

- (1) Specify the LR2 mode (**SHIFT** **MODE** **+**).
- (2) Set the range values to those shown in the table.

Range
Xmin:-10.
max:10.
scl:2.
Ymin:-5.
max:15.
scl:5.

* According to the general rule of the x -axis range values, the values for x are: $-10 \leq x < 10$.

- (3) Clear the statistical memories.

SHIFT **Sci** **EXE**

- (4) Input the data.

(-) 9 **SHIFT** **□** **(-)** 2 **DT**

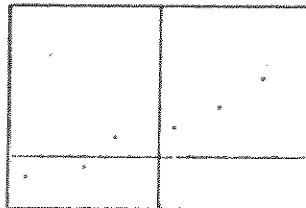
(-) 5 **SHIFT** **□** **(-)** 1 **DT**

(-) 3 **SHIFT** **□** 2 **DT**

1 **SHIFT** **□** 3 **DT**

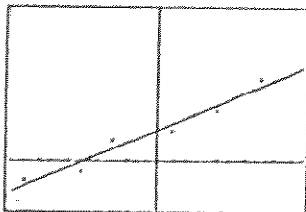
4 **SHIFT** **□** 5 **DT**

7 **SHIFT** **□** 8 **DT**



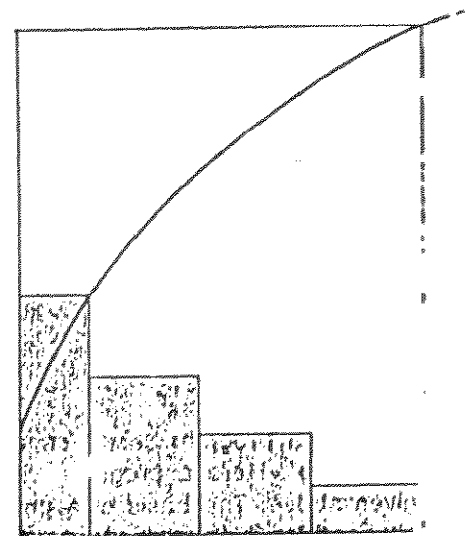
- (5) Draw the graph.

Graph **SHIFT** **Line** 1 **EXE**



- * When data is input that is outside of the preset range values, a point does not appear.
- * An Ma ERROR is generated when there is no data input and the following key operation is performed: **Graph** **SHIFT** **Line** 1 **EXE**.
- * The following must be true in the case of range settings:
 $Xmin < Xmax$.

4. PROGRAM COMPUTATIONS

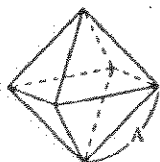


4-1 WHAT IS A PROGRAM?

This unit has a built-in program feature that facilitates repeat computations. The program feature is used for the consecutive execution of formulas in the same way as the "multistatement" feature is used in manual computations. Programs will be discussed here with the aid of illustrative examples.

EXAMPLE:

Find the surface area and volume of a regular octahedron when the length of one side is given.



Length of one side (A)	Surface area (S)	Volume (V)
10cm	() cm ²	() cm ³
7	()	()
15	()	()

* Fill in the parentheses.

① Formulas

For a surface area S , volume V and one side A , S and V for a regular octahedron are defined as:

$$S = 2\sqrt{3} A', \quad V = \frac{\sqrt{2}}{3} A'$$

② Programming

Creating a program based on computation formulas is known as "programming". Here a program will be created based upon the formulas given above. The basis of a program is manual computation, so first of all, consider the operational method used for manual computation.

Surface area (S): 2 3 Numeric value A

Volume (V): 2 3 Numeric value A 3



In the above example, numeric value A is used twice, so it should make sense to store it in memory A before the computations.

Numeric value A → ALPHA  EXE

☐ ☒ 3 ☒ ALPHA ☐ ☒ EXE *****S

2 3 X ALPHA 3 EXE

With this unit, the operations performed for manual computations can be used as they are in a program. Once program execution starts, it will continue in order without stopping. Therefore, commands are required to request the input of data and to display results. The command to request data input is "?", while that to display results is "▲".

A "?" within a program will cause execution to stop temporarily and a "?" to appear on the display as the unit waits for data input. This command cannot be used independently, and is used together with  as a "[SHIFT]  memory name". To store a numeric value in memory A, for example:

7 → A

When "?" is displayed, calculation commands and numeric values can be input within 111 steps.

The "**▲**" command causes program execution to stop temporarily and the latest formula result or alphanumeric characters and symbols (see page 129) to be displayed. This command is used to mark positions in formulas where results are to be displayed. Since programs are ended and their final results displayed automatically, this command can be omitted at the end of a program. However, if the Base-n mode is specified for base conversion during a program, do not omit the final "**▲**".

Here these two commands will be used in the previously presented procedure:

SHIFT 7 - ALPHA A 2 X . 3 X ALPHA A x² SHIFT 4
 Input to memory A Display S

2 + 3 X ALPHA A x² 3
 omitted

Now the program is complete.

③ Program storage

The storage of programs is performed in the WRT mode which is specified by pressing **[MODE] [2]**.

Operation

MODE 2

Display

```

sys mode : WRT
cal mode : COMP
  angle : Deg
display : Norm

  422 Bytes Free

Prog 0:123456789

```

When **MODE** **2** are pressed, the system mode changes to the WRT mode. Then, the number of remaining steps (see page 106) is indicated. The number of remaining steps is decreased when programs are input or when memories are expanded. If no programs have been input and the number of memories equals 26 (the number of memories at initialization), the number of usable steps should equal 422.

The larger figures located below indicate the program areas (see page 108). If the letter "P" is followed by the numbers 0 through 9, it indicates that there are no programs stored in areas P0 through P9. The blinking zero here indicates the current program area is P0.

Areas into which programs have already been stored are indicated by "-" instead of numbers.

```

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

248 Bytes Free

Prog 0-1_34_6789
  
```

Here the previously mentioned program will be stored to program area P0 (indicated by the blinking zero):

Operation	Display
EXE (Start storage)	—
SHIFT 2 → ALPHA A 1 2 × √ 3	$7 \rightarrow A : 2 \times \sqrt{3} \times A^2$
× ALPHA A x² SHIFT 4	—
√ 2 ÷ 3 × ALPHA A x² 3	$7 \rightarrow A : 2 \times \sqrt{3} \times A^2$ $\sqrt{2} \div 3 \times A \times^2 3$

After these operations are complete, the program is stored.

* The system display appears only while the **MODE** key is pressed.

MODE (Displayed while pressed)

```

**** MODE ****

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

Step P0-20
  
```

* After the program is stored, press **MODE** **1** to return to the RUN mode.

④ Program execution

Programs are executed in the RUN mode (**MODE** **1**). The program area to be executed is specified using the **Prog** key.

To execute P0: **Prog** **0** **EXE**

To execute P3: **Prog** **3** **EXE**

To execute P8: **Prog** **8** **EXE**

Here the sample program that has been stored will be executed. The surface (S) and volume (V) for the regular octahedron in the sample problem are computed as:

Length of one side (A)	Surface area (S)	Volume (V)
10cm	(346.4101615)cm ²	(471.4045208)cm ³
7	(169.7409791)	(161.6917506)
15	(779.4228634)	(1590.990258)

Operation

MODE **1**

Display

```

**** MODE ****

sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0
  
```

Prog **0** **EXE**

```

7→A: 2X√3XA²
√2÷3XA x²3
Prog 0
7
  
```

10 **EXE**

(Value of A)

```

7→A: 2X√3XA²
√2÷3XA x²3
Prog 0
7
10

346.4101615
- Disp -
  
```

(S when A = 10)

Indicates answer displayed by **▲**

EXE

```

7 → A : 2 × √3 × A²
√2 ÷ 3 × A × 3
Prog 0
?
10
346.4101615
471.4045208
    
```

(V when A = 10)

Prog 0 EXE

```

√2 ÷ 3 × A × 3
Prog 0
?
10
346.4101615
471.4045208
Prog 0
?
    
```

7 EXE (Value of A)

```

10
346.4101615
471.4045208
Prog 0
?
7
169.7409791
- Disp -
    
```

(S when A = 7)

EXE

```

10
346.4101615
471.4045208
Prog 0
?
7
169.7409791
161.6917506
    
```

(V when A = 7)

Prog 0 EXE

```

471.4045208
Prog 0
?
7
169.7409791
161.6917506
Prog 0
?
    
```

15 EXE

(Value of A)

```

7
169.7409791
161.6917506
Prog 0
?
15
779.4228634
- Disp -
    
```

(S when A = 15)

EXE

```

7
169.7409791
161.6917506
Prog 0
?
15
779.4228634
1590.990258
    
```

(V when A = 15)

* Program computations are performed automatically with each press of EXE when it is pressed after data is input or after the result is read.

* Directly after a program in P0 is executed by pressing **Prog** 0 **EXE** as in this example, the Prog 0 command is stored by the replay function. Therefore, subsequent executions of the same program can be performed by simply pressing **EXE**.

Operation

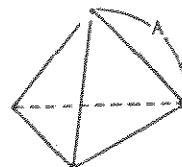
Prog 0 **EXE** (P0 program execution)
 10 **EXE** (Input 10 for A)
EXE (Display V when A = 10)
EXE (Reexecute)
 7 **EXE** (Input 7 for A)
EXE (Display V when A = 7)
 :

4-2 PROGRAM CHECKING AND EDITING (CORRECTION, ADDITION, DELETION)

Recalling a stored program can be performed in order to verify its contents. After specifying the desired program area using **⇐** or **⇒** in the WRT mode (**MODE** 2), the program contents will be displayed by pressing the **EXE** key. Once the program is displayed, the **⇐** (or **⇐**, **↑**, **↓**) key is used to advance the program one step at a time for verification. When the program has been improperly stored, editing can also be performed by adding to it or erasing portions. Here a new program will be created by checking and editing the previous sample program (the surface area and volume of a regular octahedron).

EXAMPLE:

Find the surface area and volume of a regular tetrahedron when the length of one side is given.



Length of one side (A)	Surface area (S)	Volume (V)
10 cm	() cm ²	() cm ³
7.5	()	()
20	()	()

① Formulas

For a surface area S, volume V and one side A, S and V for a regular tetrahedron are defined as:

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

② Programming

As with the previous example, the length of one side is stored in memory A and the program then constructed.

Numeric value A → **ALPHA** A **EXE**

√ 3 **×** **ALPHA** A **x²** **EXE** S

√ 2 **÷** 12 **×** **ALPHA** A **x³** 3 **EXE** V

When the above is formed into a program, it appears as follows:

SHIFT **?** **→** **ALPHA** A **:** **√** 3 **×** **ALPHA** A **x²** **SHIFT** **↵**

√ 2 **÷** 12 **×** **ALPHA** A **x³** 3

Program editing

But, a comparison of the two programs would be helpful.

Octahedron: $\text{[SHIFT] } \sqrt{\text{ }} \text{[2] } \text{[X]} \text{[3] } \text{[X]} \text{[ALPHA] } A \text{[X]} \text{[SHIFT] } \text{[]}$
 $\text{[] } \text{[2] } \text{[+]} \text{[3] } \text{[X]} \text{[ALPHA] } A \text{[X]} \text{[] } 3$

Tetrahedron: $\text{[SHIFT] } \sqrt{\text{ }} \text{[2] } \text{[X]} \text{[12] } \text{[X]} \text{[ALPHA] } A \text{[X]} \text{[SHIFT] } \text{[]}$
 $\text{[] } \text{[2] } \text{[+]} \text{[12] } \text{[X]} \text{[ALPHA] } A \text{[X]} \text{[] } 3$

The octahedron program can be changed to a tetrahedron program by editing the parts marked with wavy lines, and changing those that are marked with straight lines.

In actual practice, this would be performed as follows:

Operation

Display

$\text{[MODE] } \text{[2]}$

```
sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

402 Bytes Free

Prog _123456789
```

[EXE]

```
?→A: 2×√3×A²
√2÷3×A x³
```

Cursor located at beginning. Press $\text{[SHIFT] } \text{[EXE]}$ to bring cursor to end.

$\text{[] } \text{[] } \text{[] } \text{[] } \text{[DEL] } \text{[DEL]}$

```
?→A: √3×A²
√2÷3×A x³
```

Locate cursor at position to be deleted, and delete two characters.

$\text{[] } \text{[] } \text{[SHIFT] } \text{[INS] } 12$

```
?→A: √3×A²
√2÷123×A x³
```

Insert two characters.

[DEL]

```
?→A: √3×A²
√2÷12×A x³
```

Delete unnecessary 3.

$\text{[MODE] } \text{[1]}$

```
**** MODE ****

sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0
```

Editing complete. Return to the RUN mode.

④ Program execution

Now this program will be executed.

Length of one side (A)	Surface area (S)	Volume (V)
10 cm	(173.2050808)cm²	(117.8511302)cm³
7.5	(97.42785793)	(49.71844555)
20	(692.820323)	(942.8090416)

Operation

$\text{[MODE] } \text{[1]}$

Display

```
**** MODE ****

sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0
```

$\text{[Prog] } 0 \text{ [EXE]}$

```
?→A: √3×A²
√2÷12×A x³
Prog 0
?
```

10 [EXE]

```
?→A: √3×A²
√2÷12×A x³
Prog 0
?
10
173.2050808
- Disp -
```

[EXE]

```
?→A: √3×A²
√2÷12×A x³
Prog 0
?
10
173.2050808
117.8511302
```

Prog 0 EXE

$\sqrt{2} \div 12 \times A x^3$
Prog 0
?
10
173.2050808
117.8511302
Prog 0
?

7.5 EXE

10
173.2050808
117.8511302
Prog 0
?
7.5
97.42785793
- Disp -

EXE

10
173.2050808
117.8511302
Prog 0
?
7.5
97.42785793
49.71844555

Prog 0 EXE

117.8511302
Prog 0
?
7.5
97.42785793
49.71844555
Prog 0
?

20 EXE

EXE

7.5
97.42785793
49.71844555
Prog 0
?
20
692.820323
- Disp -

7.5
97.42785793
49.71844555
Prog 0
?
20
692.820323
942.8090416

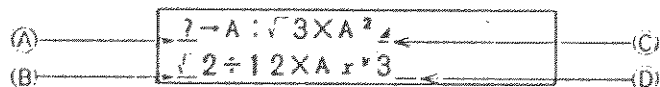
<Summary>

	Operation	Keys used
Program check	<ul style="list-style-type: none"> WRT mode specification Program area specification (Omitted if P0) Start verification Verification of contents 	MODE [2] [C] [D] EXE [C] [D] [Y] [J]
Correction	<ul style="list-style-type: none"> Move the cursor to the position to be corrected. Press correct keys. 	[C] [D] [Y] [J]
Deletion	<ul style="list-style-type: none"> Move the cursor to the position to be deleted. Delete 	[C] [D] [Y] [J] DEL
Insertion	<ul style="list-style-type: none"> Move the cursor to the position to be inserted into. Specify the insert mode. Press desired keys. 	[C] [D] [Y] [J] SHIFT INS

<Reference>

Cursor movement

Pressing the cursor keys ([C], [D], [Y], [J]) causes the cursor to move as follows:



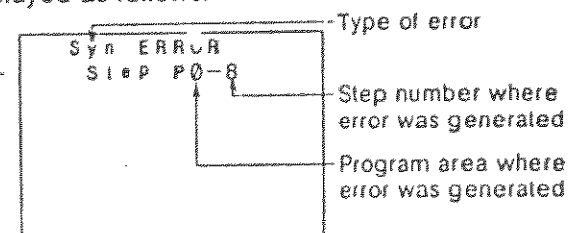
Cursor position	[C]	[D]	[Y]	[J]
(A)	Invalid	1 position right	Invalid	1 line down (B)
(B)	1 position left (C)	1 position right	1 line up (A)	End of line (D)
(C)	1 position left	1 position right (B)	Beginning of line (A)	1 line down (D)
(D)	1 position left	Invalid	1 line up (C)	Invalid

4-3 PROGRAM DEBUGGING (CORRECTING ERRORS)

After a program has been created and input, it will sometimes generate error messages when it is executed, or it will produce unexpected results. This indicates that there is an error somewhere within the program that needs to be corrected. Such programming errors are referred to as "bugs", while correcting them is called "debugging".

■ Debugging when an error message is generated

An error message is displayed as follows:



The error message informs the operator of the program area (P0 to P9) in which the error was generated. It also states the type of error, which gives an idea of the proper countermeasure to be taken. The step number indicates in which step of the program area the error was generated.

■ Error messages

There are a total of seven error messages.

- (1) Syn ERROR (Syntax error)
Indicates a mistake in the formula or a misuse of program commands.
- (2) Ma ERROR (Mathematical error)
Indicates the computation result of a numeric expression exceeds 10^{100} , an illogical operation (i.e. division by zero), or the input of an argument that exceeds the input range of the function.
- (3) Go ERROR (Jump error)
Indicates a missing Lbl for the Goto command (see page 113), or that the program area (see page 108) for the Prog command (see page 120) does not contain a program.

④ Ne ERROR (Nesting error)

Indicates a subroutine nesting overflow by the Prog command.

⑤ Stk ERROR (Stack error)

Indicates the computation performed exceeds the capacity of the stack for numeric values or for commands (see page 16).

⑥ Mem ERROR (Memory error)

Indicates the attempt to use a memory name such as Z [5] without having expanded memories.

⑦ Arg ERROR (Argument error)

Indicates the argument of a command or specification in a program exceeds the input range (i.e. Sci 10, Goto 11).

Further operation will become impossible when an error message is displayed. Press **AC**, **↵**, or **↵** to cancel the error.

Pressing **AC** cancels the error and new key input becomes possible.

With this operation, the RUN mode is maintained.

Pressing **↵** or **↵** cancels the error and changes the system mode to the WRT mode. The cursor is positioned at the location where the error was generated to allow modification of the program to eliminate the error.

■ Checkpoints for each type of error

The following are checkpoints for each type of error:

① Syn ERROR

Verify again that there are no errors in the program.

② Ma ERROR

For computations that require use of the memories, check to see that the numeric values in the memories do not exceed the range of the arguments. This type of error often occurs with division by 0 or the computation of negative square roots.

③ Go ERROR

Check to see that there is a corresponding Lbl n when Goto n is used. Also check to see that the program in P_n has been correctly input when Prog n is used.

④ Ne ERROR

Check to ensure that the Prog command is not used in the branched program area to return execution to the original program area.

⑤ Stk ERROR

Check to see that the formula is not too long thus causing a stack overflow. If this is the case, the formula should be divided into two or more parts.

⑥ Mem ERROR

Check to see that memories were properly expanded using "**MODE** **□** n **EXE**" (Defm). When using array-type memories (see page 124), check to see that the subscripts are correct.

⑦ Arg ERROR

Check whether values specified by **MODE** **7** (Sci) or **MODE** **8** (Fix) are within the range of 0 ~ 9. Also check whether values specified by Goto, Lbl, or Prog commands are within 0—9. Also ensure that memory expansion using **MODE** **□** (Defm) is performed within the remaining number of steps and that the value used for expansion is not negative.

4-4 COUNTING THE NUMBER OF STEPS

The program capacity of this unit consists of a total of 422 steps. The number of steps indicates the amount of storage space available for programs, and it will decrease as programs are input. The number of remaining steps will also be decreased when steps are converted to memories. (See page 24).

There are two methods to determine the current number of remaining steps:

- ① When **MODE** \square **EXE** are pressed in the RUN mode, the number of remaining steps will be displayed together with the number of memories.

Example:

MODE \square **EXE**

```

**Delfm**
Program : 19
Memory : 26
403 Bytes Free
  
```

Number of steps used for programming
Number of memories
Number of remaining steps

- ② Specify the WRT mode (**MODE** \square **2**), and the number of remaining steps will appear. At this time the status of the program areas can also be determined.

MODE \square **2**

```

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

403 Bytes Free
Prog 123456789
  
```

Number of remaining steps

Basically, one function requires a single step, but there are some commands where one function requires two steps.

- One function/one step: sin, cos, tan, log, (,), :, A, B, 1, 2, 3, etc.
- One function/two steps: Lbl 1, Goto 2, Prog 8, etc.

Each step can be verified by the movement of the cursor:

Example:

Present cursor position → $7 \rightarrow A : \sqrt{3} \times A^2$
 $\sqrt{2} \div 12 \times A \div 3$

At this time, each press of a cursor key (\leftarrow or \rightarrow) will cause the cursor to move to the next sequential step. For example:

6th step
 $7 \rightarrow A : \sqrt{3} \times A^2$
 $\sqrt{2} \div 12 \times A \div 3$

The display will show at what step of the program the cursor is currently located as long as **MODE** is pressed.

MODE \sim
(Hold down)

```

**** MODE ****
sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm
Step P0-6
  
```

Indicates cursor is located at 6th step.

4-5 PROGRAM AREAS AND COMPUTATION MODES

This unit contains a total of 10 program areas (P0 through P9) for the storage of programs. These program areas are all utilized in the same manner, and 10 independent programs can be input. One main program (main routine) and a number of secondary programs (subroutines) can also be stored. The total number of steps available for storage in program areas P0 through P9 is 422 maximum.

Specification of a program area is performed as follows:

RUN mode: Press any key from 0 through 9 after pressing the [Prog] key.

Then press [EXE].

Example: P0 [Prog] 0 [EXE]
P8 [Prog] 8 [EXE]

* In this mode, program execution begins when [EXE] is pressed.

WRT mode: Use [←] or [→] to move the cursor under the program area to be specified and press [EXE].

Only the numbers of the program areas that do not yet contain programs will be displayed. "—" symbols indicate program areas which already contain programs.

Example:

```

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

403 Bytes Free

Prog 123 67 9
  
```

Programs already stored in these program areas.

Program area and computation mode specification in the WRT mode

Besides normal function computations, to perform binary, octal, decimal and hexadecimal computations and conversions, standard deviation computations, and regression computations in a program, a computation mode must be specified. Program mode specification and program area specification are performed at the same time.

First the WRT mode is specified ([MODE] [2]), and then a computation mode is specified. Next, the program area is specified, and, when [EXE] is pressed, the computation mode is memorized in the program area. Henceforth, stored programs will be accompanied with the computation mode.

Example: Memorizing the Base-n mode in P2.

[MODE] [2]

```

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

422 Bytes Free

Prog 0 1 2 3 4 5 6 7 8 9
  
```

Assuming that nothing is stored.

[←] [→]

```

sys mode : WRT
cal mode : COMP
angle : Deg
display : Norm

422 Bytes Free

Prog 0 1 2 3 4 5 6 7 8 9
  
```

Specify P2.

[MODE] [—]

```

sys mode : WRT
cal mode : Base-n
          Dec

422 Bytes Free

Prog 0 1 2 3 4 5 6 7 8 9
  
```

Specify the Base-n mode.

[EXE]

```

  
```

As shown above, the computation mode will be memorized into a program area.

■ Cautions concerning the computation modes

All key operations available in each computation mode can be stored as programs, but, depending on the computation mode, certain commands or functions cannot be used.

Base-n mode

- Function computations cannot be performed.
- Units of angular measurement cannot be specified.
- All program commands can be used.
- Be sure to include a "▲" at the final result output to return to the previous computation mode when a program execution is terminated. Failure to do so may result in a decimal display or an error.

SD1, SD2 mode

- Among the functions, Abs and $\sqrt{\quad}$ cannot be used.
- Among the program commands, Dsz, > and < cannot be used.

LR1, LR2 mode

- Among the functions, Abs and $\sqrt{\quad}$ cannot be used.
- Among the program commands, \Rightarrow , =, \neq , \approx , \cong , Dsz, \approx , \approx , Dsz, > and < cannot be used.

4-6 ERASING PROGRAMS

Erasing of programs is performed in the PCL mode. Press **MODE** **3** to specify the PCL mode. There are two methods used to erase programs: erasing a program located in a single program area, and erasing all programs.

■ Erasing a single program

To erase a program in a single program area, specify the PCL mode and press the **AC** key after specifying the program area.

Example: Erase the program in P3 only.

Operation

MODE **3**

Display

```
sys mode : PCL
cal mode : COMP
angle : Deg
display : Norm

324 Bytes Free
Prog 12 45678
```

P0, P3 and P9 already contain programs.

← **→** **←**

```
sys mode : PCL
cal mode : COMP
angle : Deg
display : Norm

324 Bytes Free
Prog 12 45678
```

Align cursor with P3.

AC

```
sys mode : PCL
cal mode : COMP
angle : Deg
display : Norm

367 Bytes Free
Prog 12 345678
```

Number 3 appears after deletion.

MODE [1]

```

**** MODE ****
sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0

```

Return to RUN mode.

Erasing all programs

To erase all programs stored in program areas 0 through 9, specify the PCL mode and press **SHIFT** and then **DEL**.

Example: Erase the programs stored in P0, P4, P8 and P9.

Operation

MODE [3]

Display

```

sys mode : PCL
cal mode : COMP
angle : Deg
display : Norm

295 Bytes Free

Prog 123 567

```

SHIFT DEL

```

sys mode : PCL
cal mode : COMP
angle : Deg
display : Norm

422 Bytes Free

Prog 0123456789

```

MODE [0]

```

**** MODE ****
sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0

```

4-7 CONVENIENT PROGRAM COMMANDS

The programs for this unit are made based upon manual computations. Special program commands, however, are available to allow the selection of the formula and repetitive execution of the same formula. Here, some of these commands will be used to produce more convenient programs.

Jump commands

Jump commands are used to change the flow of program execution. Programs are executed in the order that they are input (from the lowest step number first) until the end of the program is reached. This system is not very convenient when there are repeat computations to be performed or when it is desirable to transfer execution to another formula. It is in these cases, however, that the jumps commands are very effective. There are three types of jump commands: a simple unconditional jump to a branch destination, conditional jumps that decide the branch destination by whether a certain condition is true or not, and count jumps that increase or decrease a specific memory by one and then decide the branch destination after checking whether the value stored equals zero or not.

Unconditional Jump

The unconditional jump is composed of "Goto" and "Lbl". When program execution reaches the statement "Goto n" (where n is a number from 0 through 9), execution then jumps to "Lbl n" (n is the same value as Goto n). The unconditional jump is often used in simple programs to return execution to the beginning for repetitive computations, or to repeat computations from a point within a program.

Unconditional jumps are also used in combination with conditional and count jumps.

Example: The previously presented program to find the surface area and volume of a regular tetrahedron will be rewritten using "Goto 1" and "Lbl 1" to allow repeat computations.

The previous program contained:

$\sqrt{3} \cdot A \cdot \sqrt{3} \cdot 3 \cdot X \cdot A \cdot x^2 \cdot \Delta$
 $\sqrt{3} \cdot 2 \cdot \div \cdot 1 \cdot 2 \cdot X \cdot A \cdot x^3 \cdot 3$

19 steps

* Hereinafter, commas (,) will be used to separate steps for the sake of clarity.

Add "Goto 1" to the end of the program, and add "Lbl 1" to the beginning of the program as the branch destination.

If this is simply left the way it is, however, the volume will not be displayed and execution will move immediately to the input of one side at the beginning. To prevent this situation, insert a display command (▲) in front of the "Goto 1".

The complete program with the unconditional jump added should look like this:

```
Lbl, 1, :, ?, →, A, :, √, 3, X, A, x², ▲
```

```
√, 2, ÷, 1, 2, X, A, x², 3, ▲, Goto, 1
```

25 steps

Now let's try executing this program.

For details on inputting programs and editing programs, see sections 4-1 and 4-2.

Henceforth, the displays will only show computation result output.

Operation

Display

[Prog] [0] [EXE]

?

Stored in P0.

10 [EXE]

173.2050808

The length of the side=10

[EXE]

117.8511302

[EXE]

?

7.5 [EXE]

97.42785793

The length of the side=7.5

[EXE]

49.71844555

[EXE]

?

Since the program is in an endless loop, it will continue execution. To terminate execution, press [MODE] [1].

[MODE] [1]

```
**** MODE ****
sys mode : RUN
cal mode : COMP
angle : Deg
display : Norm

Step 0
```

Besides the beginning of the program, branch destinations can be designated at any point within the program.

Example: Compute $y=ax+b$ when the value for x changes each time, while a and b can also change depending upon the computation.

Program

```
?, →, A, :, ?, →, B, :, Lbl, 1, :, ?, →, X, :
```

```
A, X, X, +, B, ▲, Goto, 1
```

23 steps

When this program is executed, the values for a and b are stored in memories A and B respectively. After that, only the value for x can be changed.

In this way an unconditional jump is made in accordance with "Goto" and "Lbl", and the flow of program execution is changed. When there is no "Lbl" to correspond to a "Goto", an error (Go ERROR) is generated.

Conditional jumps

The conditional jumps compare a numeric value in memory with a constant or a numeric value in another memory. If the condition is true, the statement following the " \Rightarrow " is executed, and if the condition is not true, execution skips the statement and continues following the next " \leftarrow ", " $:$ " or " \blacktriangle ".

Conditional jumps take on the following form:

Left side	Relational operator	Right side	\Rightarrow	Statement	$\left\{ \begin{array}{l} \leftarrow \\ : \\ \blacktriangle \end{array} \right\}^*$	Statement
-----------	---------------------	------------	---------------	-----------	---	-----------

* \leftarrow represents carriage return function (see page 122).

* Anyone can be used.

One memory name (alphabetic character from A through Z), constant numeric values or computation formulas (AX^2 , $D-E$, etc.) are used for "left side" and "right side".

The relational operator is a comparison symbol. There are 6 types of relational operators: $=$, \neq , \geq , \leq , $>$, $<$.

Left side $=$ right side (left side equals right side)

Left side \neq right side (left side does not equal right side)

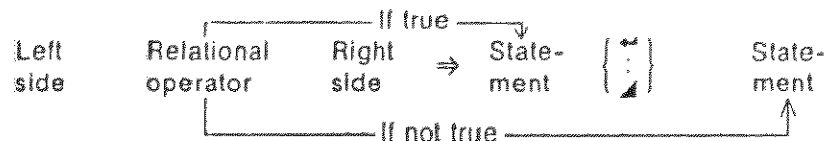
Left side \geq right side (left side is greater than or equal to right side)

Left side \leq right side (left side is less than or equal to right side)

Left side $>$ right side (left side is greater than right side)

Left side $<$ right side (left side is less than right side)

The "⇒" is displayed when [Shift] [⇒] are pressed. If the condition is true, execution advances to the statement following ⇒. If the condition is not true, the statement following ⇒ is skipped and execution jumps to the statement following the next "←", ":" or "▲".



A statement is a computation formula (sin AX5, etc.) or a program command (Goto, Prog, etc.), and everything up to the next "←", ":" or "▲" is regarded as one statement.

Example: If an input numeric value is greater than or equal to zero, compute the square root of that value. If the input value is less than zero, reinput another value.

Program

Lbl, 1, :, ?, →, A, :, A, ≥, 0, ⇒, √, A, ▲, Goto, 1

16 steps

In this program, the input numeric value is stored in memory A, and then it is tested to determine whether it is greater than, equal to or less than zero. If the contents of memory A are greater than or equal to 0 (not less than zero), the statement (computation formula) located between "⇒" and "▲" will be executed, and then Goto 1 returns execution to Lbl 1. If the contents of memory A are less than zero, execution will skip the following statement to the next "▲" and returned to Lbl 1 by Goto 1.

Example: Compute the sum of input numeric values. If a 0 is input, the total should be displayed.

Program

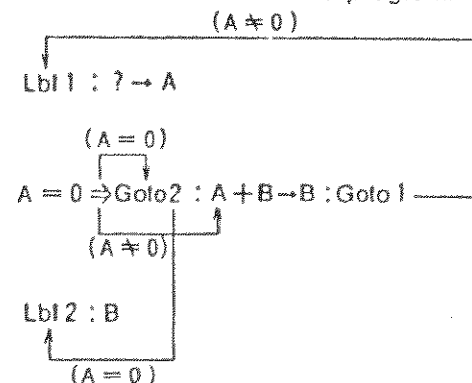
0, →, B, :,
 Lbl, 1, :, ?, →, A, :, A, =, 0, ⇒, Goto, 2, :,
 A, +, B, →, B, :, Goto, 1, :,
 Lbl, 2, :, B

31 steps

In this program, a 0 is first stored in memory B to clear it for computation of the sum. Next, the value input by "?→A" is stored in memory A by "A=0⇒" and it is determined whether or not the value stored in memory A equals zero. If A=0, Goto 2 causes execution to jump to Lbl 2. If memory A does not equal 0, Goto 2 will be skipped and the command A+B→B which follows ":" is executed, and then Goto 1 returns execution to Lbl 1.

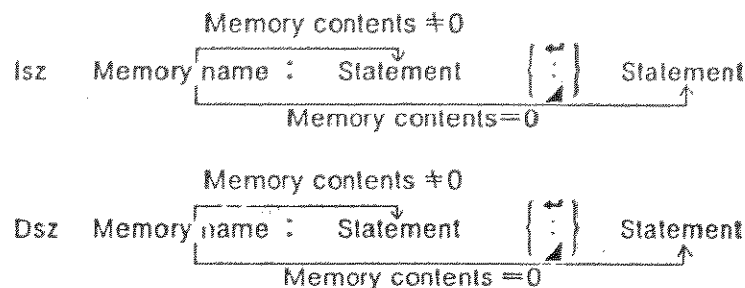
Execution from Lbl 2 will display the sum that has been stored in memory B. Actually, the display command "▲" is inserted following B, but

The following illustration shows the flow of the program:



Count jumps

The count jumps cause the value in a specified memory to be increased or decreased by 1. If the value does equal 0, the following statement is skipped, and the statement following the next "←", ":" or "▲" is executed. The "Isz" command is used to increase the value in memory by 1 and decide the subsequent execution, while the "Dsz" command is used to decrease the value by 1 and decide.



Example: Increase memory A by one Isz A
 Decrease memory B by one Dsz B

Example: Determine the average of 10 input numeric values:

Program

1, 0, →, A, :, 0, →, C, :,
 Lbl, 1, :, ?, →, B, :, B, +, C, →, C, :,
 Dsz, A, :, Goto, 1, :, C, ÷, 1, 0

32 steps

In this program, first 10 is stored in memory A, and 0 is stored in memory C. Memory A is used as the "counter" and countdown is performed the specified number of times by the Dsz command. Memory C is used to store the sum of the inputs, and so first must be cleared by inputting a 0. The numeric value input in response to "?" is stored in memory B, and then the sum of the input values is stored in memory C by "B+C→C". The statement Dsz A then decreases the value stored in memory A by 1. If the result does not equal 0, the following statement, Goto 1 is executed. If the result equals 0, the following Goto 1 is skipped and "C÷10" is executed.

Example: Determine the altitude at one-second intervals of a ball thrown into the air at an initial velocity of V_m /sec and an angle of S° . The formula is expressed as: $h = V \sin \theta t - \frac{1}{2}gt^2$, with $g = 9.8$, with the effects of air resistance being disregarded.

Program

```
Deg, :, 0, →, T, :, ?, →, V, :, ?, →, S, :,  
Lbl, 1, :, Isz, T, :, V, X, sin, S, X, T, -,  
9, -, 8, X, T, x², ÷, 2, ▴, Goto, 1
```

38 steps

In this program the unit of angular measurement is set and memory T is first initialized (cleared). Then the initial velocity and angle are input into memories V and S respectively.

Line 1 is used at the beginning of the repeat computations. The numeric value stored in memory T is counted up (increased by 1) by Isz T. In this case, the Isz command is used only for the purpose of increasing the value stored in memory T, and the subsequent jump does not depend upon any comparison or decision. The Isz command can also be used in the same manner as seen with the Dsz command for jumps that require decisions, but, as can be seen here, it can also be used to simply increase values. If, in place of the Isz command, another method such as "T+1→T" is used, five steps are required instead of the two for the (Isz T) method shown here. Such commands are convenient ways of conserving memory space.

Each time memory T is increased, computation is performed according to the formula, and the altitude is displayed. It should be noted that this program is endless, so when the required value is obtained, [MODE] [1] are pressed to terminate the program.

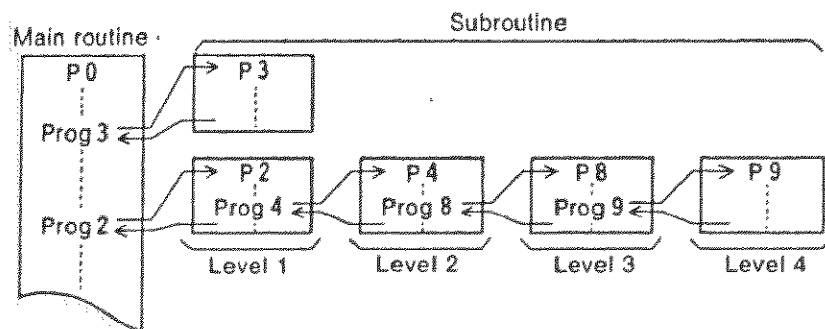
<Summary>

Command	Formula	Operation
Unconditional jump	Lbl n Goto n (n=natural number from 0 through 9)	Performs unconditional jump to Lbl n corresponding to Goto n.
Conditional jumps	Left side Relational operator Right side ⇒ Statement { : } Statement (Relational operators: =, ≠, >, <, ≥, ≤)	Left and right sides are compared. If the conditional expression is true, the statement after ⇒ is executed. If not true, execution jumps to the statement following the next ⇐, : or ▴. Statements include numeric expressions, Goto commands, etc.
Count jumps	Isz Memory name: Statement { : } Statement Dsz Memory name: Statement { : } Statement (Memory name consists of single character from A through Z, A [], etc.)	Numeric value stored in memory is increased (Isz) or decreased (Dsz) by one. If result equals 0, a jump is performed to the statement following the next ⇐, : or ▴. Statements include numeric expressions, Goto commands, etc.

■ Subroutines

A program contained in a single program area is called a "main routine". Often used program segments stored in other program areas are called "subroutines".

Subroutines can be used in a variety of ways to help make computations easier. They can be used to store formulas for repeat computations as one block to be jumped to each time, or to store often used formulas or operations for call up as required.



The subroutine command is "Prog" followed by a number from 0 through 9 which indicates the program area.

Example: Prog 0Jump to program area 0

Prog 2Jump to program area 2

After the jump is performed using the Prog command, execution continues from the beginning of the program stored in the specified program area. After execution reaches the end of the subroutine, the program returns to the statement following the Prog *n* command in the original program area. Jumps can be performed from one subroutine to another, and this procedure is known as "nesting". Nesting can be performed to a maximum of 10 levels, and attempts to exceed this limit will cause an error (Ne ERROR) to be generated. Attempting to use Prog to jump to a program area in which there is no program stored will also result in an error (Go ERROR).

* A Goto *n* contained in a subroutine will jump to the corresponding Lbl *n* contained in that program area.

Example: Simultaneously execute the two previously presented programs to compute the surface areas and volumes of a regular octahedron and tetrahedron.

Express the result in three decimal places.

This example employs two previously explained programs, and the first step is to input the specified number of decimal places (MODE 7 3).

Now let's review the two original programs.

Regular octahedron

P0 Fix, 3, :, 7, →, A, :, 2, X, √, 3, X, A, x², ▲,
 $\sqrt{, 2, \div, 3, X, A, x^3, 3}$ 23 steps

Regular tetrahedron

P1 Fix, 3, :, 7, →, A, :, √, 3, X, A, x², ▲,
 $\sqrt{, 2, \div, 1, 2, X, A, x^3, 3}$ 22 steps
 Total: 45 steps

If the two programs are compared, it is evident that the underlined portions are identical. If these portions are incorporated into a common subroutine, the programs are simplified and the number of steps required is decreased.

Furthermore, the portions indicated by the wavy line are not identical as they stand, but if P1 is modified to: $\sqrt{, 2, \div, 3, X, A, x^3, 3, \div, 4}$, the two portions become identical.

Now the portions underlined by the straight line will be stored as an independent routine in P9 and those underlined with the wavy line will be stored in P8.

P9 Fix, 3, :, 7, →, A, :, √, 3, X, A, x² 12 steps
 P8 $\sqrt{, 2, \div, 3, X, A, x^3, 3}$ 8 steps

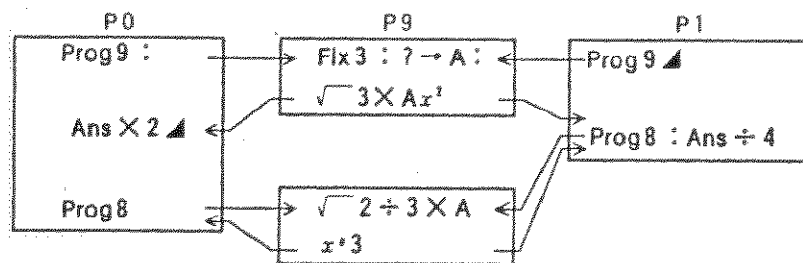
After the common segments have been removed, the remainder of the regular octahedron formula is stored in P0, and that of the regular tetrahedron is stored in P1. Of course, the "Prog 9" and "Prog 8" must be added to jump to subroutines P9 and P8.

P0 Prog, 9, :, Ans, X, 2, ▲, Prog, 8 9 steps
 P1 Prog, 9, ▲, Prog, 8, :, Ans, ÷, 4 9 steps
 Total: 38 steps

With this configuration, execution jumps to program P9 at the beginning of programs P0 and P1, three decimal places are specified, the value for one side is entered, and the surface area of the tetrahedron is computed. The expression "2X" of the original octahedron formula was omitted in P9, so when execution returns to P0, "AnsX2" is used to obtain the surface of the octahedron. In the case of P1, the result of P9 needs no further modification and so is immediately displayed upon return to P1.

Computation of the volumes is also performed in a similar manner. After a jump is made to P8 for computation, execution returns to the main routines. In P0, the program ends after the volume of the octahedron is displayed. In P1, however, the result computed in P8 is divided by four to obtain the volume of the tetrahedron. By using subroutines in this manner, steps can be shortened and programs become neat and easy to read.

The following illustration shows the flow of the program just presented.



By isolating the common portions of the two original programs and storing them in separate program areas, steps are shortened and programs take on a clear configuration.

Carriage return function

With the carriage return function, [EXE] is used in place of [] to separate commands to produce easy-to-read displays.

```

Deg : 0 -> T : 7 -> V : 7 -> S :
LbI 1 : 1 sz T : V X s I
n S X T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

Using the carriage return function in the program shown above produces the following display:

```

Deg V
0 -> T : 7 -> V : 7 -> S
LbI 1 : 1 sz T : V X s I
n S X T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

[EXE] pressed at these two locations. Nothing is displayed at the point where [EXE] is pressed, and the display advances to the next line.

This makes angle unit setting and looped operations, etc. easier to follow.

Operation procedure

[MODE] [4] [EXE] (Press in place of [])

[0] [] [ALPHA] [1] [] [SHIFT] [7] [] [ALPHA] [7] [] [SHIFT] [7] [] [ALPHA] [5] [EXE]

[SHIFT] [LbI] [1] []

- To include the carriage return function in a program that has already been input, first press [SHIFT] [INS] to specify the insert mode and then press [EXE]. Then, delete the " : ".

```

Deg : 0 -> T : 7 -> V : 7 -> S :
LbI 1 : 1 sz T : V X s I
n S X T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

Align the cursor with the " : " following "Deg" and press [SHIFT] [INS] [EXE].

[] [SHIFT] [INS] [EXE]

```

Deg
0 -> T : 7 -> V : 7 -> S : LbI
1 : 1 sz T : V X s I n S
X T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

Delete the " : ".

[DEL]

```

Deg
0 -> T : 7 -> V : 7 -> S : LbI
1 : 1 sz T : V X s I n S X
T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

Align the cursor with the " : " following "? -> S". As above, first Insert [EXE] and then delete the " : ".

[] [] [] [SHIFT] [INS]

[EXE] [DEL]

```

Deg
0 -> T : 7 -> V : 7 -> S
LbI 1 : 1 sz T : V X s I
n S X T - 9 . 8 X T ^ 2 ÷ 2
Goto 1
  
```

- Carriage return can be used in manual operations by pressing [SHIFT] [EXE].

4-8 ARRAY-TYPE MEMORIES

■ Using array-type memories

Up to this point all of the memories used have been referred to by single alphabetic characters such as A, B, X, or Y.

With the array-type memory introduced here, a memory name (one alphabetic character from A through Z) is appended with a subscript such as [1] or [2].

* Brackets are input by $\boxed{\text{ALPHA}} \boxed{[}$ and $\boxed{\text{ALPHA}} \boxed{\text{EXP}}$.

Standard memory	Array-type memory	
A	A[0]	C[-2]
B	A[1]	C[-1]
C	A[2]	C[0]
D	A[3]	C[1]
E	A[4]	C[2]

Proper utilization of subscripts shortens programs and makes them easier to use. Negative values used as subscripts are counted in relation to memory zero as shown above.

Example: Input the numbers 1 through 10 into memories A through J.

Using standard memories

1, →, A, ∴, 2, →, B, ∴, 3, →, C, ∴, 4, →, D, ∴,
5, →, E, ∴, 6, →, F, ∴, 7, →, G, ∴, 8, →, H, ∴,
9, →, I, ∴, 1, 0, →, J

40 steps

Using array-type memories

0, →, Z, ∴, Lbl, 1, ∴, Z, +, 1, →, A, [Z], ∴,
Isz, Z, ∴, Z, <, 1, 0, ⇒, Goto, 1

26 steps

In the case of using standard memories, inputting values into memories one by one is both inefficient and time consuming. What happens, if we want to see a value stored in a specific memory?

Using standard memories

Lbl, 1, ∴, 7, →, Z, ∴,
Z, =, 1, ⇒, A, ▲, Z, =, 2, ⇒, B, ▲,
Z, =, 3, ⇒, C, ▲, Z, =, 4, ⇒, D, ▲,
Z, =, 5, ⇒, E, ▲, Z, =, 6, ⇒, F, ▲,
Z, =, 7, ⇒, G, ▲, Z, =, 8, ⇒, H, ▲,
Z, =, 9, ⇒, I, ▲, Z, =, 1, 0, ⇒, J, ▲,
Goto, 1

70 steps

Using array-type memories

Lbl, 1, ∴, 7, →, Z, ∴, A, [Z, -, 1], ▲,
Goto, 1

16 steps

The difference is readily apparent. When using the standard memories, the input value is compared one by one with the value assigned to each memory (i.e. A=1, B=2, ...).

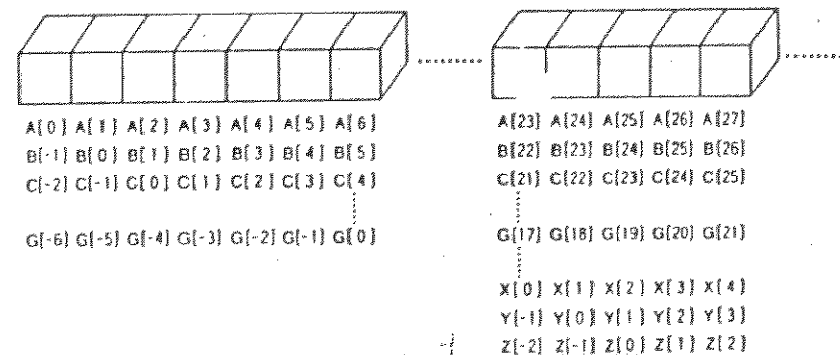
With the array-type memories, the input value is immediately stored in the proper memory determined by "[Z-1]". Formulas (Z-1, A+10, etc.) can even be used for the subscript.

■ Cautions when using array-type memories

When using array-type memories, a subscript is appended to an alphabetic character that represents a standard memory from A through Z.

Therefore, care must be taken to prevent overlap of memories.

The relation is as follows:



The following shows a case in which array-type memories overlap with standard format memories. This situation should always be avoided.

Example: Store the numeric values from 1 through 5 in memories A[1] through A[5] respectively.

```
5. →, C, :, Lbl, 1, :, C, →, A, [, C, ], :,
Dsz, C, :, Goto, 1, :,
A, [, 1, ], ▲, A, [, 2, ], ▲, A, [, 3, ], ▲,
A, [, 4, ], ▲, A, [, 5, ]
```

44 steps

In this program, the values 1 through 5 are stored in the array-type memories A[1] through A[5], and memory C is used as a counter memory. When this program is executed, the following results are obtained:

Operation

Display

[Prog] 0 [EXE]

[EXE]

[EXE]

[EXE]

[EXE]

1.
0.
3.
4.
5.

As can be seen, the second displayed value (which should be 2) in A[2] is incorrect. This problem has occurred because memory A[2] is the same as memory C.

A	B	C	D	E	F
A[1]	A[2]	A[3]	A[4]	A[5]	

The content of memory C (A[2]) is decreased from 5 to 0 in steps of 1. Therefore, the content of memory A[2] is displayed as 0.

Application of the array-type memories

It is sometimes required to treat two different types of data as a single group. In this case, memories for data processing and those for data storage should be kept separate.

Example: Store data x and y in memories. When an x value is input, the corresponding y value is displayed. There will be a total of 15 pieces of data.

Example program 1

Memory A is used as the data control memory, and memory B is used for temporary storage of the x data. The x data are stored in memories C[1] (memory D) through C[15] (memory R), and the y data are stored in memories C[16] (memory S) through C[30] (memory Z[7]).

```
1. →, A, :, Defm, 7, :,
Lbl, 1, :, ?, →, C, [, A, ], :,
?, →, C, [, A, +, 1, 5, ], :,
Isz, A, :, A, =, 1, 6, ⇒, Goto, 2, :, Goto, 1, :,
Lbl, 2, :, 1, 5, →, A, :, ?, →, B, :,
B, =, 0, ⇒, Goto, 5, :,
Lbl, 3, :, B, =, C, [, A, ], ⇒, Goto, 4, :,
Dsz, A, :, Goto, 3, :, Goto, 2, :,
Lbl, 4, :, C, [, A, +, 1, 5, ], ▲, Goto, 2, :,
Lbl, 5
```

98 steps

In this program, memories are used as follows:

x data

C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]
D	E	F	G	H	I	J	K
C[9]	C[10]	C[11]	C[12]	C[13]	C[14]	C[15]	
L	M	N	O	P	Q	R	

y data

C[16]	C[17]	C[18]	C[19]	C[20]	C[21]	C[22]	C[23]
S	T	U	V	W	X	Y	Z
C[24]	C[25]	C[26]	C[27]	C[28]	C[29]	C[30]	
Z(1)	Z(2)	Z(3)	Z(4)	Z(5)	Z(6)	Z(7)	

Example program 2

The same memories are used as in Example 1, but two types of memory names are used and the x and y data kept separate.

```
1, →, A, :, Defm, 7, :,
Lbl, 1, :, ?, →, C, [, A, ], :,
?, →, R, [, A, ], :,
Isz, A, :, A, =, 1, 6, ⇒, Goto, 2, :, Goto, 1, :,
Lbl, 2, :, 1, 5, →, A, :, ?, →, B, :,
B, =, 0, ⇒, Goto, 5, :,
Lbl, 3, :, B, =, C, [, A, ], ⇒, Goto, 4, :,
Dsz, A, :, Goto, 3, :, Goto, 2, :,
Lbl, 4, :, R, [, A, ], ▲, Goto, 2, :,
Lbl, 5
```

92 steps

Memories are used as follows:

x data

C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]
D	E	F	G	H	I	J	K
C[9]	C[10]	C[11]	C[12]	C[13]	C[14]	C[15]	
L	M	N	O	P	Q	R	

y data

R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]	R[8]
S	T	U	V	W	X	Y	Z
R[9]	R[10]	R[11]	R[12]	R[13]	R[14]	R[15]	
Z(1)	Z(2)	Z(3)	Z(4)	Z(5)	Z(6)	Z(7)	

In this way, the memory names can be changed. However, since memory names are restricted to the letters from A through Z, the expanded memories ([MODE] [□]) can only be used as array-type memories.

* The memory expansion command (Defm) can be used in a program.

Example: Expand the number of memories by 14 to make a total of 40 available.

```
Defm, 1, 4, :, .....
```

4-9 DISPLAYING ALPHA-NUMERIC CHARACTERS AND SYMBOLS

Alphabetic characters, numbers, computation command symbols, etc., can be displayed as messages. They are enclosed in quotation marks ([ALPHA] [PRG]).

■ Alpha-numeric characters and symbols

• Characters and symbols displayed when pressed following [ALPHA]:

[,], k, m, μ , n, p, l, space,
A, B, C, D, E, F, G, H, I, J, K, L, M, N,
O, P, Q, R, S, T, U, V, W, X, Y, Z

• Other numbers, symbols, calculation commands, program commands

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,

(,), $\sqrt{\quad}$, π , +, -, \times , \div , ...

sin, cos, tan, log, ln, ...

=, \neq , \geq , \leq , $>$, $<$, ...

A, B, C, D, E, F, d, h, b, o

Neg, Not, and, or, xor

\bar{x} , \bar{y} , $x\sigma_n$, $x\sigma_{n-1}$, ...

° ([SHIFT] [MODE] [4]), ° ([SHIFT] [MODE] [5]), ° ([SHIFT] [MODE] [6])

* All of the above noted characters can be used in the same manner as the alphabetic characters.

In the preceding example requiring an input of two types of data (x , y), the prompt "?" does not give any information concerning the type of input expected. A message can be inserted before the "?" to verify the type of data required for input.

```
Lbl, 1, :, ?, →, X, :, ?, →, Y, :, ...
```

The messages "X=" and "Y=" will be inserted into this program.

```
Lbl, 1, :, "X=", ?, →, X, :,
```

```
"Y=", ?, →, Y, :, ...
```

If messages are included as shown here, the display is as follows.
(Assuming that the program is stored in P1)

[Prog] 1 [EXE]

10 [EXE]

:

X = ?

Y = ?

:

Messages are also convenient when displaying result in program computations.

Example:

Lbl, 0, :, ", N, =, ", ?, →, B, ~, C, :,

0, →, A, :,

Lbl, 1, :, C, ÷, 2, →, C, :, Frac, C, ≠, 0, ⇒, Goto, 3,

: , Isz, A, :, C, =, 1, ⇒, Goto, 2, :, Goto, 1, :,

Lbl, 2, :, ", X, =, ", ▲, A, ▲, Goto, 0, :,

Lbl, 3, :, ", N, O, ", ▲, Goto, 0

70 steps

This program computes the x power of 2. A prompt of "N=?" appears for data input. The result is displayed by pressing [EXE] while "X=" is displayed. When an input data is not the x power of 2, the display "NO" appears and execution returns to the beginning for reinput.

* Always follow a message with a ▲ whenever a formula follows the message.

Assuming that the program is stored in P2:

[Prog] 2 [EXE]

4096 [EXE]

[EXE]

[EXE]

3124 [EXE]

[EXE]

512 [EXE]

[EXE]

N = ?

X =

12.

N = ?

NO

N = ?

X =

9.

Strings longer than 16 characters are displayed in two lines. When alphabetic characters are displayed at the end of the bottom line, the entire display shifts upwards and the uppermost line disappears from the display.

[Prog] 0

123+45

168.

852-87

765.

968+125-65

1028.

Prog 0

[EXE]

123+45

168.

852-87

765.

968+125-65

1028.

Prog 0

ABCDEFGHIJKLMN

| After a while

852-87

168.

968+125-65

765.

1028.

Prog 0

ABCDEFGHIJKLMN

QRSTUVWXYZ

4-10 USING THE GRAPH FUNCTION IN PROGRAMS

Using the graph function within programs makes it possible to graphically represent long, complex equations and to overwrite graphs repeatedly. All graph commands (except the trace function) can be included in programs. Range values can also be written into the program. Generally, manual graph operations can be used in programs without modification.

Ex. 1) Graphically determine the number of solutions (real roots) that satisfy both of the following two equations.

$$y = x^4 - x^3 - 24x^2 + 4x + 80$$

$$y = 10x - 30$$

The range values are as follows:

```
Range
Xmin:-10.
max:10.
sci:2.
Ymin:-120.
max:150.
sci:50.
```

First, program the range settings. Note that values are separated from each other by commas " , ".

Range, (—), 1, 0, , 1, 0, , 2, , (—), 1, 2, 0, , 1, 5, 0, , 5, 0

Next, program the equation for the first graph.

Graph, X, x, 4, —, X, x, 3, —, 2, 4, X, x², +, 4, X, +, 8, 0

Finally, program the equation for the second graph.

Graph, 1, 0, X, —, 3, 0

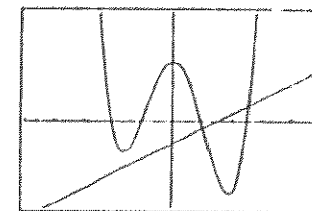
Total 49 steps

When inputting this program, press [EXE] after input of the ranges and the first equation.

```
Range -10,10,2,-
120,150,50
Graph Y=Xx^4-Xx^3-24X^2+4X+80
Graph Y=10X-30
```

The following should appear on the display when the program is executed:

[Prog] 0 [EXE]



A "▲" can be input in place of the [EXE] key operation after the first equation to suspend execution after the first graph is produced. To continue execution to the next graph, press [EXE].

The procedure outlined above can be used to produce a wide variety of graphs.

The library at the end of this manual includes a number of example graph programming.