instrument incorporates the following functions.

Function Name		Form	
Trigonometric Function	$\sin x$	SINx	[F A SIN]
	$\cos x$	COSx	[F S COS]
	$\tan x$	TANx	[F D TAN]
Inverse Trigonometric Function	$\sin^{-1} x$	ASNx	[F F ASN]
	$\cos^{-1} x$	ACSx	[F G ACS]
	$\tan^{-1} x$	ATNx	[F H ATN]
Square Root	$\sqrt{x}$	SQRx	[F Z SQR]
Exponential Function	e	EXP1★	[F L EXP]
Natural Logarithm	$\ln x$	LNx	[F K LN]
Common Logarithm	$\log x$	LOGx	[F J LOG]
Change to Integer	INTx	INTx	[F V INT]
Cancel Integer Portion	FRACx	FRACx	[F B FRAC]
Change to Absolute Value	$\lvert x \rvert$	ABSx	[F X ABS]
Change to Symbol	Positive Number → 1 0 → 0 Negative Number → −1	SGNx	[F C SGN]
Rounding Off	(10^y of x is rounded off)	RND (x, y)	[—]
Random Numbers		RAN #	[F N RAN#]

* In the case of the RND function, the argument must be enclosed in parentheses.

★ EXP is a command to call out the numerical value of the exponential table.

2-3-2 Previous Calculation Result Callout

The result of a manual or program calculation is stored until the next calculation has been executed. This result can be displayed by pressing the ANS Key.

Example: 741+852=1593
2431−1593=838

Operation:

741+852	741+852
EXE	1593
2431−ANS	2431− 1593
EXE	838

Also, the number value displayed following the calculation can be used as is in the next calculation.

Example: (Subsequently to the above)
838×2=1676

Operation:

*2	838*2
EXE	1676

2-3-3 Error Message

If the formula or substitute sentence is not written correctly according to BASIC language or if the calculation range is exceeded, an error will occur at the time of run and an error message will be displayed. Concerning power ($x\uparrow y$) however, when y is a natural number, an error message will not be displayed even if x is smaller than 0 (zero). The following error messages will be displayed during manual calculation.

The following error messages will be displayed during manual calculation.

```
ERR2
```
(sentence structure error)

```
ERR3
```
(mathematical error)

The following error messages will be displayed during program calculation.

```
ERR2 PØ-1Ø
```
program area line number

(A sentence structure error has occured at line 10 in program area PØ)

```
ERR3 P2-2Ø
```
(A mathematical error has occurred at line 20 in program area P2)

Furthermore, for the meaning of the error messages, see page 65 for the error message list.

* When the calculation range is exceeded (±9.999999999x10^{+99}), an overflow condition occurs and an error message is displayed. Also, below 1.0x10^{-99}, an underflow condition occurs and the calculation result will be "0".

2-4 Key Operation

First, turn the power ON.
"READY P0" is displayed and the calculator is ready for input.

1. Key Input

- **Alphabetical Input**

Example: Input ABC

Operation: [A][B][C]

ABC

Example: Input SIN

Operation: [S][I][N] (or [F][SIN])

SIN

* Either one key command or alphabetical command may be used.

- **Numerical Input**

Example: Input 123

Operation: [1][2][3]

123

Example: Input 96.3

Operation: [9][6][·][3]

96.3

- **Symbol Input**

Example: Input $#?

Operation: [S][$ R][S][# E][S][? U]

$ # ?

Example: Input ¥ΣΩ

Operation: [MODE][·][S][¥ R][S][Σ B][S][Ω N][MODE][·]

¥ΣΩ

- **Input of numbers with exponents**

Example: Input 7.896 x 10^{15}

Operation: [7][·][8][9][6][π E][1][5]

↑ Exponential symbol

7.896E15

Example: Input –2.369 x 10^{-45}

Operation: [–][2][·][3][6][9][π E][–][4][5]

-2.369E⁻45

2. Changing Input Contents (correction, deletion and insertion)

- **Correction**

For correction, move the cursor to the location to be corrected (using [⇐] and [⇒]) and at that position, press the correct character, number or symbol.

Example: Correct "A$" to "B$".

A$_

Operation: Move the cursor 2 character positions to the left.

[⇐][⇐]

A$

Press the [B] Key.

B$

Example: Correct "LIST" to "RUN".

```
LIST_
```

Operation: Move the cursor 4 character positions to the left.

← ← ← ←

```
LIST
```

Press R U N SPC or S RUN/B.

```
RUN _
```

- **Deletion**

For deletion, move the cursor to the position to be deleted and press the DEL Key. Each time the key is pressed, one character is deleted and the characters to the right of that will move one position to the left.

Example: Delete one of the "I" characters from "SIIN".

```
SIIN_
```

Operation: Move the cursor 2 character positions to the left.

← ←

```
SIIN
```

Press DEL.

```
SIN
```

Example: Delete "X," from "INPUT X, Y".

```
INPUT X,Y_
```

Operation: Move the cursor 3 positions to the left.

← ← ←

```
INPUT X,Y
```

Press DEL DEL.

```
INPUT Y
```

- **Insertion**

For insertion, move the cursor to a position located just to the right of the character after which you want to make an insertion. At that position, press S INS/DEL and one character space will be opened up. Then press the desired character, number or symbol key.

Example: Change "T=A$" to "T$=A$".

```
T=A$_
```

Operation: Move the cursor 3 character positions to the left.

← ← ←

```
T=A$
```

Press S INS/DEL and open up one character space.

```
T_=A$
```

Press S $/R.

```
T$=A$
```

Example: Change "PRINT X" to "PRINT SIN X".

```
PRINT X_
```

Operation: Move the cursor 1 character position to the left.

←

```
PRINT X
```

Press S INS/DEL S INS/DEL S INS/DEL.

```
PRINT    X
```

Press S I N.

```
PRINT SINX
```

The above are methods for changing input contents.

Chapter 3

Manual Calculation

3-1 Explanation of Manual Calculation

For manual calculation, calculations are not automatically performed using previously stored formulas as in program calculation. Operations for moving numbers from right to left, calculating by hand and calling out contents of variables are called manual calculations.

3-2 Manual Calculation Operation

- Addition, subtraction, multiplication and division are by true algebraic logic. [+], [−], [*] (×), [/] (÷) and [EXE] (=) are used. The [EXE] Key is used to obtain calculation results and functions as "=".

Example: 12+36−9×5÷4=36.75

Operation: [1][2][+][3][6][−][9][*][5][/][4] → `12+36-9*5/4`

[EXE] → `36.75`

- Function calculation includes addition, subtraction, multiplication and division and is the same as true algebraic logic. Data is written after function commands.

Example: log 1.23=0.0899051114

Operation: LOG 1[.]23 → `LOG1.23`

[EXE] → `0.0899051114`

* Letters and numbers will be written in this manual without frames.

Example: [S][I][N][S][(][1][5][+][8][S][)][EXE] → SIN[S][(]15[+]8[S][)][EXE]

* Function and program commands can be written using either one key commands or alphabetical commands but will be written in this manual using alphabetical commands.

- Calculations such as memory calculations used to store number values or calculation results and to obtain totals will use variables. These variables are alphabetical letters (A–Z), or letters and numbers (0–9) in combination (when the memories are used as the array variables). To put a number value or calculation result into a variable, use substitution.

Example: Store 1234 in variable A

Operation: A[=]1234 → `A=1234`

[EXE] → `_`

Example: Add the result of 23 x 56 to variable K

Operation: K[=]K[+]23[*]56 | `K=K+23*56`

[EXE] | `—`

This method is the manual way to perform the same operation as sentence substitution in programs.

- To make corrections prior to pressing the [EXE] Key, move the cursor to the position to be corrected and press the correction key. (See Chapter 2)
- Press the [AC] Key to cancel the entire display.

3-3 Manual Calculation Examples

3-3-1 Fundamental Calculations

■ Addition, subtraction, multiplication and division

Example: $23+4.5-53=-25.5$

Operation: 23[+]4[.]5[-]53[EXE] | `-25.5`

Example: $56\times(-12)\div(-2.5)=268.8$

Operation:
56[*][S][(][-]12[S][)][/][S][(][-]2[.]5[S][)][EXE] | `268.8`

(may be omitted)

Example: $12369\times7532\times74103=6.9036806\times10^{12}$ (=6903680600000)

Operation: 12369[*]7532[*]74103[EXE] | `6.9036806E12`

Example: $1.23\div90\div45.6=2.9970760\times10^{-4}$ (=0.00029970760)

Operation: 1[.]23[/]90[/]45[.]6[EXE] | `2.9970760E-04`

* When the result is greater than 10^9 or less than 10^{-3}, it will be displayed exponentially.

Example: $\underline{7\times8}+\underline{4\times5}=76$

Operation: 7[*]8[+]4[*]5[EXE] | `76`

Example: $12+\underline{(2.4\times10^5)\div42.6}-\underline{78\times36.9}=2767.602817$

Operation:

12[+]2[.]4[EXP]5[/]42[.]6[-]78[*]36[.]9[EXE] | `2767.602817`

- **Memory Calculation**

Example: $\underline{12}\times45=540$

$\underline{12}\times31=372$

$75\div\underline{12}=6.25$

Operation:

A ⊟ 12 EXE	_
A ⊠ 45 EXE	54Ø
A ⊠ 31 EXE	372
75 ◪ A EXE	6.25

Example: 23+9=32

53−6=47

−) 45×2=90

99÷3=33

Total 22

Operation:

M ⊟ 23 ⊞ 9 EXE	
M ⊟ M ⊞ 53 ⊟ 6 EXE	
M ⊟ M ⊟ 45 ⊠ 2 EXE	
M ⊟ M ⊞ 99 ◪ 3 EXE	
M EXE	22

* Using this operation method, individual calculation results cannot be identified. To see the individual results, perform in the following manner.

23 ⊞ 9 EXE	32
M ⊟ ANS EXE	
53 ⊟ 6 EXE	47
M ⊟ M ⊞ ANS EXE	
45 ⊠ 2 EXE	9Ø
M ⊟ M ⊟ ANS EXE	
99 ◪ 3 EXE	33
M ⊟ M ⊞ ANS EXE	
M EXE	22

3-3-2 Function Calculations

■ Trigonometric Functions (sin, cos, tan) and Inverse Trigonometric Functions ($\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$)

- When using trigonometric and inverse trigonometric functions, be sure that the angular units are designated.
  (If angular units are not changed, this does not have to be accomplished again.)

Example: sin 12.3456° = 0.2138079201

Operation: [MODE] [4] → "DEG"

SIN 12 [·] 3456 [EXE] | DEG 0.2138079201

(or [F] [SIN] 12 [·] 3456, same for following)

Example: 2·sin 45° × cos 65.1° = 0.5954345575

Operation:

2 [✱] SIN 45 [✱] COS 65.1 [EXE] | DEG 0.5954345575

Example: $\sin^{-1}0.5 = 30°$ (To obtain x when $\sin x° = 0.5$)

Operation: ASN 0 [·] 5 [EXE] | DEG 30

(or [F] [ASN] 0 [·] 5, same for following)

Example: $2.5 \times (\sin^{-1}0.8 - \cos^{-1}0.9) = 68.22042398$

Operation:

2 [·] 5 [✱] [S] [(] ASN 0 [·] 8 [−] ACS 0 [·] 9 [S] [)] [EXE] | DEG 68.22042398

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

Operation: [MODE] [5] → "RAD"

COS [S] [(] [S] [π] [/] 3 [S] [)] [EXE] | RAD 0.5

Example: $\cos^{-1}\frac{\sqrt{2}}{2} = 0.7853981634$

Operation: ACS [S] [(] SQR 2 [/] 2 [S] [)] [EXE] | RAD 0.7853981634

Example: tan(−35 gra) = −0.612800788

Operation: [MODE] [6] → "GRA"

TAN [−] 35 [EXE] | GRA −0.612800788

■ Logarithmic Functions (log, ln) and Exponential Functions (e, x^y)

> EXP (e^x) cannot be used in a continuous calculation. Also, it cannot be used in a multistatement.

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: LN 90[EXE] | 4.49980967

Example: $\log 456 \div \ln 456 = 0.4342944819$

Operation: LOG 456[÷]LN 456[EXE] | 0.4342944819

Example: $e=2.718281828$

(This function is a command to call out the numerical value of the exponential table.)

Operation: EXP 1[EXE] | 2.718281828

Example: $10^{1.23}=16.98243652$

(To get the antilogarithm of common logarithm 1.23)

Operation: 10[S][↑]1[·]23[EXE] | 16.98243652

Example: $5.6^{2.3}=52.58143837$

Operation: 5[·]6[S][↑]2[·]3[EXE] | 52.58143837

Example: $123^{\frac{1}{7}}\ (=\sqrt[7]{123})=1.988647795$

Operation: 123[S][↑][S][(]1[÷]7[S][)][EXE] | 1.988647795

Example: $(78-23)^{-12}=1.3051118\times10^{-21}$

Operation: [S][(]78[−]23[S][)][S][↑][−]12[EXE] | 1.3051118E-21

Example: $2^2+3^3+4^4=287$

Operation: 2[S][↑]2[+]3[S][↑]3[+]4[S][↑]4[EXE] | 287

Example: $\log \sin 40°+\log \cos 35°=-0.278567983$

The antilogarithm is 0.5265407845 (logarithmic calculation of sin 40° x cos 35°)

Operation: [MODE] [4] (DEG designation)

LOG SIN 40[+]LOG COS 35[EXE] | −0.278567983

10[S][↑][ANS][EXE] | 0.5265407845

* The input range of power ($x \uparrow y$) is $x>0$.

■ Other Functions (√‾ , SGN, RAN#, RND, ABS, INT, FRAC)

Example: $\sqrt{2}+\sqrt{5}$=3.65028154

Operation: SQR 2 [+] SQR 5 [EXE] | 3.65028154

Example: Give "1" to a positive number, "−1" to a negative number, and "0" to a zero.

Operation: SGN6 [EXE] | 1

SGN0 [EXE] | 0

SGN [−] 2 [EXE] | −1

Example: Random number generation (pseudo random number of 0 < RAN# < 1)

Operation: RAN [S] [#] [EXE] | 0.904186914

(or [F] [RAN#] [EXE])

Example: The result of 12.3 x 4.56 is rounded to 10^{-2} 12.3×4.56=56.088

Operation: RND [S] [(] 12 [.] 3 [×] 4 [.] 56 [S] [,] [−] 2 [S] [)] [EXE] | 56. 1

Example: |−78.9÷5.6|=14.08928571

Operation: ABS [S] [(] [−] 78 [.] 9 [/] 5 [.] 6 [S] [)] [EXE] | 14.08928571

Example: The integer portion of $\frac{7800}{96}$ is 81

Operation: INT [S] [(] 7800 [/] 96 [S] [)] [EXE] | 81

* This command will not exceed the original number value.

Example: The decimal portion of $\frac{7800}{96}$ is 0.25

Operation: FRAC [S] [(] 7800 [/] 96 [S] [)] [EXE] | 0.25

■ **Effective Number of Positions Designation and Decimal Designation**

Effective number of positions designation and decimal designation are performed using "SET" command.

Effective number of positions designation . . . . SET E n (n=0 to 9)
Decimal designation . . . . SET F n
Designation cancellation . . . . SET N

* For effective number of positions designation, "SET E 0" is a 8 position designation.

* If designation is performed, the last designated position will be displayed rounded off. The original number value will remain in the internal calculation section or in the memory.

Example: 100÷6=16.66666666........

Operation: SET [E] 4 [EXE] (designates 4 effective positions)

100 [/] 6 [EXE] | 1.667E01

Example: 123÷7=17.57142857........

Operation: SET [F] 2 [EXE] (designates 2 decimal positions)

123 [/] 7 [EXE] | 17.57

Example: 1÷3=0.3333333333. ....

Operation: SET [N] [EXE] (designation cancellation)

1 [/] 3 [EXE] | 0.3333333333

Chapter 4

Program Calculation

This instrument uses BASIC as its program language. BASIC is an abbreviation for Beginner's All-purpose Symbolic Instruction Code. Also, it is said that it is a fundamental language system which is easy for beginners to use.

BASIC Language Features

1. It is a problem solving language and programming efficiency is improved.
2. Not only is it a program language which is not limited to a particular field, but is a general use language which is applicable to many fields, such as natural science, social science and business fields.
3. It is a conversation type language and the computer user can compose programs while conversing with the computer and use it for making entries.
4. It has many of the capabilities and features of FORTRAN but is free from the rules encountered when using FORTRAN.
5. Many fundamental built-in functions have been prepared and it has an expanded calculation range.

4-1 Program Outline

Program calculation **1)** programs calculation contents to be executed, **2)** calculator stores programs, and **3)** using these programs. Simply input the data and the results can be obtained automatically.

■ Programming Fundamentals

Let's take a look at the programming necessary to use the computer for a given problem, and at the concept and programming procedures.

● Programs and Programming

When the operator uses the computer to manage a problem, instructions must be used which are in words that the computer can understand. These instructions are called programs and the composing of these instructions is called programming.

● What is a program?

In order to make a program, there are many grammatical rules, but these details will be explained later. First, to discuss what a program is and its form, let's take a look at an example of a fundamental program.

```
    ┌────Command
    │     ┌────Operand
10 INPUT A, B  ──────────Input statement

20 C = A + B   ──────────Operation statement

30 PRINT C     ──────────Output statement
 └────Line number
```

The above program is a fundamental program and consists of an input statement, an operation statement, an output statement and line numbers. That is to say, the input statement inputs data. According to that data, the operation statement performs various operations, and the output statement outputs the execution results. Also, each line has a line number preceding it. These operation statements are not limited to one but are performed several times and, by adding decision statements, the program becomes long and complicated. Nevertheless, they are fundamentally the same.

Also, on one line, following the line number, a COMMAND is written which tells the calculator what to perform next. It is composed of alphabetical characters. It is followed by OPERANDS which indicate the necessary information for the command.

The above is called a PROGRAM and is in the fundamental form.

- **Number of program steps**

Program steps are counted as follows.

1) Program command . . . . 1 step/1 command
2) Functional command . . . . 1 step/1 command
3) Line number . . . . 2 steps/1 line number
4) Character . . . . 1 step/1 character
5) Pressing the EXE Key after each line number's key input to store in the calculator. . 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    1  1  1  1  1  1   1
                                                  Total 22 steps
```

■ Programming Sequence

Programming progresses in the following sequence.

1) Problem analysis
2) Flow chart preparation
3) Writing of the program on a coding sheet
4) Debugging execution

Explanation

Step 1: Analyze the given problem to determine the required steps for solving.

Step 2: Write a flow chart to represent the flow of logic for the solution of the problem. The flow chart is composed of symbols which show processing and decision making elements.

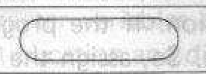
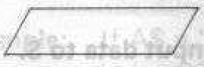
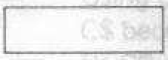
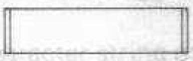
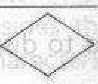
Step 3: Based on the flow chart, write a program on coding sheets or similar forms using BASIC language.

Step 4: Check the program for mistakes.

The above is the procedure for composing a program.

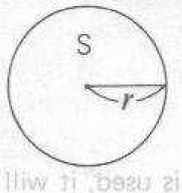
■ Flow Chart

The most widely used flow chart symbols are written below.

Symbol	Nomenclature	Meaning
	Terminal	Initiate, terminate, etc.
	Input/Output	Input/output functions.
	Process	Numerous processing functions.
	Predefined process	A group of commands defined elsewhere, such as a subroutine.
	Decision	Decision on the choice of a specific route from among several possible routes.

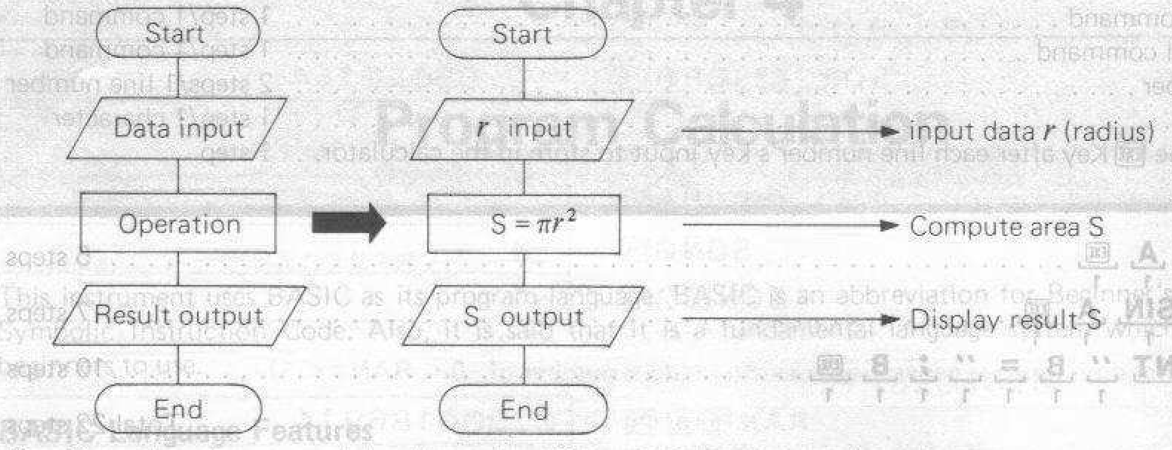
- **Flow Chart Example**

Here, let's take a look at a simple example of a program for determining the area of a circle.



$S = \pi r^2$

First, following programming fundamentals, consider the data input, operation and result output separately as shown below.



The flow chart is prepared in this manner following the program flow and is composed viewing the program in its entirety.

When programming, make it a habit to compose a flow chart. If more complex programs are created, confusion can be avoided.

■ Coding

Coding is the process of writing programs on coding sheets or similar forms after program assembly.

When assembling programs, certain operating symbols are required. The simplest fundamental operating symbols are written below.

The four fundamental arithmetic operation symbols

+ and − are expressed as " + " and " − "

x and ÷ are expressed as " * " and " / "

Powers, such as x^2 and x^3, are expressed as "x↑2" and "x↑3".

The "=" used in assignment statements will be explained in greater detail later but is explained briefly here. For example, the "=" in S = πr^2 means to assign the calculation result of πr^2 to S as opposed to having the meaning of equal as in mathematics.

Let's write a program to determine the area of the previously specified circle.

1) Write an input statement to input data r.

There are many input commands but usually the "INPUT" command is used to input data from the keyboard during program execution. The input statement used to input data r becomes INPUT R. If a line number is added to the input statement to make it complete, it becomes 1Ø INPUT R.

These line numbers can be in the range 1 to 9999 but identical line numbers cannot be used in the same program. Otherwise, the later line number will take priority. Usually, it is more convenient to assign line numbers in increments of ten for easy addition, correction and deletion. If the program is started, the statements are executed in smaller to larger line number sequence. So, assign the line numbers in program execution sequence.

2) Next, make an assignment statement to assign the result of the calculation from the input data to S.

Since πr^2 means $\pi \times r^2$

then **S = π*R↑2** (X is represented by * and R^2 is represented by R↑2)

Therefore, with the line number attached as usual, it becomes

2Ø S = π*R↑2

3) Write an output statement to output (display) the result of the operation.

Since this is just for calculation and does not cause a display, use "PRINT" command to display the result.

The output statement to output (display) result S becomes

PRINT S

And, adding a line number to this, it becomes

3Ø PRINT S

So, the complete program for determining the area of a circle is:

```
1Ø INPUT R
2Ø S=π*R↑2
3Ø PRINT S
```

For program coding, it is not necessary to use the exclusive coding paper but, if it is used, it will be more convenient for problem analysis or writing flow charts in a standard format after composing.

4-2 Program Fundamentals

■ Constants and Variables

Characters used for BASIC are upper case letters of the alphabet (A to Z) and numbers (0 to 9) as well as some special characters (such as symbols).

- **Constants**

A constant is a fixed value which can be written directly into a program.

Example: In the expression $S=\pi r^2$, it becomes S=π*R↑2 and 2 is the constant.

- **Variables**

A variable is a value that is used in a program but is input from the keyboard during execution and is used when the result of a calculation is unknown initially because the result is assigned during execution.
A variable is a single capital letter (A through Z) or a single capital letter with "$" attached (character variable). They may be freely selected from within this range.

Example: In the expression $S=\pi r^2$, which becomes S=π*R↑2, R is the variable.

```
Example:  Y = 2 * X ↑ 2 + 3 * X + 4              V: Variable
          |   |   |   |   |   |   |
          V   C   V   C   C   V   C              C: Constant
```

That is, algebraics and constants used in mathematics correspond to variables and constants, respectively.
Also, in addition to the above, there are character constants and character variables. Character constants are strings of characters which are written in directly, such as "ABC" and "END", and enclosed in quotation marks and spelled out. Character variables are not a numeric value but variables which accept a character string. Each time a character string is given the contents will change.
A numerical variable and a character variable which use the same letter can not be used at the same time.
A character string is composed of characters enclosed in quotation marks, such as "123" and is not a numerical value. Thus, "123" is just the numbers 1 and 2 and 3 written in sequence and is considered the same as "ABC" in quotation marks.
Character variables are general variables (such as A, B, X and Y) which have a $ sign attached. Selections can be made from within this range.

Example: A$, B$, C$, X$, Y$

Character variables can be compared or added to one another but other operations (such as subtraction, multiplication and division) cannot be performed.

Example: If A$ = "123" and B$ = "456"
Using C$ = A$ + B$
C$ becomes "123456"
(If C$ = B$ + A$, C$ becomes "456123")

A character string of up to 7 characters can be inserted into this character variable.
Also, besides these character variables, there is an exclusive character variable ($) which can contain up to 30 characters.

Example: $ = "1234567890ABCDEFG"

This exclusive character variable can use character functions (MID functions) which will be explained later. This can be used more conveniently than other character variables.

■ Assignment Statements

BASIC assignment statements are in the form shown below.

Variable = Numeric Expression

In BASIC assignment statements, an expression having arithmetic operations (+, −, ✱, /) on the right side is called a numeric expression.

Example: For Y = 2✱X+3, the 2✱X+3 on the right side is a numeric expression.
This " = " does not mean "equals", it means "assign".

Example: For Y = 2✱X+3, the left side is the variable and the right side is the numeric expression. Thus, it does not mean, as in usual mathematics, that the Y on the left side and the 2✱X+3 on the right side are equal. It means to assign the result of 2✱X+3 to Y. (Y =2✱X+3 can be better understood if it is thought of as Y ← 2✱X+3)

Example Assignment Statements

A = B . . . . Assign the value of B to A (the former value of A is deleted).
N = 2✱M . . . . . . Assign double the value of M to N.
X = Y + Z . . . . . Assign the result of the sum of Y and Z to X.
I = I + 1. . . . . . . Assign the value of I + 1 to I (the number value in I will be increased by 1).

4-3 Program Writing and Execution

■ Program Writing

Storing a program in the calculator's memory system is called program writing.
This operation is performed by keying in inputs from the keyboard.

1. WRT mode designation
2. Program area designation
3. Program input by line units (writing)
4. The program area can be divided into 10 parts from P0 to P9. Programs are written somewhere in this program area.

(1) WRT Mode Designation

Program writing is performed in the WRT mode.
Press MODE 1. "WRT" will be displayed.

(2) Program Area Designation

To designate a program area, press the S Key then a 0 to 9 numerical key.

S $\overset{P0}{0}$ →P0
S $\overset{P1}{1}$ →P1
S $\overset{P2}{2}$ →P2
S $\overset{P3}{3}$ →P3
S $\overset{P4}{4}$ →P4
S $\overset{P5}{5}$ →P5
S $\overset{P6}{6}$ →P6
S $\overset{P7}{7}$ →P7
S $\overset{P8}{8}$ →P8
S $\overset{P9}{9}$ →P9

(3) Program Input (Writing)

Program writing is performed in line units. Up to 62 characters, including the line number, can be written. Finally, press the EXE Key.

*** The Role of the EXE Key.**

The EXE Key is pressed to write programs, input data and to obtain the result of manual calculations. For program writing, press the EXE Key after each line number's key input to store in the calculator. Program writing or written program content change, addition and deletion are all followed by pressing the EXE Key as the final step. If the contents of the display are changed and the EXE Key is not pressed, the written in contents will not be changed.

Example: Write the following program to P0

```
10 INPUT A,B
20 V=A+B
30 W=A-B
40 PRINT V,W
50 END
```

Operation:

1. **Designate WRT mode.**

[MODE] [1]

Number of remaining steps.

```
WRT  1568
P 0123456789
```

Unwritten program areas.

Currently designated program area will blink.

* Number of steps will be changed by the number of memories or amount of programs written.

2. **Designate program area P0**

[S] [P0]

Number of remaining steps

```
WRT  1568
P 0123456789
```

Currently designated program area will blink.

Unwritten program areas

Omitted hereafter

3. **If a previously written program remains, clear it.**

CLEAR [EXE]

```
WRT  1568
P 0123456789
```

(When nothing is written, omit this step.)

4. **Write line number 10.**

10␣INPUT␣A [S] [,] B [EXE]

```
     1561
10 INPUT A,B
```

Means 1 character space (May be omitted)

* Be sure to press when changing lines.

5. **Write line number 20.**

20␣V [=] A [+] B [EXE]

```
     1553
20 V=A+B
```

6. **Write line number 30.**

30␣W [=] A [-] B [EXE]

```
     1545
30 W=A-B
```

7. **Write line number 40**

40␣PRINT␣V [S] [,] W [EXE]

```
     1538
40 PRINT V,W
```

8. **Write line number 50**

50␣END [EXE]

```
     1534
50 END
```

- Use the "END" command to terminate a program. The "END" command may be omitted in case of a program as above, but it should be written to clarify the termination of a program when using GOTO statements or GOSUB statements.
- A space is written between line numbers and commands and operands to make it easier to read the display. In BASIC language, except messages such as PRINT statement, it has no special meaning.
- In this case, line numbers are input in units of 10. Line numbers can be freely selected from within a range of 1 to 9999. Selecting in units of 10 makes it easy to make additions and insertions later. Program execution is performed in line number sequence from small numerical value, so attach line numbers in program run sequence.
- Use the "CLEAR" command as above to clear a previously written program, while the "CLEAR A" command to clear all of previously written programs (P0 to P9).

■ Program Execution

Program execution is performed in the RUN mode (press [MODE] [0]. . . "RUN" will be displayed). There are two methods for executing written programs.

(1) Program Execution Methods

1. Execution by program area designation

For this method, execution is begun at the same time as program area designation.

[S] { [P0 0] ∫ [P9 9] } (Press one of the [P0 0] to [P9 9] keys after pressing [S])

Example: Start the program specified in the previous example.

RUN mode (omitted hereafter)

Operation: [S] [P0 0]

```
                RUN
 ?
```

* This "?" is because the INPUT statement is written at the beginning.

2. Execution by RUN command

RUN[EXE] (RUN can be input by pressing [R] [U] [N] or [S] [RUN .])

```
 ?
```

* Continuing from the previous example, a "?" is displayed. In an input waiting situation, cancellation will not be effected by [AC]. After pressing [MODE] [0], operation 2 is performed.
Also, to execute along the way, after the RUN command, input the line number and press the [EXE] Key.

Example: Start from line 20.

Operation: RUN20[EXE]

* For method 1, it is not necessary to designate the program area to be executed. However, for method 2, it is necessary to designate the program area to be executed. (If the program area is different, the program written in that program area will be executed.)

(2) Key Input during Program Execution

Key input during program execution uses the INPUT statement and KEY function. Key input by the KEY function is 1 key input only but even when no key is input, execution continues.
For key input using the INPUT statement, a "?" is displayed and there is a pause awaiting input. After data input, execution is started by pressing the [EXE] Key.

Example: Execute the program written in P0 in the previous example.

Operation: Execute program

[S] [P0 0]

```
 ?
```

● For this program, 2 variables are input. First, the value of variable A is input.

A VALUE 47[EXE]

```
 ?
```

● Next, the value of variable B is input.

B VALUE 69[EXE]

```
 116                STOP
```

[EXE]

```
 -22                STOP
```

In this manner, for key input during execution using the INPUT statement, input data using "data [EXE]."
In addition, in a waiting input situation, if [MODE] [0] are pressed, program execution will stop.

4-4 Program Edit

- Program edit is used to make a program logically correct and makes it possible to make changes, additions and deletions or to accomplish line number rewrite.
- Program edit is performed using the LIST command to call out each line.
- LIST command is usable in both the RUN mode and the WRT mode but if the RUN mode is used, program contents will be displayed. If the WRT mode is used, program edit can be performed.

(1) Display of the Program List Using the RUN Mode. (Display lasts about 2 seconds)

Operation: LIST EXE

(LIST can be L I S T or S LIST/N)

```
   10 INPUT A,B
   20 V=A+B
   30 W=A-B
   40 PRINT V,W
   50 END
 READY P0
```

Furthermore, if the list is not needed from the beginning, designate the line number.

To list from line number 30:

Operation: LIST 30 EXE

```
   30 W=A-B
   40 PRINT V,W
   50 END
 READY P0
```

* During the LIST command execution, the display will be made in sequence until the end. Press the STOP Key to stop.
Also, to continue the stopped LIST command, press the EXE Key.

(2) Program Change, Addition and Deletion in the WRT Mode.

Press MODE 1 and designate the WRT mode.

1. Changing

Using the LIST command, each line will be displayed in sequence from the designated line number each time the EXE Key is pressed. Also, if the designated line number is omitted, the display will start from the first line.

a. Partial change

Example: Change the "+" to "*" on line 20 in the previous example.

Operation:

- In case the program area is not designated at P0, designate P0.

S P0

```
      1534
 P ⩶123456789
```

- Call out line number 20 using the LIST command.

LIST 20 EXE

```
     1534
  20 V=A+B_
```

- Move the cursor and set it at the location of the desired change (i.e. "+").

⇦ ⇦

```
     1534
  20 V=A+B
```

* If a cursor movement key (⇦ ⇨) remains pressed for more than 1 second, fast movement can be achieved.

- Make the correction.

* EXE

```
     1534
  30 W=A-B_
```

* Be sure to press the EXE Key. If it is not pressed, only the display will be changed and the written program contents will not be changed.

● At this time, line 30 will be changed. Press the AC Key to clear the display and the change is complete.

```
AC          1534
            _
```

* Operation of other keys at a line where no change is required causes them to be written in on that line. Therefore, do not press any keys other than EXE and AC.

● Use LIST to make sure of the change.

```
MODE 0      READY P0
LIST EXE     10 INPUT A,B
             20 V=A*B
             30 W=A-B
             40 PRINT V,W
             50 END
            READY P0
```

b. Changing an entire line

Input the line number to be changed. (In this manner, the previously input line number is cleared.)

Example: Change "W=A−B" to "W=V/2" on line 30.

Operation:

```
MODE 1          1534
            P ⁚123456789
```

● Write the new line 30.

```
30␣W=V/2 EXE    1534
             30 W=V/2
```

● Check the program list.

```
MODE 0      READY P0
LIST EXE     10 INPUT A,B
             20 V=A*B
             30 W=V/2
             40 PRINT V,W
             50 END
            READY P0
```

2. Addition

To add line units, use a line number that falls between the line numbers where the addition is desired.

Example: Add "U=V×2" between line 30 and line 40 in the previous example and change line 40 to "PRINT V, W, U".

Operation:

```
MODE 1          1534
            P ⁚123456789
```

● To input between line 30 and line 40, input line number 35.

```
35␣U=V*2 EXE    1526
             35 U=V*2
```

* Select a line number from 31 to 39 for input between line 30 and line 40.

● To change line 40, call it out using the LIST statement and add ", U".

```
LIST 40 EXE     1526
             40 PRINT V,W
→ S , U EXE     1524
             50 END _
AC              1524
            _
```

- Check the list to make sure of the program additions.

```
MODE 0        READY P0
LIST EXE        10 INPUT A,B
                20 V=A*B
                30 W=V/2
                35 U=V*2
                40 PRINT V,W,U
                50 END
              READY P0
```

3. Deletion

a. Partial deletion

Example: Delete "V , " from line 40 in the previous example.

Operation:

```
                    1524
MODE 1        P  123456789
```

- As in the partial change method, call out line 40 using the LIST statement.

```
                    1524
LIST 40 EXE     40 PRINT V,W
```

- Move the cursor and set it below the V (to the desired deletion position).

```
                    1524
← ←             40 PRINT V,W
```

- Use the DEL Key to delete "V,".

```
                    1524
DEL DEL         40 PRINT W,U
                    1526
EXE             50 END _
```

* If the EXE Key is not pressed, the program contents will not be changed.

```
                    1526
AC              _
```

* Be sure to press the AC Key to cancel the line number 50 change situation.

- Check the list to make sure the deletion was accomplished properly.

```
MODE 0        READY P0
LIST EXE        10 INPUT A,B
                20 V=A*B
                30 W=V/2
                35 U=V*2
                40 PRINT W,U
                50 END
              READY P0
```

b. Deletion of an entire line

Input the same line number as the one to be deleted and that entire line will be deleted.

Example: Delete line 30.

Operation:

```
                    1526
MODE 1        P  123456789
```

- Input the line number to be deleted, i.e. 30.

```
                    1534
30 EXE        _
```

- Make sure of the deleted position.

```
MODE 0        READY P0
LIST EXE        10 INPUT A,B
                20 V=A*B
                35 U=V*2
                40 PRINT W,U
                50 END
              READY P0
```

4. Line number correction

Example: Write the following program in P2.

```
10 INPUT N
20 M=N*N
30 L=SQR N
40 PRINT M,L
50 END
```

Move line number 20 between line 30 and line 40.

Operation: MODE 1 S P2

```
         1500
P _1※3456789
```

- Call out line number 20 using the LIST command.

LIST 20 EXE

```
     1500
20 M=N*N_
```

- Move the cursor beneath the 2 on line 20.

← ← ← ← ← ← ← ←

```
     1500
20 M=N*N
```

- Change the 20 to read 35 and input.

35 EXE

```
     1495
30 L=SQR N
```

- To complete the change, press AC and cancel the change command.

AC

```
     1495
_
```

- Use LIST to see how the program contents were changed.

MODE 0
LIST EXE

```
READY P2
 10 INPUT N
 20 M=N*N
 30 L=SQR N
 35 M=N*N
 40 PRINT M,L
 50 END
READY P2
```

- At this time, the contents of line 20 were moved between line 30 and line 40, but line 20 still remains. So, delete the unneeded line.

MODE 1
20 EXE

```
        1495
P _1※3456789
     1503
_
```

- This completes the line number move. Check the results using LIST.

MODE 0
LIST EXE

```
READY P2
 10 INPUT N
 30 L=SQR N
 35 M=N*N
 40 PRINT M,L
 50 END _
READY P2
```

■ Program Debug

(1) Program Debug System

This equipment's debug system is divided roughly into desk top debug and conversation type debug via display.

Debug Systems

- I. Desk Top Debug
  - a. **Complete Dubug**
    Program logic structure check.
  - b. **Partial Debug**
    Program line unit check.
- II. **Conversation type debug via display**
  Using the display, check program execution flow and errors in BASIC language composition using the automatic checking mechanism of the calculator.

The desk top debug is executed during programming.
Here the explanation of the conversation type debug via the display will be made.

(2) Conversation Type Debug

Any error made during program execution stage will be given on the display using an error message. These errors are displayed in line units and display the type of BASIC language error. Based upon the error message on the display, debug is accomplished by conversation using the manual.
Furthermore, for the meaning of the error messages, see page 65 for the error message list.

Example:

```
   10  INPUT X
   20  IF X≦0;PRINT"X≦0":GOTO 10
   30  Y=X↑2+3*X+15
   40  PRINT Y
   50  END
```

Line 20 of the above program is a judgement of the input range of power ($x\uparrow y$). When $x<0$, program execution will return to the INPUT statement on line 10.
Y = X↑2+3X+15 was entered on line 30 in this program by mistake.

Operation: If this program is executed, "?" is displayed using the INPUT statement on line 10.

MODE 0 RUN EXE

```
 ?
```

- At this time, input 45

45 EXE

```
 ERR2 P0-30
```

- This error message indicates that a statement structure error occurred at line 20 and the program contents must be confirmed.

AC MODE 1

```
    1540
 P_123456789
```

LIST 30 EXE

```
    1540
 30 Y=X↑2+3X+
```

- Since "*" is missing between "3" and "X" on line 30, correction is made using the program edit method.

← S INS DEL

```
    1540
 30 Y=X↑2+3_X
```

* EXE

```
    1539
 40 PRINT Y_
```

(3) Program Debug during Program Execution

Conversation type debug is debug that is performed using the information obtained from the calculator by an error message. However, if an error is not displayed but the calculation result is not as expected, repeat program execution and conduct debug by confirming the calculations along the way.

This method uses the STOP command to stop program progress and uses the TRACE mode to debug line by line.

● Debug using the STOP command

Example: Write the following program.

```
    10  Y=0
    20  INPUT N,X
    30  FOR I=1 TO N
    40  Y=Y+X*X
    50  NEXT I
    60  PRINT Y
    70  END
```

To see the value of Y in the FOR•NEXT loop, see the result of each loop using the STOP statement.

Operation: The best place to input the STOP statement is right after the calculation formula, so write the STOP statement between line 40 and line 50.

```
            MODE 1      P ⁂123456789     1522
      45 STOP EXE       45 STOP          1518
```

● By doing this, the program progress is stopped after the completion of the calculation on line 40 and debug can be performed.

```
            MODE 0      READY P0
           RUN EXE      ?
               4 EXE    ?
              87 EXE    _           STOP
                        ↑
                        Cursor blinks
```

● What is the value of Y at the time of this stop?

```
               Y EXE    7569        STOP
```

● When the program starts again, it will stop at the next STOP statement and request the value of Y again.

```
                 EXE    _           STOP
               Y EXE    15138
```

● By repeating this operation, the calculation process can be seen.
This example uses a simple program but when a complicated program is actually composed, it is very difficult to check the calculation process using desk top debug. So, if the check of variable is performed using this kind of STOP statement, program errors can be seen and corrected.

• Debug using TR (trace) mode

If program execution is performed using the TR mode (press MODE 2), it will execute each line and then stop and execution debug can be performed easily. Using the previous STOP command, let's perform a sample debug using the TR mode.

Operation:

Designate the RUN mode..........	MODE Ø	READY PØ
Designate the TR mode.........	MODE 2	TR READY PØ
	RUN EXE	TR STOP PØ-1Ø
	EXE	TR ?
Check the execution process..........	STOP	TR STOP ?
	STOP	TR STOP PØ-2Ø
Continue program execution..........	EXE	? "TR" and "STOP" will be omitted hereafter.
	4 EXE	?
	87 EXE	–
	STOP	PØ-2Ø
	EXE	PØ-3Ø
	EXE	PØ-4Ø
The value of Y	Y EXE	7569
	EXE	PØ-45

Repeated hereafter

Debug using the TR mode is ideal for checking the entire flow and is convenient for checking to see where mistakes have been made.

4-5 Program Commands

4-5-1 Jump and Loop

■ **Jump Commands**

Jump commands can be broadly separated into two commands. One is called a GOTO statement and is an unconditional jump. The other is a conditional jump combined with an IF statement.

● **GOTO Statement**

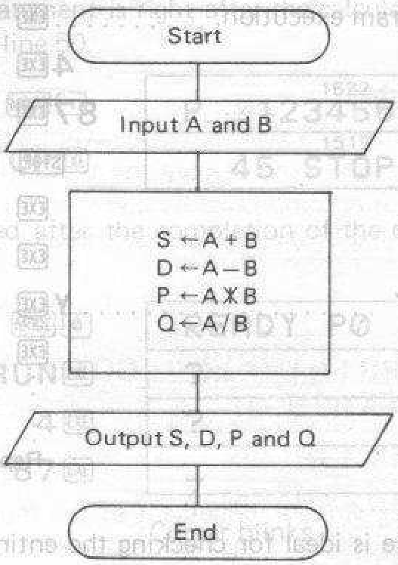
The GOTO statement is called an unconditional jump because it is a command that unconditionally advances the program to a designated location (line number).

Example 1: Add a GOTO statement in the program to determine the sum, difference, product and quotient of the data A and B.

First, the fundamental program is shown below.

```
10  INPUT  A,B
20  S=A+B
30  D=A-B
40  P=A*B
50  Q=A/B
60  PRINT  S,D,P,Q
70  END
```

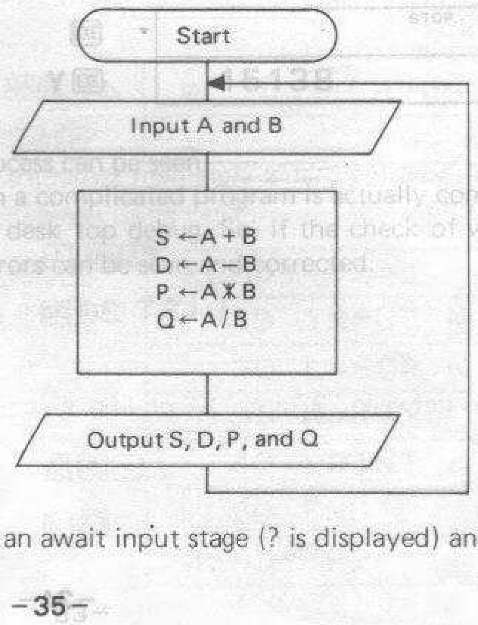
Flow Chart



Whenever this is executed, it must be allowed to "RUN" to insert data repeatedly. So a GOTO statement is input at line 70 instead of END so control will return to the proper location where the data is to be input. (Line 10 in this program)

This GOTO statement (GOTO 10) unconditionally jumps to line 10. The program is shown below.

```
10  INPUT  A,B
20  S=A+B
30  D=A-B
40  P=A*B
50  Q=A/B
60  PRINT  S,D,P,Q
70  GOTO  10
```



In this program, once the data is input, it goes to an await input stage (? is displayed) and the next data can be input right away.

Execution: Input data 15 and 3, and 903 and 43.

Operation:

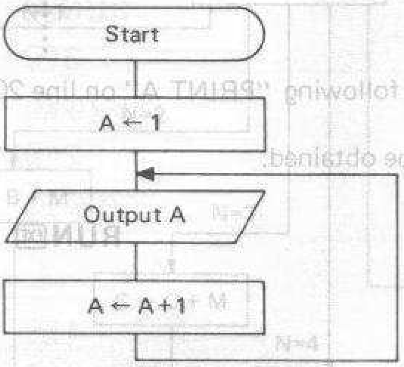
RUN EXE	?
15 EXE	?
3 EXE	18
EXE	12
EXE	45
EXE	5
EXE	?
903 EXE	?
43 EXE	946
EXE	860
EXE	38829
EXE	21

For this program, using the GOTO statement, which is input at the end, it returns to line number 10 "INPUT A, B". Also, as in this example, if a line number is written after GOTO, it will jump to the designated line. However, instead of a line number, if "#" and "0 to 9" are written, it will jump to a designated program area.

Example:
GOTO 10 (Jump to line 10 and execute from line 10)
GOTO #5 (Jump to program area P5 and execute P5 program)

Example 2: Make a program to increase the value of A by increments of 1.

```
10 A=1
20 PRINT A
30 A=A+1
40 GOTO 20
```



Explanation:
Since A is increased in value sequentially in increments of 1, it is necessary to assign an initial value of 1 to A. That is, "A=1" on line 10.
Next, PRINT A
At line 30, the result when 1 is added to A is assigned to A. That is, "A=A+1". Then since the program process causes a jump using the GOTO statement, it is necessary to jump to line 20 instead of returning to the beginning. So it becomes GOTO 20.
In this manner, the GOTO statement causes the program process to jump unconditionally to a designated location.

Note: When using the GOTO statement, it is necessary to be sure to designate the correct destination line number. If the line number is omitted from the GOTO statement, an error will occur.

PRINT Statement Application
The PRINT statement used in this program changes the manner of display in accordance with the delimiter designations following the operands.
First, for example 1, S, D, P and Q are separated by a "," (comma).
Display stops after S. To display the D, press the EXE Key. That is, if items are separated by a comma, the results will be displayed one at a time. If the "," is a ";", what will the display be like?
The result will be as follows.

Operation:

RUN EXE	?
15 EXE	?
3 EXE	18 12 45 5 STOP
EXE	?
9Ø3 EXE	?
43 EXE	946 86Ø 388
	86Ø 38829 21

The " ; " in this case (semicolon) operates when there are several results and causes everything to be displayed sequentially.

Furthermore, on the display, one space appears after each result. The "+" is not displayed at the sign column but is actually present.

␣18␣12␣45␣5

sign column

Also, if example 2 is executed as is, the result will be as shown below.

Operation:

RUN EXE	1
EXE	2
EXE	3
EXE	4
⋮	⋮

However, if " ; " is input following "PRINT A" on line 20,

2Ø PRINT A;

the following result will be obtained.

RUN EXE	1 2 3 4 5 6
	1 2 3 4 5 6
	2 3 4 5 6 7
	2 3 4 5 6 7
	3 4 5 6 7 8
	⋮

Therefore, continuous results are displayed by using " ; ".

- **GOTO Statement Indirect Designation**

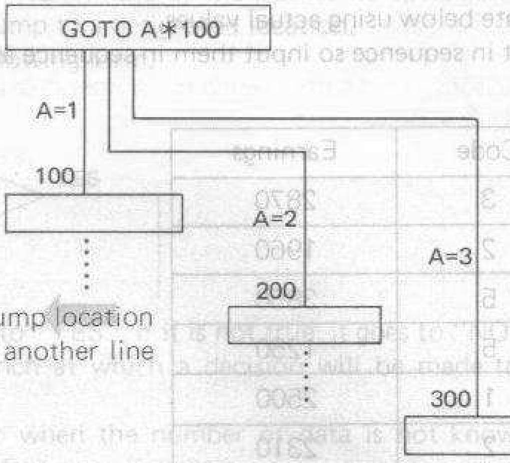
The previous GOTO statement designated the jump designation directly and caused jump. However, the indirect designation determines the jump destination according to the value of the variable. It is used when the jump destination cannot be written at the beginning of the program as when the operation method is understood from the data.

Indirect designation consists of;

GOTO variable or numeric expression
GOTO # variable or numeric expression

This uses the value of the variable (A,B,X,Y, etc.) or the numeric expression (A+B, X+10, etc.) to determine which line number to jump to or which program area to jump to.

Example 1: GOTO A✱100

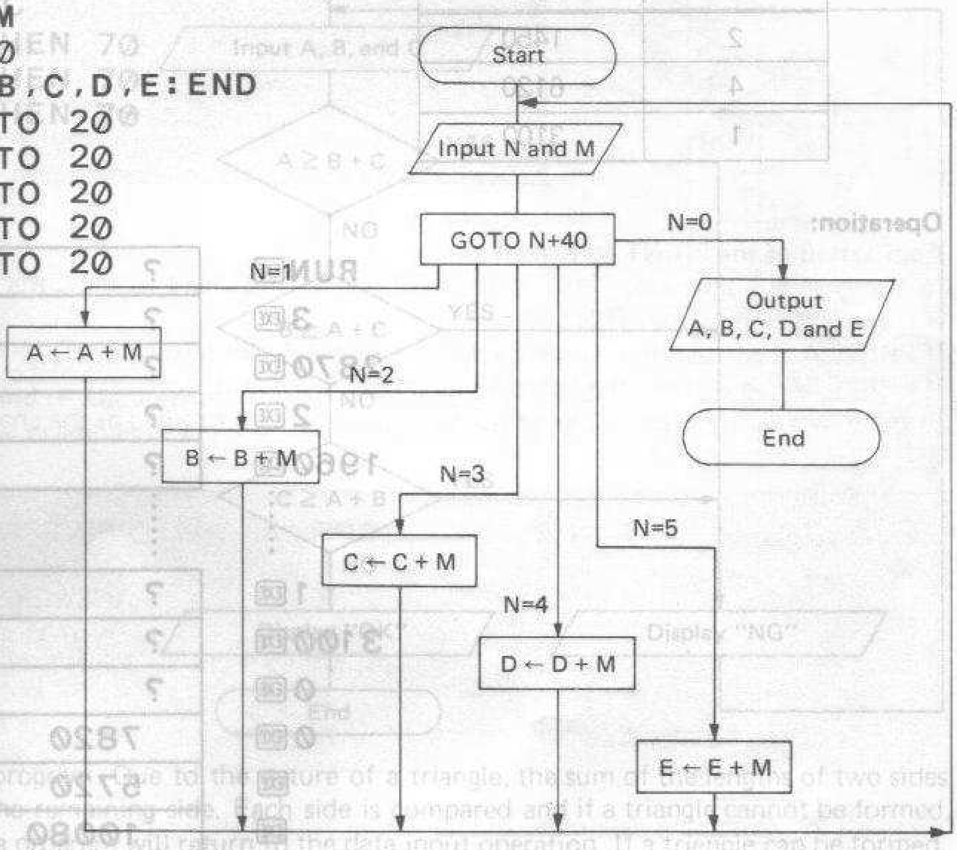


In this program, when A is 1, 2 or 3
For A=1, GOTO 100 means jump to line 100
For A=2, GOTO 200 means jump to line 200
For A=3, GOTO 300 means jump to line 300

If A is something other than 1, 2 or 3, then the jump location is unknown. This will cause an error or jump to another line number.

Example 2: Make a sorted totals program using indirect designation. (Classified here into 5 divisions)
The program will be as shown below.

```
10 VAC
20 INPUT N,M
30 GOTO N+40
40 PRINT A,B,C,D,E:END
41 A=A+M:GOTO 20
42 B=B+M:GOTO 20
43 C=C+M:GOTO 20
44 D=D+M:GOTO 20
45 E=E+M:GOTO 20
```



The "VAC" command on line 10 is a command for clearing the data use memory (makes it 0). In this program, clearance in advance is necessary in order to total the data input on lines 41 to 45. At the next INPUT statement, code number (N) and amount of earning are input.

Using line 30 GOTO statement, if the code number (N) is 1, jump to line 41, that is to say, the total earnings of code number 1.

In the case of code number 2, jump to line 42. In this manner, earnings (M) will be divided into code numbers 1 to 5.

```
N=1 : GOTO 41→41    A=A+M : GOTO 20 Execute
N=2 : GOTO 42→42    B=B+M : GOTO 20 Execute
N=3 : GOTO 43→43    C=C+M : GOTO 20 Execute
N=4 : GOTO 44→44    D=D+M : GOTO 20 Execute
N=5 : GOTO 45→45    E=E+M : GOTO 20 Execute
```

Therefore, when N is 0, it will jump to line 40. A, B, C, D and E sorted results are displayed and terminated. Furthermore, for this program, N must be input as 0 to 5. Any other number will cause an error. Let's demonstrate below using actual values.
Receipts are not in sequence so input them in sequence and sort the totals (within the 5 divisions).

Code	Earnings
3	2870
2	1960
5	3850
5	1250
1	2500
2	2310
3	1850
5	4370
3	5360
1	2220
2	1450
4	6120
1	3100

➡

Code	Earnings
1	7820
2	5720
3	10080
4	6120
5	9470

Operation:

```
     RUN EXE    ?
       3 EXE    ?
    287Ø EXE    ?
       2 EXE    ?
    196Ø EXE    ?
        :       :
       1 EXE    ?
    31ØØ EXE    ?
       Ø EXE    ?
       Ø EXE     782Ø
         EXE     572Ø
         EXE     1ØØ8Ø
         EXE     612Ø
         EXE     947Ø
```

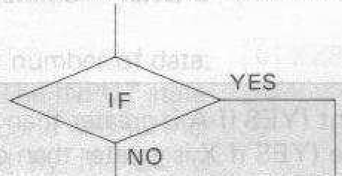
* The " : " used in lines 40 to 45 in this program is called "multistatement". It allows different commands to be written on one line. If commands are continuously performed, line numbers can be omitted and memory can be saved. Using this multistatement, many commands can be combined. However, the number of characters that can be written on one line, including the line number, is limited to 62 characters.

Note: A VAC (memory clear) command cannot be used in a multistatement.

- **IF Statement**

An IF statement is called a conditional jump due to its nature. It executes its operation only when certain conditions are satisfied and is a command to jump to a designated location.

If this is written in a flow chart, it will take the following form.



This means that if the IF statement is true, it goes to "YES", if it is not true, it goes to "NO".

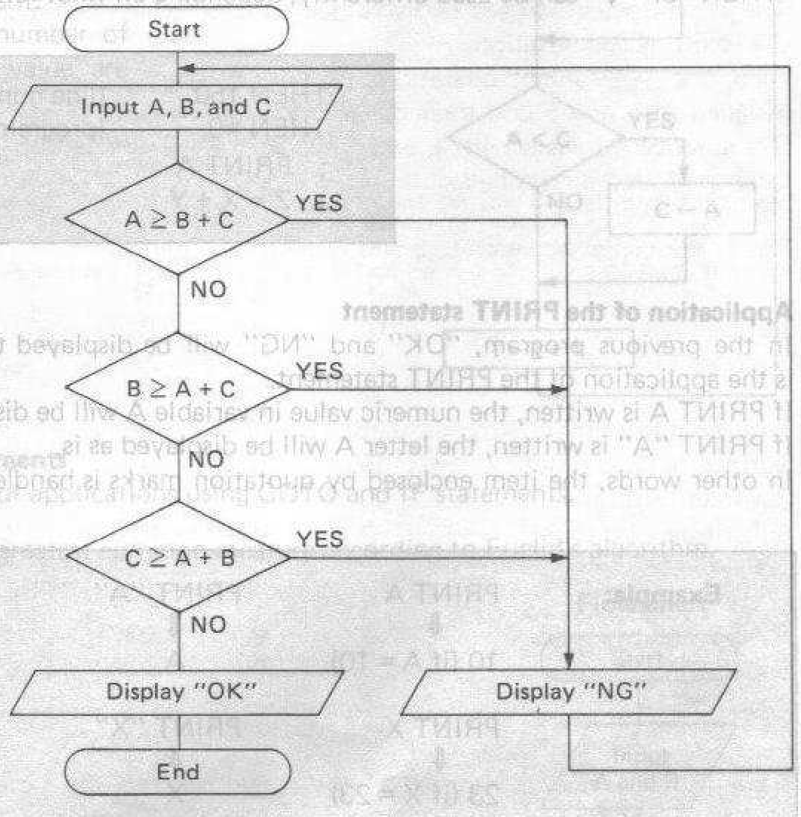
In other words, an IF statement indicates a branch at which a decision will be made to determine the next operation according to the result.

This IF statement is used for terminating a loop when the number of data is not known, or when the next operation is changed according to the result of an operation, etc.

Example 1: Input the lengths of three sides of a triangle and determine if a triangle can be formed using the three sides.

The program is shown below.

```
10 INPUT A,B,C
20 IF A≥B+C THEN 70
30 IF B≥A+C THEN 70
40 IF C≥A+B THEN 70
50 PRINT "OK"
60 END
70 PRINT "NG"
80 GOTO 10
```



Three data are input for this program. Due to the nature of a triangle, the sum of the lengths of two sides is greater than the length of the remaining side. Each side is compared and if a triangle cannot be formed, "NG" will be displayed and the program will return to the data input operation. If a triangle can be formed, "OK" will be displayed and program is terminated.

The IF statement is composed using expressions as shown below.

IF comparison expression THEN line number (numerical expression) or #n (n= 0 to 9)

or

IF comparison expression ; command or assignment statement

The comparison expression following "IF" compares the right side of an equality or inequality sign with the left side. If YES, it will proceed after "THEN" or " ; ". If NO, it proceeds to the next line. In other words, for line 20, if A is greater than or equal to the sum of B and C, a triangle cannot be formed. So, "THEN 70", in other words jump to line 70, command is performed.

This "THEN" includes the GOTO statement function.

If a line number is written following "THEN", it will jump to the designated line number. If "#" and 0 to 9 are written, it will jump to a program area (P0 to P9).
Also, when the comparison expression is YES, if a command or assignment statement is desired instead of a jump, " ; " is used instead of "THEN".
In these comparisons, constants, variables, numeric expressions, character constants and character variables can be used.

A > 10	variable and constant (YES if A is greater than 10)
X ≥ Y	variable and variable (YES if X is greater than or equal to Y)
N = L + 3	variable and numeric expression (YES if N is equal to the sum of L and 3)
A$ = "XYZ"	character variable and character constant (YES if the character string in A$ is equal to "XYZ")
P$ ≠ Q$	character variable and character variable (YES if the character string in P$ is unequal to the character string in Q$)

* Comparison of variables with character variables cannot be performed.
* Character string comparison applies to the ASCII codes.

"THEN" or " ; " can be used differently, depending on what follows them.

THEN 150	(line number)
THEN #9	(program area)
; PRINT A	
; Z = X + Y	

Application of the PRINT statement

In the previous program, "OK" and "NG" will be displayed to show the result of the comparison. This is the application of the PRINT statement.
If PRINT A is written, the numeric value in variable A will be displayed.
If PRINT "A" is written, the letter A will be displayed as is.
In other words, the item enclosed by quotation marks is handled as the character itself and displayed as is.

```
Example:   PRINT A          PRINT "A"     PRINT "ANSWER"
              ⇓                 ⇓               ⇓
           10 (if A = 10)       A         ANSWER

           PRINT X          PRINT "X"     PRINT "N=";N
              ⇓                 ⇓               ⇓
           23 (if X = 23)       X         N =␣15 (if N = 15)
```

Example 2: Obtaining maximum and minimum value is shown below.

```
1Ø  INPUT A
2Ø  B=A
3Ø  C=A
4Ø  I=1
5Ø  INPUT A
6Ø  IF A=Ø THEN 11Ø
7Ø  IF A>B;B=A
8Ø  IF A<C;C=A
9Ø  I=I+1
1ØØ GOTO 5Ø
11Ø PRINT I;B;C
12Ø END
```

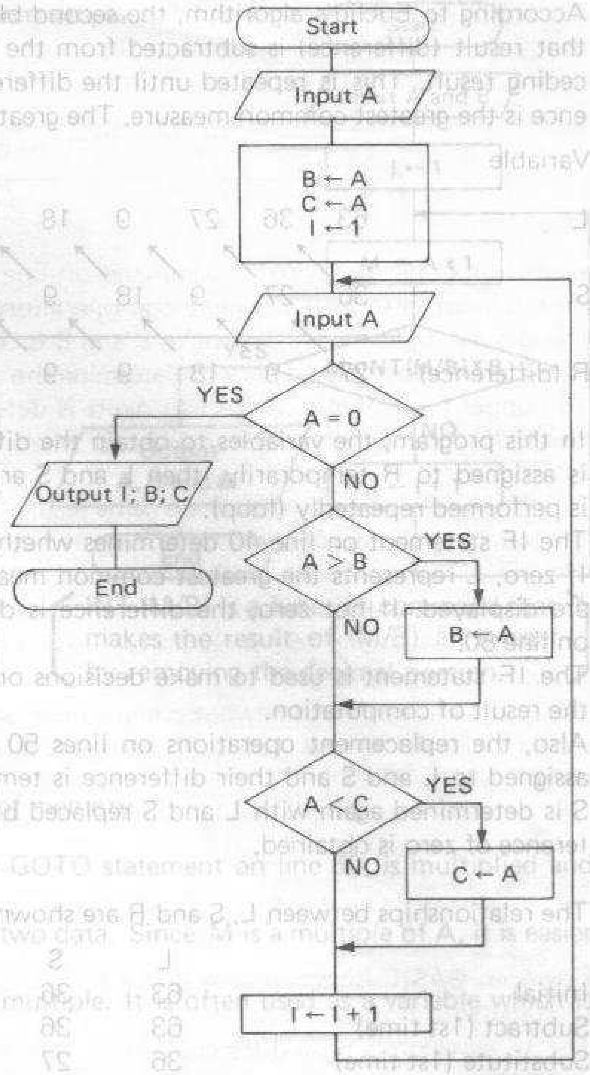
The INPUT statement on line 10 is used to input the initial data. Initial data is both the maximum value and the minimum value. Therefore, it is processed together with the data on line 20 and line 30 and the initial data is assigned with maximum value B and minimum value C.

Line 40 uses variable I to count the number of data. So, "1" is input as the initial data. The INPUT statement of line 50 is used to input second and further data. So, the procedure is repeated by the GOTO statement on line 100. If data "0" is input using the IF statement on line 60, it will cause termination.

In other words, if "0" is input using the INPUT statement on line 50, control jumps to the PRINT statement to output the result using line 110.

The IF statements on line 70 and 80 judge whether or not the input data is greater than the maximum value of the data already input or smaller than the minimum value and accomplishes replacement.

Upon completion of all data input, the number of data and the maximum and minimum value are displayed using line 110.



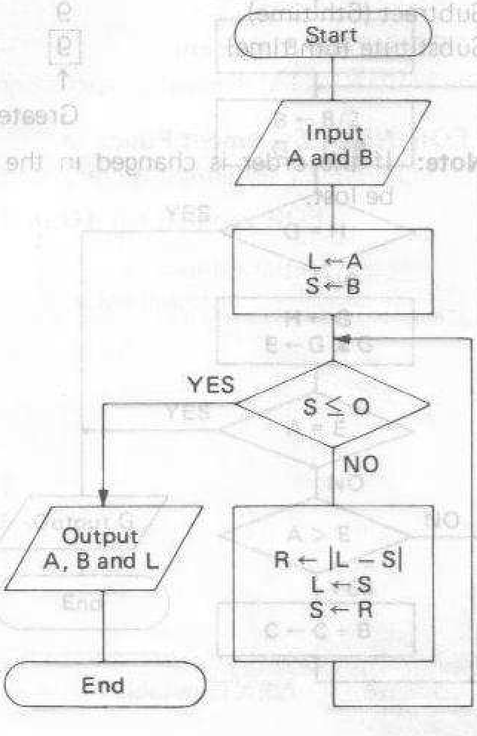
- **Various GOTO Statements and IF Statements**

Here we will take a look at some examples of applications using GOTO and IF statements.

Example 1: A program for determing the greatest common measure according to Euclid's algorithm.

```
 10 INPUT A,B
 20 L=A
 30 S=B
 40 IF S≤0 THEN 90
 50 R=ABS(L-S)
 60 L=S
 70 S=R
 80 GOTO 40
 90 PRINT A;B;L
100 END
```

Flow chart



According to Euclid's algorithm, the second block of data is subtracted from the first block of data. Then, that result (difference) is subtracted from the second block. Then, subtract that difference from the preceding result. This is repeated until the difference becomes zero. When this happens, the previous difference is the greatest common measure. The greatest common measure of 63 and 36 is shown below.

Variable							
L	63	36	27	9	18	$\boxed{9}$	← greatest common measure
S	36	27	9	18	9	9	
R (difference)	27	9	18	9	9	$\boxed{0}$	← when 0

In this program, the variables to obtain the differences are designated L and S respectively. The difference is assigned to R temporarily, then L and S are replaced by S and R respectively and the same operation is performed repeatedly (loop).

The IF statement on line 40 determines whether S, that is the difference after replacement, is zero or not. If zero, L represents the greatest common measure, and data A and B and their greatest common measure are displayed. If not zero, the difference is determined repeatedly as directed by the GOTO statement on line 80.

The IF statement is used to make decisions on which command (operation) is to be performed based on the result of computation.

Also, the replacement operations on lines 50 to 70 deserve special attention. Data A and B are initially assigned to L and S and their difference is temporarily assigned to R. Then the difference between L and S is determined again with L and S replaced by S and R respectively. This process is repeated until a difference of zero is obtained.

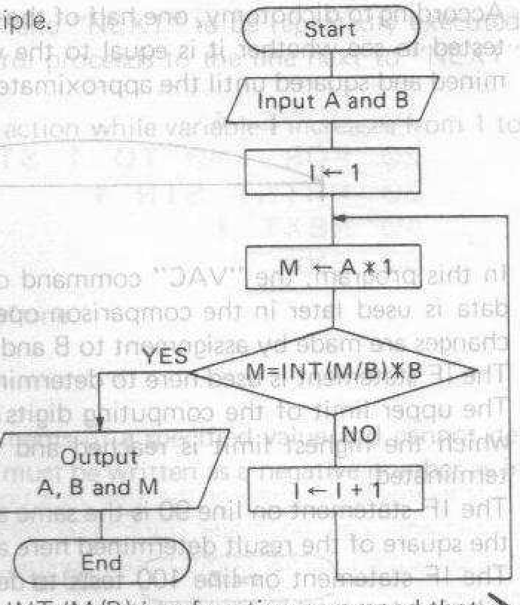
The relationships between L, S and R are shown below.

	L	S	R	
Initial	63	36	0	
Subtract (1st time)	63	36	27	R = ABS (L−S)
Substitute (1st time)	36	27	27	L = S , S=R
Subtract (2nd time)	36	27	9	R = ABS (L−S)
Substitute (2nd time)	27	9	9	L = S , S=R
Subtract (3rd time)	27	9	18	R = ABS (L−S)
Substitute (3rd time)	9	18	18	L = S , S=R
⋮		⋮		⋮
Subtract (6th time)	9	9	0	R = ABS (L−S)
Substitute (6th time)	$\boxed{9}$	0	0	L = S , S=R
	↑ Greatest common measure			

Note: If the order is changed in the above process, the significance of the substitution operations will be lost.

Example 2: Make a program to determine the least common multiple.

```
10 INPUT A,B
20 I=1
30 M=A*I
40 IF M=INT(M/B)*B THEN 70
50 I=I+1
60 GOTO 30
70 PRINT A,B,M
80 END
```



INT (M/B) is a function command that makes the result of (M/B) an integer by removing the decimal portion.

This example program first inputs data A and data B and multiplies the initial value of A by one, two, three and so on until the result of that multiplication becomes equal to the least multiple of the value of B as tested by an IF statement.

The IF statement on line 40 compares variable "M" with the numerical expression "INT (M/B) *B". In other words, variable "M" is compared with the result of the numerical expression "INT (M/B) *B". This method is the same as:

```
N=INT(M/B)*B
IF M=N THEN 70
```

However, the method used here saves lines and the amount of memory.

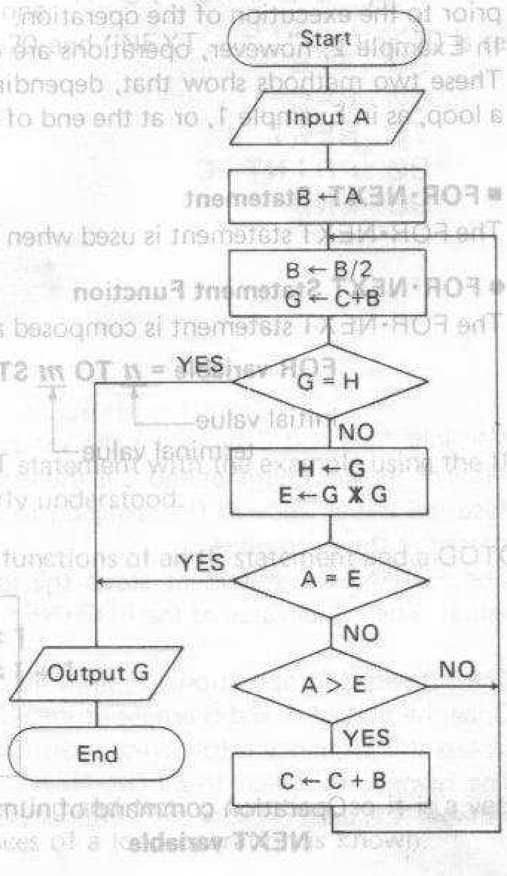
Then, multiplier I is increased one by one and, using the GOTO statement on line 60, is multiplied and judged again.

The variables used here are A, B, M and I. A and B are the two data. Since M is a multiple of A, it is easier to understand the value of M later.

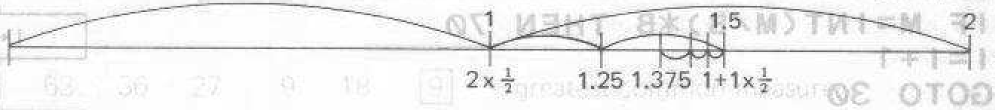
Multiplier I is incremented one by one to produce a new multiple. It is often used as a variable which is incremented upon each occurrence of looping.

Example 3: Make a program to determine the square root according to dichotomy. The program is as shown below.

```
 10 VAC
 20 INPUT A
 30 B=A
 40 B=B/2
 50 G=C+B
 60 IF G=H THEN 120
 70 H=G
 80 E=G*G
 90 IF A=E THEN 120
100 IF A>E;C=C+B
110 GOTO 40
120 PRINT G
130 END
```



According to dichotomy, one half of the value of data A is first determined and the square of the one half is tested to see whether it is equal to the value of A or not. If not, one half of the one half is further determined and squared until the approximate value of A is obtained. The computation process is shown below.



In this program, the "VAC" command on line 10 is used to clear the memory. Then data A is input. This data is used later in the comparison operations of the IF statements. Direct changes cannot be made. So changes are made by assignment to B and B is changed.

The IF statement is used here to determine the approximate value at line 60.

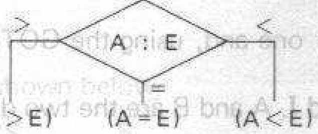
The upper limit of the computing digits is determined. Then G and H are used to determine the point at which the highest limit is reached and when the result is equal to the previous result, the operation is terminated.

The IF statement on line 90 is the same as the IF statement in example 2. If the square root can be divided, the square of the result determined here and data A will be equal and the result will be displayed.

The IF statement on line 100 tests to determine in which of the two equally divided parts of the result the solution (approximate value) lies.

The IF statements in this program are used differently from before, but the function of each IF statement is the same as before.

The program itself is somewhat complicated, so a combination of IF statements is required. The IF statements on lines 90 and 100 both compare A and E with each other. The flowchart for this operation is shown below.



However, BASIC does not provide for the use of three-branch IF statements. Instead, a combination of dual IF statements is used.

The IF statements have different effects depending on where they are used in a program, as in Example 1 and Example 2.

In Example 1, an IF statement is used at the beginning of the loop so that it may cause a jump to END prior to the execution of the operation.

In Example 2, however, operations are executed and then testing of the specified condition is performed. These two methods show that, depending on the data, the IF statement can be entered at the beginning of a loop, as in Example 1, or at the end of an operation, as in Example 2.

■ FOR·NEXT Statement

The FOR·NEXT statement is used when the number of occurrences of the same loop operation is known.

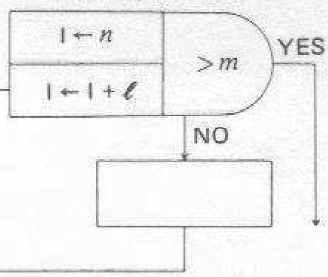
● FOR·NEXT Statement Function

The FOR·NEXT statement is composed as shown below.

FOR variable = n TO m STEP ℓ

initial value —— n
terminal value —— m
increment ⇒ variable or numeric expression —— ℓ

Flow chart



Operation command of numeric expression, etc.

NEXT variable

This statement commands the action specified between "FOR" and "NEXT" to be repeatedly executed while a variable changes from n to m in increments of ℓ. Control proceeds to the line next to "NEXT" when the variable is changed to m.

For example, the program below shows how to execute a given action while variable I increases from 1 to 10 in increments of 2.

```
FOR I = 1 TO 10 STEP 2
.....................
NEXT I
```

This STEP can be omitted if the variable increases in increments of one.

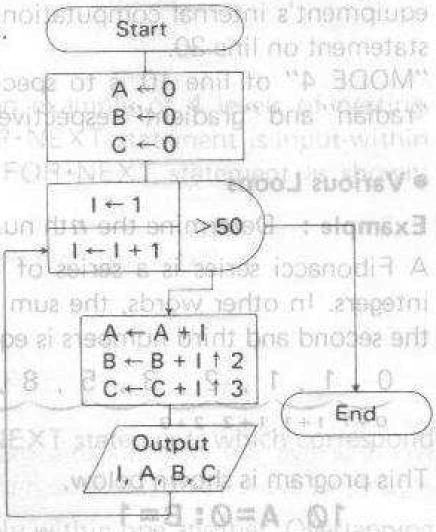
```
FOR I = 1 TO 10 STEP 1   is the same as
FOR I = 1 TO 10
```

* The FOR•NEXT statement only increases a variable in increments of a specified value and cannot decrease it. If a variable is to be decreased from 10 to 1, then it must be written as a negative number as in "STEP −1".

Example 1: Make a table of $\sum_{i=1}^{n} i$, $\sum_{i=1}^{n} i^2$, $\sum_{i=1}^{n} i^3$, with n ranging from 1 to 50.

This program is shown below.

```
10 VAC
20 FOR I=1 TO 50 STEP 1
30 A=A+I
40 B=B+I↑2
50 C=C+I↑3
60 PRINT I,A,B,C
70 NEXT I
80 END
```



This example can only be done using a FOR•NEXT statement. The totals of i, i^2 and i^3 ($\sum_{i=1}^{n}$) are displayed while variable n is increased from 1 to 50 in increments of one.

Namely, the operation specified between "FOR . . . . ." on line 20 and "NEXT . . . . ." on line 70 is repeated 50 times as I is increased from 1 to 50 in increments of 1.

This example can be accomplished using an IF statement.

```
10 VAC
20 I=1
30 A=A+I
40 B=B+I↑2
50 C=C+I↑3
60 PRINT I,A,B,C
70 I=I+1
80 IF I=51;END
90 GOTO 30
```

In this manner, if we compare the example using the FOR•NEXT statement with the example using the IF statement, the function of the FOR•NEXT statement will be clearly understood.

In other words, the FOR•NEXT statement combines the testing functions of an IF statement and a GOTO statement as well as an incrementing function.

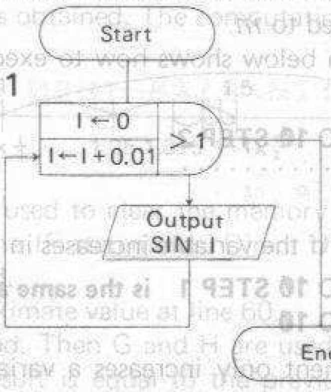
```
20 FOR I=1 TO 50 STEP 1 ⇨ { 20 I=1
                          { 70 I=I+1

80 NEXT I ⇨ { 80 IF I=51;END
            { 90 GOTO 30
```

In this manner, the FOR•NEXT statement has both an incrementing and testing function, so it is a very flexible and convenient command when the number of occurrences of a loop operation is known.

Example 2: Make a program to produce a table of sine functions from 0 to 1 in increments of 0.01.

This program is shown below.

```
10 MODE 4
20 FOR I=Ø TO 1 STEP Ø.Ø1
30 PRINT SIN I
40 NEXT I
50 END
```



This example determines the sine functions from 0 to 1 in increments of 0.01. The FOR•NEXT statement increments can be made using 0.01 increments. This incrementation can be accomplished because this equipment's internal computation system uses a decimal system. SIN I can be obtained and input using the statement on line 30.

"MODE 4" of line 10 is to specify "degree" for the unit of angle. "MODE 5" and "MODE 6" specify "radian" and "gradient" respectively.

- **Various Loops**

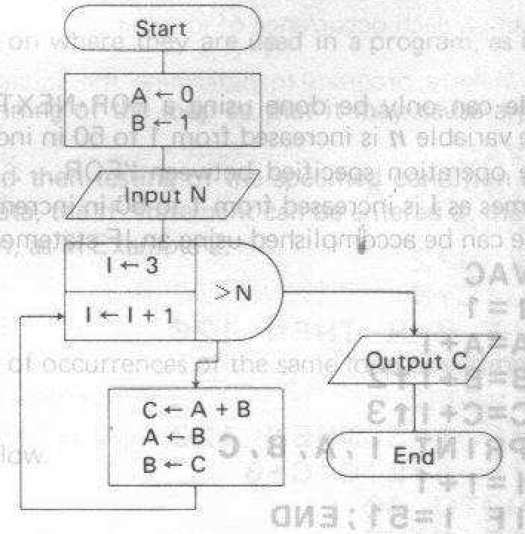
Example : Determine the *n*th number in a Fibonacci series.

A Fibonacci series is a series of integers in which each integer is equal to the sum of the two preceding integers. In other words, the sum of the first and second numbers is equal to the third number, the sum of the second and third numbers is equal to the fourth number, etc.

$$\underbrace{0 , 1}_{0+1} , \underbrace{1 , 1}_{1+1} , \underbrace{1 , 2}_{1+2} , \underbrace{2 , 3}_{2+3} , 5 , 8 , 13, 21, 34,$$

This program is shown below.

```
10 A=Ø: B=1
20 INPUT  N
30 FOR  I=3  TO  N
40 C=A+B
50 A=B
60 B=C
70 NEXT  I
80 PRINT  C
90 END
```



Variable A, B and C represent the variables in the series. A is the value of the preceding number, B is the value of the next number and C is the total (the number following B).

Also, an initial value of 0 is assigned to A and an initial value of 1 is assigned to B. Then the operation is started at the third position.

The FOR•NEXT statement starts the loop beginning at 3 and continues it up to the desired point (Nth value). The initial value of the FOR•NEXT statement does not necessarily start from 1.

The substitution operations on lines 40 to 60 require special attention. The sequence cannot be changed. Once the sum of A and B are input into C, then B is input into A, then C is input into B.

Unless this sequence is followed, substitution cannot be performed correctly.

The manner in which this FOR•NEXT statement is used differs from the previous example. Therefore, variable N is used because the terminal value is changed based on later inputs.

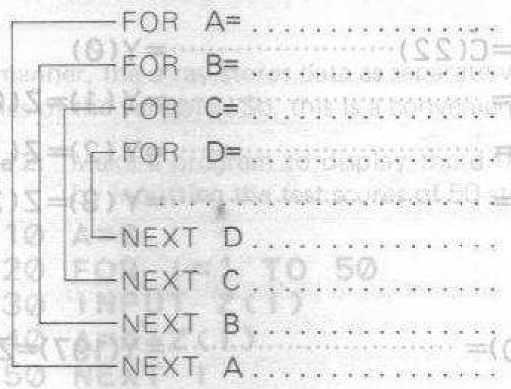
This example can be accomplished using an IF statement. A sample program using an IF statement is shown below.

```
10 A=Ø: B=1: I=3
20 INPUT N
30 C=A+B
40 A=B
50 B=C
60 IF I≠N; I=I+1: GOTO 3Ø
70 PRINT C
80 END
```

In this program, the IF statement on line 60 increments variable I while effecting the function of a GOTO statement. Initial values are set using a multistatement on line 10.

- **Nesting**

FOR·NEXT loops can be used to embed up to 4 levels. This embedding is called "nesting".



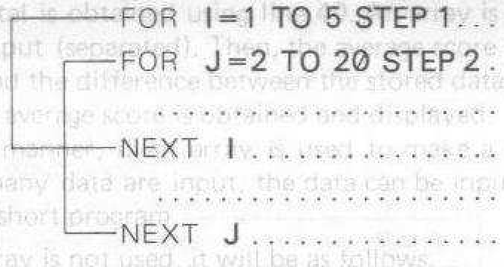
```
FOR  A= ................
  FOR  B= ................
    FOR  C= ................
      FOR  D= ................
        ..................
      NEXT  D ................
    NEXT  C ................
  NEXT  B ................
NEXT  A ................
```

This is an example of 4 levels of nesting. One FOR·NEXT statement is input within another FOR·NEXT statement as shown.

When embedding a number of these, care must be used concerning the NEXT statements which correspond to their FOR statements and their variables.

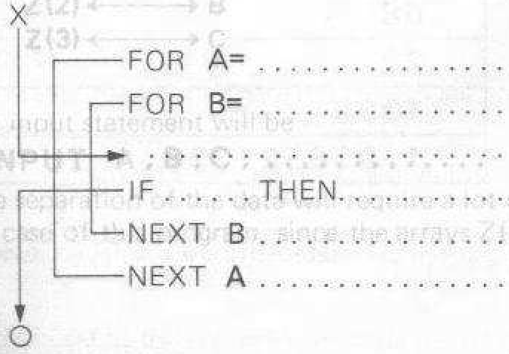
Also, the nesting must be done as shown above with the loops completely within one another. Overlapping FOR·NEXT loops cannot be used with a portion of the loop falling outside as in the following example of an incorrect nesting.

This type of FOR·NEXT loop cannot be used.



```
FOR  I=1 TO 5 STEP 1....
  FOR  J=2 TO 20 STEP 2.
    ....................
NEXT  I..................
    ....................
  NEXT  J ................
```

Furthermore, control can exit a FOR·NEXT loop but cannot enter a FOR·NEXT loop.



```
X
FOR  A= ................
  FOR  B= ................
    ....................
    IF ........THEN.......
  NEXT  B ................
NEXT  A ................
O
```

4-5-2 Arrays

For arrays, one-dimensional arrays are used with letters attached such as A(*i*), B(*j*), etc. Since these arrays are used both with the normal 26 memories and with expanded memories, pay attention to the following array arrangement.

```
                    A=A(0)
                    B=A(1)=B(0)
                    C=A(2)=B(1)=C(0)
                    D=A(3)=B(2)=C(1)=D(0)
                    E=A(4)=B(3)=C(2)=D(1)=E(0)
                       :
                    Y=A(24)=B(23)=C(22)·················=Y(0)
                    Z=A(25)=B(24)= ·····················=Y(1)=Z(0)
                   ⎧  A(26)=B(25)= ·····················=Y(2)=Z(1)
                   ⎪  A(27)=B(26)= ·····················=Y(3)=Z(2)
Expanded memories ⎨       :
                   ⎪      :
                   ⎩  A(221)=B(220)= ···················=Y(197)=Z(196)
```

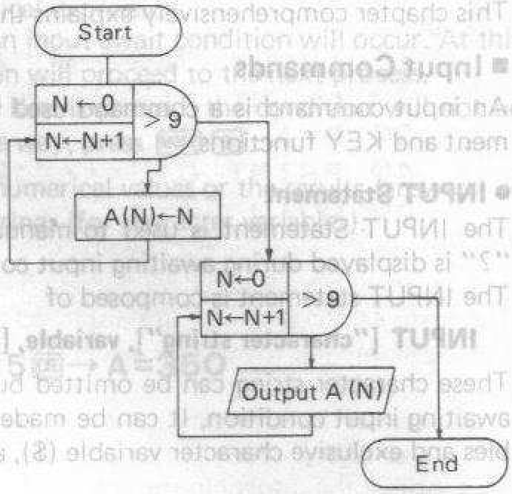
When arrays are used in this manner, since the same memory may be used depending on the array argument, avoid using the same memory in the same program.

Example:
Can be used at the same time .... A, B, C, F(0), F(9)
Cannot be used at the same time .... F, G, A(5), A(6)

Perform memory expansion correctly according to the size of the array.

Example 1: Make a program to display the value of *i* in a one dimensional array A(*i*) as it changes from 0 to 9.

Flow chart



This program is shown below.

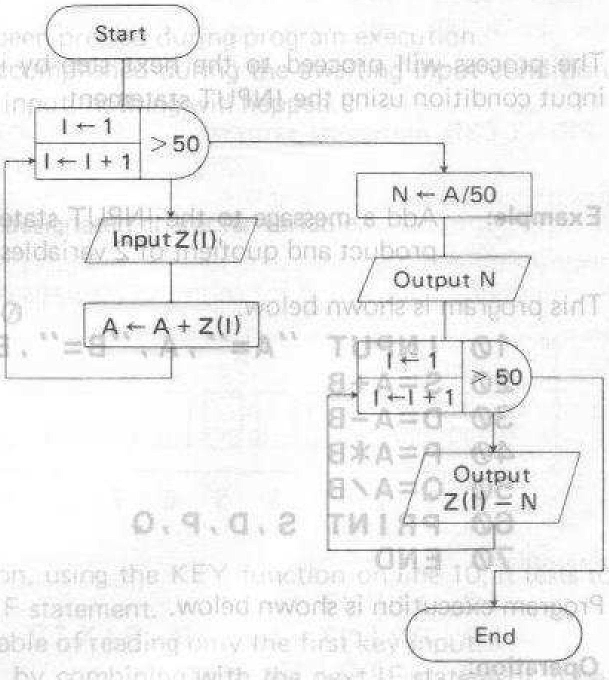
```
10 FOR N=Ø TO 9
20 A(N)=N
30 NEXT N
40 FOR N=Ø TO 9
50 PRINT A(N)
60 NEXT N
70 END
```

The FOR·NEXT statements on lines 10 to 30 have been explained previously. However, A(N) is used on line 20 and a value of N is assigned. Also, the FOR·NEXT statement on lines 40 to 60, in a similar manner, display the value of the array contents using the PRINT statement on line 50.

In this manner, the array stores data as separate variables simply by changing the elements without changing the names of the variables. So, this is a convenient function when used with the FOR·NEXT statement.

Example 2: Make a program to display the difference between the average score and the individual scores by inputting the test scores of 50 students.

```
 10 A=Ø
 20 FOR I=1 TO 5Ø
 30 INPUT Z(I)
 40 A=A+Z(I)
 50 NEXT I
 60 N=A/5Ø
 70 PRINT N
 80 FOR I=1 TO 5Ø
 90 PRINT Z(I)-N
100 NEXT I
110 END
```



For this program, the scores are input on line 30 using array Z(I).

Sum total is obtained using line 40. An array is used for score input (separated). Then, the average score is calculated and the difference between the stored data (scores) and the average score is obtained and displayed.

In this manner, if an array is used to make a program when many data are input, the data can be input simply using a short program.

If an array is not used, it will be as follows:

Z(1) ←——→ A
Z(2) ←——→ B
Z(3) ←——→ C
⋮ ⋮

and the input statement will be

```
INPUT A,B,C,..........
```

and the separation of the data will require a lot of time.

In the case of this program, since the arrays Z(1) to Z(50) are used, it is necessary to expand the memories to 50.

4-5-3 Input/Output Commands

This chapter comprehensively explains the input/output commands used with this equipment.

■ Input Commands

An input command is a command used to input data during program execution and uses the INPUT statement and KEY functions.

● INPUT Statement

The INPUT Statement is used to manually input data into the variables during program execution when "?" is displayed during awaiting input condition.
The INPUT statement is composed of

INPUT ["character string"], variable, ["character string"], variable . . . .

These character string can be omitted but once they are input, the character string is displayed before the awaiting input condition. It can be made into a message. Variables are numerical variables, character variables and exclusive character variable ($), and input data are assigned.

Example:

When	`INPUT A`	?
When	`INPUT "A=",A`	A=?

The process will proceed to the next step by inputting data and pressing the EXE Key during the awaiting input condition using the INPUT statement.

Example: Add a message to the INPUT statement for the program for determining the sum, difference, product and quotient of 2 variables.

This program is shown below.

```
10 INPUT "A=",A,"B=",B
20 S=A+B
30 D=A-B
40 P=A*B
50 Q=A/B
60 PRINT S,D,P,Q
70 END
```

Program execution is shown below.

Operation:

RUN EXE	A=?
45 EXE	B=?
23 EXE	68
EXE	22
EXE	1035
EXE	1.956521739

In this manner, the character strings which are written in the INPUT statement are displayed as messages and key input is easier.

For this INPUT statement, if character data is input for the numerical variables, an error will occur. If the error is cancelled using the AC Key, "?" will be displayed again and return to awaiting input condition.
Also, the input for the numerical variable can be input as a numerical expression.
As a result of an INPUT statement, a " ? " will be displayed and an input await condition will occur. At this time, if data is input and the EXE Key is pressed, program execution will proceed to the next process.
Furthermore, when in an input await condition, even if the AC Key is pressed, the condition will not be released. Therefore, when you want to stop the program along the way, press MODE 0.

* Data which can be input using an INPUT statement includes numerical values or the results (answers) of numerical expression (for numerical variables) and character strings (for character variables).

In the case of INPUT A

```
Numerical value ..........123EXE→A=123
Result of a numerical expression ..........14*25EXE→A=350
```

In the case of INPUT B$

```
Character string ........ABCEXE→B$=ABC
                         789EXE→B$=789
```

Furthermore, other numerical variables can also be used as input for numerical variables.

In the case of INPUT A (make X = 987654)

```
Variable ........XEXE→A=X
                     =987654
```

• KEY Functions

This function reads one character for each key which has been pressed during program execution.
This function, unlike the INPUT statement, cannot be accomplished during the awaiting input condition.
The program progresses as scheduled, so if there is no key input, nothing will happen.
The KEY function is composed as shown below.

Character variable = KEY

The character read by the KEY function is assigned to the designated character variable.

Example:

```
10 A$=KEY:IF A$="" THEN 10
20 IF A$="A" THEN 100
30 IF A$="B" THEN 200
40 IF A$="C" THEN 300
50 GOTO 10
   :
   :
```

This program is only for data input and separation portion, using the KEY function on line 10, it tests to see if the character data read was keyed in using the next IF statement.
Even if the EXE Key is not pressed, this KEY function is capable of reading only the first key input.
However, it does not stop like the INPUT statement. So, by combining with the next IF statement, it becomes an awaiting input condition.
Also, lines 20 to 40 compare the contents in the read character variables and determines the jump destinations. In this manner, the KEY function can only read the character of one key.

■ Output Commands

In the output command, there is a PRINT statement to display calculation results.

• PRINT Statement

A PRINT statement is composed as shown below.

PRINT [**Output control function CSR**] { **Numerical expression** / **Character expression** } [{ , / ; } ....]

{ } either one of the enclosed contents can be used.
[ ] enclosed contents can be omitted.

This PRINT statement displays the value of the numerical expression or the numerical variable and displays the character strings enclosed with " " or character variable contents.

Example:

```
10 INPUT "A=",A,"B=",B
20 C=INT (A/B)
30 D=A-B*C
40 PRINT C;"...";D
50 GOTO 10
```

This program computes the remainder, as in the quotient and remainder when A is divided by B.
The values of the variables are displayed in the manner as on line 40.
The items enclosed by " " are displayed as the characters themselves.
Furthermore, the manner in which the " , " and " ; " are displayed between the variables and character strings is different.
In the case of " ; ", the next data is displayed following the previous display.
In the case of " , ", the next data is displayed after the previous display is cleared once.
To continue the display in the case of " , ", press the EXE Key.

■ Output Control Functions

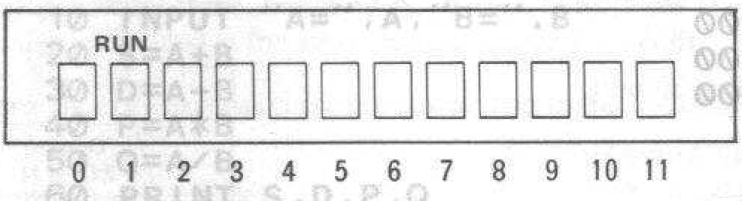
Output control functions designate the location of the output.

● CSR Function

This is a control function which is used in a PRINT statement. It designates the location of the output on the display (12 positions) using the PRINT statement.
The format of the CSR function is shown below.

PRINT CSR numerical expression (The decimal portion of the numerical expression is cut off and the value is 0 to 11.)

It uses the value of the numerical expression to determine at which unit from the left of the display to begin to output the data.
Furthermore, the method for counting the units on the display is shown below.



Example:

```
10 INPUT X
20 PRINT X;CSR 5;X↑2
30 GOTO 10
```

This program obtains the square of data already input and displays the result as well as the data. The CSR function is used in the PRINT statement on line 20.
The initial data "X" is displayed as is from the left side and the next "X↑2" is displayed from the fifth position from the left using the CSR function.

The program, when executed, will be as shown below.

Operation:

```
RUN EXE    ?
  7 EXE    7      49
    EXE    ?
 45 EXE    45     2025
    EXE    ?
852 EXE    852    725904
```

In this manner, since the two types of output are constantly displayed from a certain position, it is easier to see.

* When there is a ";" following "CSR", it is displayed from the designated position but when "," is used, display will not be made unless the display is cleared once. So the display will not be made continuously.

● SET Statement

The SET statement explained the number of effective positions designation and decimal designation for manual calculation but here we will explain how designation is made by writing in the program.

In the SET statement example shown below, E designates the number of effective positions, F designates the decimal and N cancels the designations.

$$\text{SET} \begin{Bmatrix} \text{E}\,n \\ \text{F}\,n \\ \text{N} \end{Bmatrix} \qquad (n \text{ is } 0 \text{ to } 9)$$

Example 1:

```
10 SET E4
20 MODE 4
30 FOR X=Ø TO 18Ø STEP 5
40 PRINT SIN X
5Ø NEXT X
6Ø SET N
7Ø END
```

This program obtains sin x in 5 degree increments of x from 0 to 180 degrees.

In order to obtain the number of effective positions up to 4 positions in this example, sin$x$ is obtained using "SET E4".

If the SET statement is written at the beginning of the program, all of the subsequent displays (result output) will be displayed with an effective number of positions of 4 positions.

If calculation is continued, the result can only be obtained up to 4 effective positions. So, "SET N" is used on line 60 in order to cancel the designation.

Example 2: Designate the number of decimal positions to be 4 digits for the program in Example 1.

```
1Ø SET F4
2Ø MODE 4
3Ø FOR X=Ø TO 18Ø STEP 5
4Ø PRINT SIN X
5Ø NEXT X
6Ø SET N
7Ø END
```

In this manner, the number of output positions can be controlled using the SET statement similar to manual calculation.

4-5-4 Character Functions

Character functions are related to character variables (A$, B$, $, etc.). The handling method for these is different because they are used to input characters instead of numeric values. Therefore, these character functions determine the size of the character variables, extract some characters from character strings in character variables and transform number in character variables to numerical values.

● LEN and MID

A LEN function is a command to count the number of characters in a character variable. The size of the character variable can thus be known.
A MID function is a command to extract some characters from the exclusive character variable ($). It is used for rearranging the character variables.
Since the LEN function can tell the size of the character variable, it is a convenient means of reserving output space for character variables or for dividing them when the size is not fixed.

Example:

```
10 A$="123"
20 B$="456"
30 C$=A$+B$
40 D=LEN(C$)
50 PRINT D
60 END
```

This program displays the size (number of characters) of the character variable by adding character variables A$ and B$.
Character variables can only perform addition. However, since the size of the character variable is not known using that result, the LEN function is used to determine the size of that addition.
Only character variables can be used for the LEN function.

The format of the MID function is shown below.

MID (*m*[,*n*])   (*m, n*: numerical expression)

It extracts the *n* character portion from the *m*th character of the character string stored in the exclusive character variable ($).
If *n* is omitted, all the characters after the *m*th character will be extracted.
Furthermore, the decimal is cut off for both *m* and *n*, and their values are 1 to 30. And *m* must be smaller than the stored number of characters. If the sum of *m* and *n* is greater than the stored number of characters plus 1, an error will occur.

Example: when $ = "ABCDEFGHIJ"
To extract four characters from the third character on → MID (3,4) . . . CDEF
To extract two characters from the first character on → MID (1,2) . . . AB
To extract all characters from the fifth character on → MID (5) . . . EFGHIJ

Example:

```
10 INPUT $
20 N=LEN($)/2
30 X$=MID(1,N):Y$=MID(N+1)
40 PRINT X$,Y$
50 END
```

This program divides the input character strings in two using the LEN function and the MID function and displays.
This determines the size of the exclusive character variable $ on line 20. Those halves are assigned to N and the first half and the second half are divided using the MID function.
In this manner, when the size of the input character string is not fixed, the halves or thirds cannot be extracted, so the size is determined using the LEN function.
Furthermore, in this case, the character string that can be input into $ is limited to 30 characters.
However, since the number of characters in X$ and Y$ is limited to 7 characters, the number of characters input into $ is limited to 14 characters.

• VAL

The VAL function changes the numbers in a character variable into a numerical value.

Format: VAL (character variable)

Since this function changes the numbers in the character variable into a numerical value, when there are no numbers in the character variable (for example, "ABC"), an error will occur.

Example: If Z$ = "78963", VAL (Z$) = 78963

Note: When this function is used in a program and an error occurs as a result of the data in the variable being other than numbers, "ERR 2" will be displayed and the program area and line number will not be displayed.

4-5-5 Subroutines

A subroutine (also called a subprogram) has a main routine which is different from the programs (also called main routines) considered previously. The subroutine is called from the main routine and, upon completion of the operation, returns to the original position in the main routine.

In other words, using the command (GOSUB) to jump from within the main routine to the subroutine, it jumps from that position to the designated subroutine. After the subroutine operation is completed, it returns to the position (GOSUB command) in the main routine where the jump to the subroutine occurred and the main routine process is continued.

```
          :
┌──►100 GOSUB 1000        subroutine callout (jump) command
│         :
└──►1000 A=B            ┐
          :             ├ subroutine area
────1100 RETURN         ┘ command to return to main routine
```

The commands required for this subroutine are "GOSUB" and "RETURN" statements. The command to jump (callout) to the subroutine is the GOSUB statement and the command to return from the subroutine to the main routine is the RETURN statement.

This GOSUB statement is used in the following formats.

(1) **GOSUB numerical expression** (the numerical expression is cut off at the decimal and uses line numbers 1 to 9999)

(2) **GOSUB # numerical expression** (the numerical expression is cut off at the decimal and uses $0 \leq n < 10$)

The first method uses direct designation to write the line number directly following "GOSUB" and can indirectly designate the subroutine location using the value contained in the variable.

The second method uses a subroutine in another program area (P0 to P9) to designate directly and indirectly.

Furthermore, neither will return unless the RETURN statement is written at the end of the subroutine.

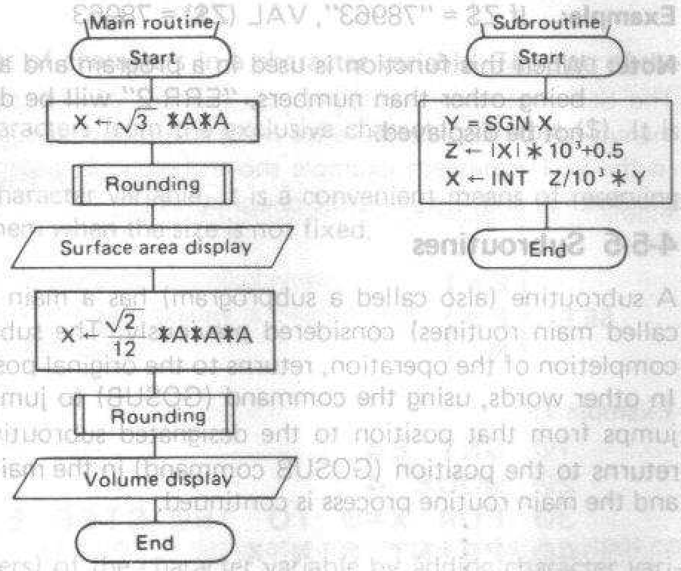
■ Subroutine Fundamental Programs

Here we will explain how to arrange the subroutines using actual program examples.

Example 1: Make a program to determine the surface area and volume of a regular tetrahedron, in which the length of the sides is A, computed down to three decimal places. Provided that a rounding program has been incorporated into a subroutine.

$$\left[\begin{array}{l} S = \sqrt{3}\,a^2 \\ V = \dfrac{\sqrt{2}}{12}\,a^3 \end{array}\right]$$

```
  10  INPUT  A
  20  X=SQR  3*A*A
  30  GOSUB  1000
  40  PRINT  X
  50  X=SQR  2/12*A*A*A
  60  GOSUB  1000
  70  PRINT  X
  80  END
1000  Y=SGN  X
1010  Z=ABS  X*10↑3+0.5
1020  X=INT  Z/10↑3*Y
1030  RETURN
```



The main routine for computing the surface area and volume of the regular tetrahedron can be easily understood, as it is similar to the programs considered previously.

The difference is the rounding program included in the main routine as a subroutine from line 1000 on. This subroutine has the function of computing the surface area and volume, as determined on lines 20 and 50, down to three decimal places by rounding.

The subroutine can be used as many times as desired.
It is used twice in this program, first for rounding the value of the surface area and second for rounding the value of the volume.
If a subroutine is not used and only a main routine is used as in the past, the program will be longer, as shown below.

```
 10  INPUT  A
 20  X=SQR  3*A*A
 30  Y=SGN  X
 40  Z=ABS  X*10↑3+0.5
 50  X=INT  Z/10↑3*Y
 60  PRINT  X
 70  X=SQR  2/12*A*A*A
 80  Y=SGN  X
 90  Z=ABS  X*10↑3+0.5
100  X=INT  Z/10↑3*Y
110  PRINT  X
120  END
```

This program and the program using the subroutine described above perform the same processing (operations) but the program using the subroutine is shorter and simpler.
In other words, the subroutine is used to simplify the program by eliminating repetitive operations.

* In this example, since the rounding program is incorporated into a subroutine, "PRINT X" is not included in the subroutine. Actually, in this program, lines 30 to 60 and lines 80 to 110 are the same and this portion is incorporated into a subroutine.

This subroutine is also a part of the main routine, so it is not incorporated within the main routine but is incorporated after the end ("END") and must be ended with a RETURN statement. If it is incorporated along the way in the main routine, it will be read as the main routine and processed repeatedly, thereby causing an error upon execution of the RETURN statement.
Therefore, as in this example, the subroutine area must be separated by starting it at line 1000 or 5000.
The subroutine is similar to the GOTO command (GOTO statement), and is considered a jump command but differs in that a RETURN statement is always attached so that it returns to the GOSUB statement after completion.

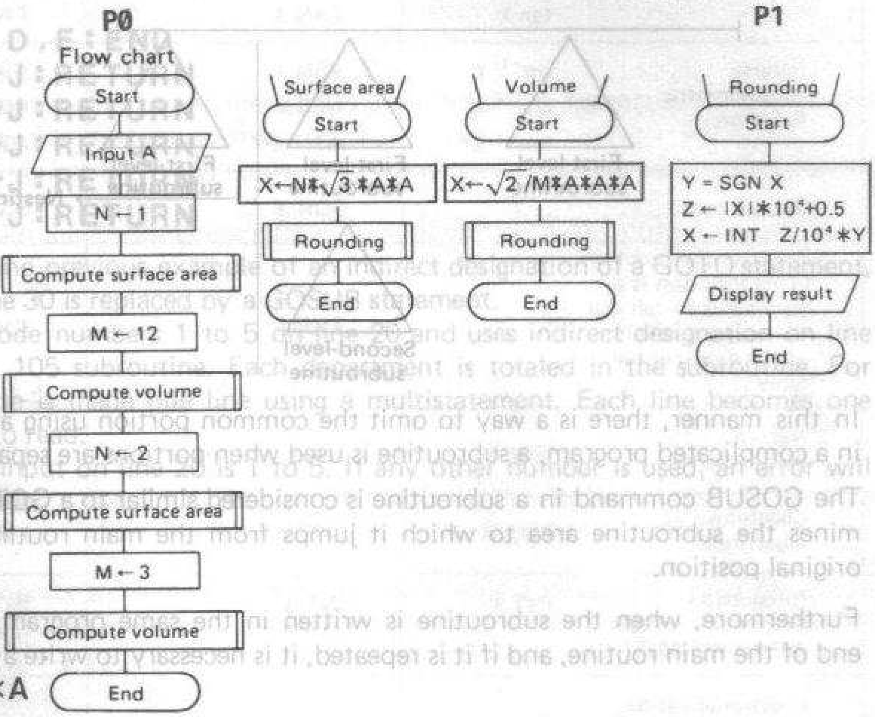
Example 2: Make a program to determine the surface area and volume of a regular tetrahedron and a regular octahedron in which the length of sides is A, computed down to four decimal places. Provided that the main routine and the subroutine for determining the area and volume are incorporated into P0 and the rounding subroutine is incorporated into P1.

Regular tetrahedron

$$S=\sqrt{3}\,a^2,\quad V=\frac{\sqrt{2}}{12}a^3$$

Regular octahedron

$$S=2\sqrt{3}\,a^2,\quad V=\frac{\sqrt{2}}{3}a^3$$



```
P0  10  INPUT A
    20  N=1
    30  GOSUB 1000
    40  M=12
    50  GOSUB 2000
    60  N=2
    70  GOSUB 1000
    80  M=3
    90  GOSUB 2000
   100  END
  1000  X=N*SQR 3*A*A
  1010  GOSUB #1
  1020  RETURN
  2000  X=SQR 2/M*A*A*A
  2010  GOSUB #1
  2020  RETURN
P1  10  Y=SGN X
    20  Z=ABS X*10↑4+0.5
    30  X=INT Z/10↑4*Y
    40  PRINT X
    50  RETURN
```

Three subroutines are used in this program: two common portions for computing surface areas and volumes, and a rounding program.

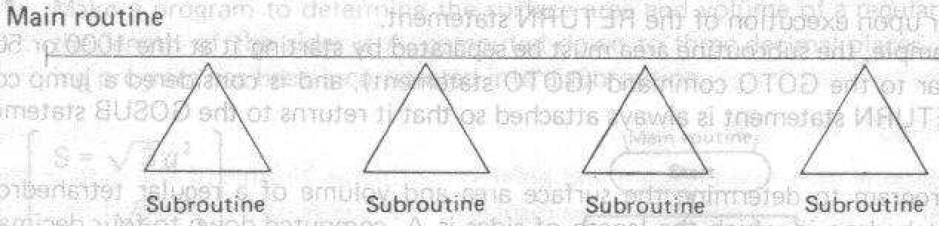
Using this method, commonly used computational operations can be incorporated into a subroutine for repetitive execution just by changing the values of N and M. In this manner, if common portions are incorporated into subroutines, memory can be saved and contents and computation method can be understood more easily.

Therefore, on lines 1010 and 2010 of this program, a subroutine is called out from a subroutine. This method is called "nesting." It calls out a subroutine from a subroutine similar to calling out a subroutine from a main routine.

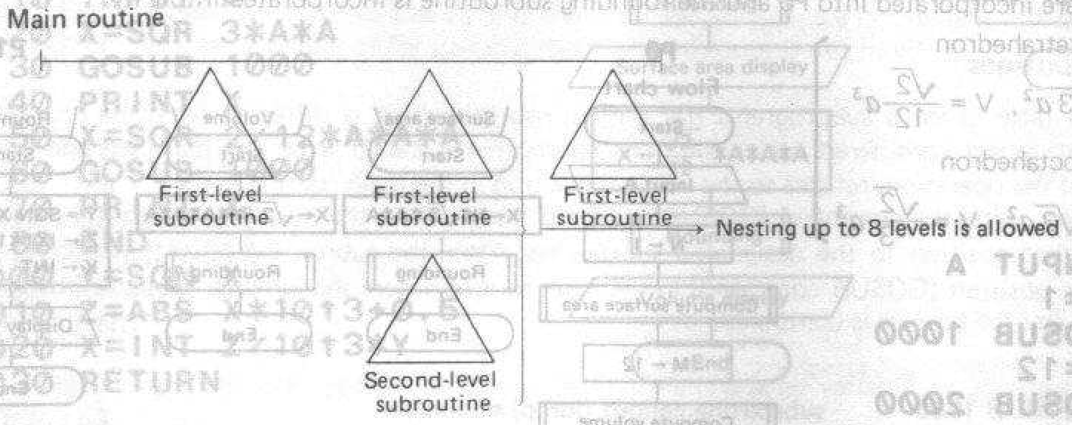
This nesting can be performed up to 8 levels (8 times), so any nesting beyond that causes an error.
In other words, it goes from the main routine to the first level subroutine. Then that subroutine acts as the main routine and the next (second level) subroutine is called out. Even at this time, the RETURN statement on which the subroutine is based must be written at the end of each subroutine without fail.
Similar to the rounding subroutine, this subroutine can be incorporated into another program area (in this case, P1). This method is convenient because it can be used as a subroutine in common with another program (a program written in another program area).

In this manner, many subroutines can be used in one program.

In the case of "nesting", however, up to 8 levels can be accomplished.



Using this method, many subroutines can be used.



In this manner, there is a way to omit the common portion using a subroutine. Also, when incorporated in a complicated program, a subroutine is used when portions are separated into groups.

The GOSUB command in a subroutine is considered similar to a GOTO statement. In other words, it determines the subroutine area to which it jumps from the main routine and then operates to return to the original position.

Furthermore, when the subroutine is written in the same program, it is necessary to write END at the end of the main routine, and if it is repeated, it is necessary to write a GOTO statement.

■ Indirect Designation of a Subroutine

Similar to the explanation for the GOTO statement, indirect designation can also be used for a subroutine.
The method of using is similar to that of a GOTO statement. It jumps to a designated subroutine and then returns to the original position.

Example 1:

```
10  INPUT N
20  GOSUB N+100
30  PRINT X*6
40  END
101  X=5:RETURN
102  X=10:RETURN
103  X=15:RETURN
104  X=20:RETURN
105  X=25:RETURN
106  X=30:RETURN
107  X=35:RETURN
108  X=40:RETURN
109  X=45:RETURN
110  X=50:RETURN
```

In this program, the subroutine is designated indirectly using the value of input N. When N is 1 to 10, it goes to a subroutine between 101 and 110 and determines the value of variable X and displays the computation result of X * 6.

In this manner, since the indirect designation of the GOSUB statement determines the jump destination (subroutine) using the value of the variable, when the variable is one, it jumps to the first subroutine (line 101).
When the variable is two, it jumps to the second subroutine (line 102), and so on. It designates indirectly depending on the value of the variable.

Example 2: Use the indirect designation of the GOSUB statement to make the sorted totals program which was given in a previous example and which used indirect designation of the GOTO statement (5 divisions).

```
 1Ø  VAC
 2Ø  INPUT  I
 3Ø  GOSUB  I+1ØØ
 4Ø  GOTO  2Ø
1ØØ  PRINT  A,B,C,D,E:END
1Ø1  INPUT  J:A=A+J:RETURN
1Ø2  INPUT  J:B=B+J:RETURN
1Ø3  INPUT  J:C=C+J:RETURN
1Ø4  INPUT  J:D=D+J:RETURN
1Ø5  INPUT  J:E=E+J:RETURN
```

This program is almost the same as the previous example of an indirect designation of a GOTO statement. However, the GOTO statement on line 30 is replaced by a GOSUB statement.
This method also basically inputs code numbers 1 to 5 on line 20 and uses indirect designation on line 30 and separates using lines 101 to 105 subroutine. Each department is totaled in the subroutine. For this program as well, each subroutine is made one line using a multistatement. Each line becomes one subroutine, thereby making it easier to read.
Even in this case, the code number input on line 20 is 1 to 5. If any other number is used, an error will occur when there is no jump destination.

4-5-6 General Functions

General functions include trigonometric functions such as sine, cosine and tangent, and built-in functions. These functions are formatted by combining alphabetic characters on the keyboard. They can also be entered with the "one-key command" using a single key and can be used as easily as operating a small electronic calculator.

General functions may be used manually or incorporated within programs.

Function name		Format	Example
Trigonometric function	sin x	SIN x	SIN **30** SIN **A** SIN **(N+5)**
	cos x	COS x	COS **3.14** COS **I** COS **(L−3)**
	tan x	TAN x	TAN **70** TAN **F** TAN **(F X 2)**
Inverse trigonometric function	$\sin^{-1} x$	ASN x	ASN **0.07** ASN **P** ASN **(Z+Y)**
	$\cos^{-1} x$	ACS x	ACS **π/5** ACS **D** ACS **(C−X)**
	$\tan^{-1} x$	ATN x	ATN **6.5** ATN **V** ATN **(Q+0.5)**
Square root	$\sqrt{x}$	SQR x	SQR **69** SQR **A** SQR **(R X S)**
Exponential function (This function is a command to call out the numerical value of the exponential table.)	e	EXP 1	EXP 1
Natural logarithm	$\log_e x$	LN x	LN **43** LN **P** LN **(U+T)**
Common logarithm	$\log_{10} x$	LOG x	LOG **24.6** LOG **R** LOG **(G+15)**
Integration (Maximum integer not exceeding x)	INT x	INT x	INT **347.457** INT **V** INT **(Q+U)** INT **−45.43**
Fractionalization (x with its integer part removed)	FRAC x	FRAC x	FRAC **73.54** FRAC **N** FRAC **(H+B)**
Absolute value	$\lvert x \rvert$	ABS x	ABS **−9.43** ABS **L** ABS **(K/P)**
Sign (If $x > 0$, 1) (If $x = 0$, 0) (If $x < 0$, −1)		SGN x	SGN **79** SGN **E** SGN **(P−0)**
Significant digit specification (x is determined down to the 10^y-th significant digit place by rounding)		RND (x, y)	RND **(123.456, 2)** RND **(A, C)** RND **(F+E, G−5)**
Random number generation (Uniform random number generation in the range $0 < x < 1$)		RAN #	RAN #
Unit of angular measure	Degree	MODE 4	One right angle = 90 degrees
	Radian	MODE 5	One right angle = $\frac{\pi}{2}$ radians
	Gradient	MODE 6	One right angle = 100 gradients

* x and y are constants, variables or numerical expressions.

Since these functions are built in they can be used in a program at once.

> EXP (e) cannot be used in a continuous calculation. Also, it cannot be used in a multistatement.

Example 1: Make a program to obtain the length of one side of a triangle using the angle enclosed by the other two sides.

$$[\, C = \sqrt{a^2 + b^2 - 2ab\cos\theta} \,]$$

```
10 MODE 4
20 INPUT A,B,C
30 D=SQR (A*A+B*B-2*A*B*COS C)
40 PRINT D
50 GOTO 20
```

Since the angular unit "degree" is used in the trigonometric function, MODE 4 appears on line 10 in this program. Then, the lengths of the two sides and the enclosed angle are input.
In accordance with the formula, square root SQR, and cosine COS are written in the numeric expression.

Example 2: Make a program to obtain the amplifier gain dB with input voltage X and output voltage Y.

$$[\, dB = 20 \cdot \log_{10} \frac{Y}{X} \,]$$

```
10 INPUT X,Y
20 Z=20*LOG (Y/X)
30 PRINT Z
40 GOTO 10
```

If the input voltage is X and the output voltage is Y, the formula can be written as shown on line 20 to obtain the gain (Z).

Example 3: Make a program to generate three-digit random numbers using a random number generating function and a significant digit specification function.

```
10 N=RND(RAN#,-4)*1000
20 PRINT N
30 END
```

Since random numbers can be generated as 10-digit mantissa in the range $0 < x < 1$, three digits must be taken as significant digits and multiplied by 1000 to extract three-digit numbers.

Example 4: Make a program to perform a calculation using the exponential function.

$$[\text{ Calculate } \frac{A + e^{1.5}}{B} \,]$$

```
10 E=EXP 1
20 INPUT A,B
30 C=(A+E↑1.5)/B
40 PRINT C
50 GOTO 20
```

The base (e) of natural logarithm is input into a variable (E) on line 10, and e^x is calculated using the value (variable E) and power function on line 30.

4-5-7 Option Specifications

■ Cassette magnetic tape

In order to record programs or data stored in this unit on a cassette tape, use the FA-3 cassette interface and an ordinary tape recorder. If the tape recorder has a remote terminal, remote control can be conveniently performed from the FX-700P through the FA-3.

For tape recorder connection procedures and detailed operating procedures, refer to the FA-3 operation manual.

- **Program recording**

Format: **SAVE** ["filename"] (Item in brackets may be omitted.)

The filename may be composed using 8 characters or less enclosed by quotation marks and may contain alphabetical letters/numbers/symbols.

Example:

```
"ABC"
"NO. 1"
```

This command starts the tape recorder in the RECORD position.

Operation:

SAVE ["filename"] EXE

A SAVE command can only be used manually.

- **Program callout**

Format: **LOAD** ["filename"] (Item in brackets may be omitted.)

This command starts the tape recorder in the PLAYBACK position.

Operation: LOAD ["filename"] EXE

Display during program load

```
PF : ABC
```

Program file filename

Even if a program has already been written in the designated program area prior to callout, the former program will be erased starting with the initial line number of the program to be loaded and the new load will be made correctly.

- **Recording of all programs**

Format: **SAVE A** ["filename"] (Item in brackets may be omitted.)

This command simultaneously records all of the programs which are written in all program areas from P0 through P9.

The operation method is similar to the SAVE command and the tape recorder is started in the RECORD position.

Operation:

SAVE A ["filename"] EXE

- **Callout of all programs**

Format: **LOAD A** ["filename"] (Item in brackets may be omitted.)

This command simultaneously calls out the programs of all the program areas which were recorded using the SAVE A command. The operation is similar to the LOAD command and the tape recorder is started in the PLAYBACK position.

Operation: **LOAD A** ["filename"] EXE

Display during program load

```
AF : FX-700P
```

All files filename

Even if programs are already written in program areas prior to callout, the former programs will be cleared and then the new programs will be called out.
Furthermore, both the SAVE A command and the LOAD A command can only be used manually.

- **Data recording**

Format: **PUT** ["filename"] variable 1 [, variable 2] (Items in brackets may be omitted.)

The data which is recorded on the tape is the data in the variables from variable 1 through variable 2.

Example: **PUT "PB" A** .... data of variable A
PUT "1-2" A, Z .... data of variables A through Z
PUT "DT" $, A, Z(10) .... data of character variable $ and variables A through Z(1Ø)

When recording the data in the exclusive character variable $, write $ first.
This command can be used either manually or by writing it in a program.
For manual operation, start the tape recorder in the RECORD position.

Operation:

PUT ["filename"] variable 1 [, variable 2] EXE

When performing by writing it in the program, write the PUT command along with the line number and start the written program.

- **Data callout**

Format: **GET** ["filename"] variable 1 [, variable 2] (Items in brackets may be omitted.)

This command can be used both manually and by writing it in a program.
For manual use, start the tape recorder in the PLAYBACK position and operate as follows.

Operation:
GET ["filename"] variable 1 [, variable 2] EXE

For use in a program, write it with a line number attached and start the program.

- **Checking of the file which has been recorded on the tape**

A VER command is used to check whether the programs or data have been recorded properly.

Format: **VER** ["filename"] (Item in brackets may be omitted.)

The operation sequence is similar to program load.

■ Printer

An exclusive mini printer can be connected to the FX-700P. By connecting this printer, program lists or data can be extracted. Also, the output of calculation results during execution can be printed out.
For printer connection procedures and operating procedures, refer to the mini printer operation manual.
To print a program list, press MODE 7 and designate the PRT mode.

Program list
MODE 0 MODE 7
LIST EXE or **LIST A** EXE
MODE 8 (PRT mode release)

After printout is complete, be sure to press MODE 8 and release the PRT mode.
Also, to print calculation results or operation contents, printout can be performed automatically by writing "MODE 7" and "MODE 8" in the program.

Example:

```
  ⌇
100 MODE 7
110 PRINT A
120 MODE 8
```

When "MODE 7" is written in the program, be sure to write "MODE 8" prior to program termination and release the PRT mode.

Error Message List

Error code	Meaning	Cause	Corrective measure
1	Memory overflow or system stack overflow.	● Number of steps insufficient. Program cannot be written in. ● Stack overflow	● Clear unneeded programs or reduce the number of memories. ● Divide the formulas and make them simpler.
2	Syntax error	● Format error in program, etc. ● Left-hand and right-hand formats differ in an assignment statement, etc.	● Correct error in input program, etc.
3	Mathematical error	● The result of a numeric expression calculation exceeds 10^{100} ● Outside the numeric function argument input range ● Result is indefinite or impossible	● Correct the calculation formula or data ● Verify the data
4	Undefined line number error	● No designated line number for GOTO statement or GOSUB statement	● Correct the designated line number
5	Argument error	● For a command function that requires an argument, the argument is outside the input range.	● Correct the argument error
6	Variable error	● Attempt was made to use a memory which has not been expanded. ● Attempt was made to use the same memory for a numerical variable and a character variable at the same time.	● Expand the memory properly. ● Do not use the same memory for a numerical variable and a character variable at the same time.
7	Nesting error	● RETURN statement comes out when subroutine is not being executed ● NEXT statement comes out when not in FOR loop ● Subroutine levels exceed 8 levels ● FOR•NEXT loops exceed 4 levels	● Remove unneeded RETURN statements or NEXT statements ● Keep the subroutines or FOR•NEXT statement loops within required levels
9	Option error	● No printer or tape recorder is connected and execution is attempted in the PRT mode or optional command such as SAVE is executed	● Connect printer or tape recorder ● Cancel PRT mode

Program Command List

Classification	Command name	Format	Function
Input statement	INPUT	INPUT variable string	Causes data to be entered from the keyboard during execution of a program. The program execution is stopped until after the end of input. **P. 51**
	KEY	Character variable = KEY	Reads a character entered during execution of a program and assigns it to a character variable. Since the program is not stopped by this command, nothing is assigned to the character variable if no key-in entry is made. **P. 52**
Output statement	PRINT	PRINT output control function $\left\{\begin{matrix};\\,\end{matrix}\right\}$ output element $\left[\left\{\begin{matrix};\\,\end{matrix}\right\}\cdots\cdots\right]$	Outputs a specified output element in a specified format. **P. 52**
	CSR	CSR $n\left\{\begin{matrix};\\,\end{matrix}\right\}$ $(0 \leqq n \leqq 11)$	Displays from the designated nth position. **P. 53**
Branching	GOTO	GOTO $\left\{\begin{matrix}\text{line number}\\\text{variable}\end{matrix}\right\}$	Causes control to jump to the specified line number. **P. 35**
	IF$\cdots\left\{\begin{matrix}\text{THEN}\\;\end{matrix}\right\}\cdots$	IF comparison $\left\{\begin{matrix}\text{THEN} & \text{line number}\\; & \text{command}\end{matrix}\right\}$	Causes control to jump to the line number following THEN, or executes the command following " ; ", if the result of the comparison is true. Causes control to proceed to the next line number if the result of the comparison is false. **P. 40**
	GOSUB	GOSUB $\left\{\begin{matrix}\text{line number}\\\text{variable}\end{matrix}\right\}$	Calls the subroutine with the specified line number for execution. After the subroutine is executed, control returns to the GOSUB statement by the RETURN statement to proceed to the command following that statement. **P. 56**
	RETURN	RETURN	Signifies the end of a subroutine; returns control to a line number next to the GOSUB statement. **P. 56**

Classification	Command name	Format	Function
Looping	FOR	FOR $v=e_1$ TO e_2 [STEP e_3] * v denotes a numerical variable, and e_1, e_2 and e_3 represent a numerical expression respectively.	Declares the beginning of a loop in which numerical value v changes from initial value e_1 to terminal value e_2 in increments of e_3. The loop is repeated " $\left[\frac{e_2-e_1}{e_3}\right]+1$ " times between the FOR and NEXT statements. If the increment e_3 is omitted, e_3 is regarded as "1". **P. 45**
	NEXT	NEXT v	Signifies the end of a FOR loop. If the result of v plus e_3 is equal to or smaller than e_2, the loop is repeated again. If it is greater than e_2, control proceeds to the line next to the NEXT statement. **P. 45**
Execution stop	STOP	STOP	Stops the execution of a program temporarily to bring the system into a key-in wait state. The execution can be continued by pressing the EXE key. **P. 33**
Execution end	END	END	Signifies the end of a program, the system returning to its pre-execution state. The execution of a program, once ended, cannot be continued even if the EXE key is pressed. **P. 26**
Data clearing	VAC	VAC	Clears all variable data for a program. **P. 38**
Program listing	LIST	LIST [line number]	Displays a listing of all the statements in a program from the specified line number downward. **P. 28**
Program/data listing	LIST A	LIST A	Displays a listing of the statements in all programs and the data in all memories. **P. 64**
Program execution	RUN	RUN [line number]	Causes a program to start from the specified line number. **P. 27**
Program erasing	CLEAR	CLEAR	Clears the currently specified program area of a program. **P. 26**
	CLEAR A	CLEAR A	Clears all the programs. **P. 26**

Classification	Command name	Format	Function
Angular unit designation	MODE	MODE $\begin{Bmatrix}4\\5\\6\end{Bmatrix}$	Designates trigonometric angular units as degree (4), radian (5) or gradient (6). **P. 47**
Format designation	SET	SET $\begin{Bmatrix}\mathrm{E}\,n\\\mathrm{F}\,n\\\mathrm{N}\end{Bmatrix}$ $(0 \leq n \leq 9)$	Designates the number of effective positions or number of decimal positions for the displayed numerical value. **P. 54**
Character function	LEN	LEN (character variable)	Calculates the size of the character variable. **P. 55**
	MID	MID $(m\,[,n])$	Extracts n characters from the mth character in the exclusive character variable ($). **P. 55**
	VAL	VAL (character variable)	Converts the numbers in a character variable to a numerical value. **P. 56**
Option use	SAVE	SAVE ["filename"]	Records only the program in the currently designated program area on tape. **P. 63**
	LOAD	LOAD ["filename"]	Calls out the program from the tape and loads it to the currently designated program area. **P. 63**
	SAVE A	SAVE A ["filename"]	Records the programs in all program areas on tape at the same time. **P. 63**
	LOAD A	LOAD A ["filename"]	Calls out all programs from the tape and loads them to the respective program areas. **P. 63**
	PUT	PUT ["filename"] variable	Records the data in the variable on tape. **P. 64**
	GET	GET ["filename"] variable	Calls out the data from the tape and loads it in the variable. **P. 64**
	VER	VER ["filename"]	Checks to confirm that the programs or data have been recorded on the tape properly. **P. 64**

* Items enclosed in [ ] may be omitted. Either one of the items enclosed in { } may be used.

Specifications

- **Type**

FX-700P

- **Fundamental calculation functions**

Negative numbers, exponentials, 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/gradient), logarithmic/exponential functions, square root, powers, conversion to integer, deletion of integer portion, absolute value, symbolization, designation of number of effective positions, designation of number of decimal positions, random numbers, π

- **Function digit capacity**

	Input range	Result accuracy
$\sin x$, $\cos x$, $\tan x$	$\lvert x\rvert < 1440°$ (8πrad, 1600gra)	10th digit ±1
$\sin^{-1} x$, $\cos^{-1} x$	$\lvert x\rvert \leq 1$	—"—
$\tan^{-1} x$		—"—
$\log x$, $\ln x$	$x > 0$	—"—
e^x	$x = 1$	—"—
$\sqrt{x}$	$x \geqq 0$	—"—
x^y $(x \uparrow y)$	$x > 0$	—"—

- **Commands**

INPUT, PRINT, GOTO, FOR•NEXT, IF-THEN, GOSUB, RETURN, STOP, END, RUN, LIST, LIST A, MODE, SET, VAC, CLEAR, CLEAR A, DEFM, SAVE, SAVE A, LOAD, LOAD A, PUT, GET, VER

- **Program functions**

KEY, CSR, LEN, MID, VAL

- **Calculation range**

$\pm 1 \times 10^{-99}$ to $\pm 9.999999999 \times 10^{99}$ and 0 (internal calculation uses 12 mantissa positions)

- **Program system**

Stored system

- **Program language**

BASIC

- **Number of steps**

Maximum 1,568 steps

- **Program capacity**

Maximum 10 programs (P0 through P9)

- **Number of memories**

26 to maximum 222 memories and exclusive character variable ($)

- **Number of stacks**

Subroutine — 8 levels
FOR•NEXT loop — 4 levels
Numerical value — 6 levels
Calculation elements — 12 levels

- **Display system and contents**

10 mantissa positions (including minus sign) or 8 mantissa positions (7 positions for negative number) and 2 exponential positions. Also, display of respective conditions such as EXT, S, F, RUN, WRT, DEG, RAD, GRA, TR, PRT, STOP

■ **Display elements**

12-position 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**

Mainframe only — approximately 300 hours (Continuous use)

■ **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 x 165W x 71mmD (3/8"H x 6-1/2"W x 2-3/4"D)

■ **Weight**

118 g (4.2 oz) including batteries

GUIDELINES LAID DOWN BY FCC RULES FOR USE OF THE UNIT IN THE U.S.A. (not applicable to other areas).

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

. . . reorient the receiving antenna

. . . relocate the computer with respect to the receiver

. . . move the computer away from the receiver

. . . plug the computer into a different outlet so that computer and receiver are on different branch circuits.

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

CASIO®

112B SA 英 Printed in Japan