

# Chapter

# 8

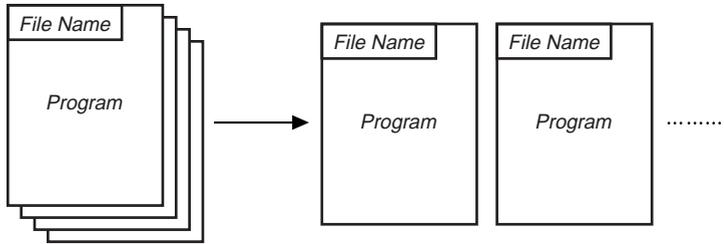


## Programming

1. Before Programming
2. Programming Examples
3. Debugging a Program
4. Calculating the Number of Bytes Used by a Program
5. Searching for a File
6. Editing Program Contents
7. Deleting a Program
8. Useful Program Commands
9. Command Reference
10. Text Display
11. Using Calculator Functions in Programs

# 1. Before Programming

The programming function helps to make complex, often-repeated calculations quick and easy. Commands and calculations are executed sequentially, just like the manual calculation multistatements. Multiple programs can be stored under file names for easy recall and editing.



Select the **PRGM** icon in the Main Menu and enter the PRGM Mode. When you do, a program list appears on the display.

Selected memory area —  
(use ▲ and ▼ to move)



- F1** (EXE) ..... Execute program
- F2** (EDIT) ..... Program edit
- F3** (NEW) ..... New program



- F1** (DEL) ..... Specific program delete
- F2** (DEL•A) .... Delete all
- F3** (SRC) ..... File name search

Press to return to the previous menu.

- If there are not programs stored in memory when you enter the PRGM Mode, the message **"No Programs"** appears on the display and only the NEW item (**F3**) is shown in the function menu.



P.129

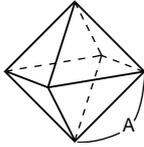
P.129

P.126

## 2. Programming Examples

**Example 1** To calculate the surface area and volume of three regular octahedrons of the dimensions shown in the table below

Store the calculation formula under the file name OCTA.



Length of One Side (A)	Surface Area (S)	Volume (V)
7 cm	cm <sup>2</sup>	cm <sup>3</sup>
10 cm	cm <sup>2</sup>	cm <sup>3</sup>
15 cm	cm <sup>2</sup>	cm <sup>3</sup>

The following are the formulas used for calculating surface area S and volume V of a regular octahedron for which the length of one side is known.

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

When inputting a new formula, you first register the file name and then input the actual program.

### •To register a file name

**Example** To register the file name OCTA

- Note that a file name can be up to eight characters long.

1. While the program list is on the display, press **F3** (NEW).

**F3** (NEW)

```
Program Name
[ ]
```

```
SYBL
```

**F4**

**F4** (SYBL) ..... Symbol menu

2. Input the name of the file.

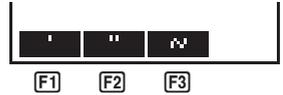
**O C T A**

```
Program Name
[OCTA ]
```

- The cursor changes form to indicate alpha character input.
- The following are the characters you can use in a file name:  
A through Z, spaces, [, ], {, }, ', ", ~, 0 through 9, ., +, -, ×, ÷

- Pressing **F4** (SYBL) displays a menu of symbols that can be input.

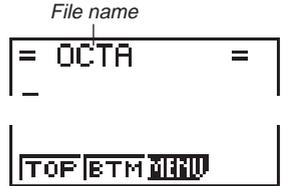
**F4** (SYBL)



- You can delete a character while inputting a file name by moving the cursor to the character you want to delete and pressing **DEL**.

- Press **EXE** to register the file name and change to the program input screen.

**EXE**



- Registering a file name uses 17 bytes of memory.
- The file name input screen remains on the display if you press **EXE** without inputting a file name.
- To exit the file name input screen and return to the program list without registering a file name, press **QUIT**.

### •To input a program

Use the program input screen to input the contents of a program.



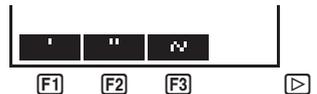
P.127

P.127

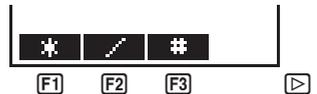
- F1** (TOP) ..... Top of program
- F2** (BTM) ..... Bottom of program
- F3** (MENU) .... Mode menu

- Pressing **▶** displays a menu of symbols that can be input into a program.

**▶**



**▶**



Press **▶** to return to the previous menu.



●To change modes in a program

- Pressing **F3** (MENU) while the program input screen is on the display causes a mode change menu to appear. You can use this menu to input mode changes into your programs. For details on each of these modes, see “Using the Main Menu”, as well as the sections of this manual that describe what you can do in each mode.

**F3** (MENU)



- Pressing **SHIFT** **SETUP** displays a menu of commands that can be used to change set up screen settings inside a program. For details on each of these commands, see “To change a mode set up”.

**SHIFT** **SETUP**



Actual program contents are identical to manual calculations. The following shows how the calculation of the surface area and volume of a regular octahedron would be calculated using a manual calculation.

Surface Area S .. **2** **X** **SHIFT** **✓** **3** **X** <value of A> **x<sup>2</sup>** **EXE**

Volume V ..... **SHIFT** **✓** **2** **÷** **3** **X** <value of A> **^** **3** **EXE**

You could also perform this calculation by assigning the value for the length of one side to variable A.

Length of One Side A

..... <value of A> **→** **ALPHA** **A** **EXE**

Surface Area S ..  $2 \times \sqrt{3} \times A^2$   $\times$   $\alpha$   $x^2$  EXE

Volume V .....  $\sqrt{2+3} \times A^3$   $\times$   $\alpha$   $\wedge$  3 EXE

If you simply input the manual calculations shown above however, the calculator would execute them from beginning to end, without stopping. The following commands make it possible to interrupt a calculation for input of values and display of intermediate results.

**?**: This command pauses program execution and displays a question mark as a prompt for input of a value to assign to a variable. The syntax for this command is: ?  $\rightarrow$  <variable name>.

**▲**: This command pauses program execution and displays the last calculation result obtained or text. It is similar to pressing **EXE** in a manual calculation.

- For full details on using these and other commands, see "Useful Program Commands".



The following shows examples of how to actually use the **?** and **▲** commands.

SHIFT PRGM  $\rightarrow$  F1 (?)  $\rightarrow$  ALPHA A  $\rightarrow$  F2 (:)  
 $2 \times \sqrt{3}$   $\times$   $\alpha$   $x^2$   
 $\rightarrow$   $\rightarrow$  F2 (▲)



SHIFT  $\sqrt{2+3}$   $\times$   $\alpha$   $\wedge$  3



QUIT QUIT



**•To run a program**

1. While the program list is on the display, use  $\blacktriangle$  and  $\blacktriangledown$  to highlight the name of the program you want to run.
2. Press **F1** (EXE) or **EXE** to run the program.

Let's try running the program we input above.

Length of One Side (A)	Surface Area (S)	Volume (V)
7 cm	169.7409791 cm <sup>2</sup>	161.6917506 cm <sup>3</sup>
10 cm	346.4101615 cm <sup>2</sup>	471.4045208 cm <sup>3</sup>
15 cm	779.4228634 cm <sup>2</sup>	1590.990258 cm <sup>3</sup>

```
Program List
UC1H
```

```
EXE EDIT NEW
[F1]
```

[F1] (EXE) or [EXE]

```
?
?
```

[7] [EXE]  
(Value of A)

```
?
7
169.7409791
- DISP -
```

Intermediate result produced by ▲

[EXE]

```
?
7
169.7409791
161.6917506
```

[EXE]

```
?
7
169.7409791
161.6917506
?
```

[1] [0] [EXE]

```
161.6917506
?
10
346.4101615
- DISP -
```

[EXE]

```
161.6917506
?
10
346.4101615
471.4045208
```

⋮

⋮



- Pressing [EXE] while the program's final result is on the display re-executes the program.
- You can also run a program while in the **RUN Mode** by inputting:  
Prog "<file name>" [EXE].
- An error (Go ERROR) occurs if the program specified by Prog "<file name>" cannot be found.

### 3. Debugging a Program

A problem in a program that keeps the program from running correctly is called a “bug,” and the process of eliminating such problems is called “debugging.” Either of the following symptoms indicates that your program contains bugs and that debugging is required.

- Error messages appearing when the program is run
- Results that are not within your expectations

#### ●To eliminate bugs that cause error messages

An error message, like the one shown below, appears whenever something illegal occurs during program execution.

```
Ma ERROR
```



P.175

When such a message appears, press ◀ or ▶ to display the location where the error was generated, along with the cursor. Check the “Error Message Table” for steps you should take to correct the situation.

#### ●To eliminate bugs that cause bad results

If your program produces results that are not what you normally expect, check the contents of the program and make necessary changes. See “Editing Program Contents” for details on how to change program contents.



P.126

## 4. Calculating the Number of Bytes Used by a Program

This unit comes with 7 kbytes of memory. A byte is a unit of memory that can be used for storage of data.

There are two types of commands: 1-byte commands and 2-byte commands.

- Examples of 1-byte commands: sin, cos, tan, log, (, ), A, B, C, 1, 2, etc.
- Examples of 2-byte commands: Lbl 1, Goto 2, etc.

While the cursor is located inside of a program, each press of ◀ or ▶ causes the cursor to move one byte.



P.37

- You can check how much memory has been used and how much remains at any time by selecting the **MEM** icon in the Main Menu and entering the MEM Mode. See “Memory Status (MEM)” for details.

## 5. Searching for a File

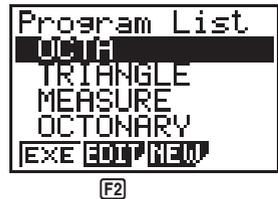
You can search for a specific file name using any of the three following methods.

- Scroll Search — scroll through the file names in the program list.
- File Name Search — input the name of the file.
- Initial Character Search — input the first few letters of the name of the file.

### ●To find a file using scroll search

**Example** To use scroll search to recall the program named OCTA

1. While the program list is on the display, use  $\blacktriangle$  and  $\blacktriangledown$  to scroll through the list of program names until you find the one you want.



2. When the highlighting is located at the name of the file you want, press  $F2$  (EDIT) to recall it.

$F2$ (EDIT)



### ●To find a file using file name search

**Example** To use file name search to recall the program named OCTA

1. While the program list is on the display, press  $F3$  (NEW) and input the name of the file you want to find.

$F3$ (NEW)

[O] [C] [T] [A]



2. Press  $F5$  to recall the program.
  - If there is no program whose file name matches the one you input, a new file is created using the input name.

●To find a file using initial character search

**Example** To use initial character search to recall the program named OCTA

1. While the program list is on the display, press **F3** (SRC) and input the initial characters of the file you want to find.

**F3** (SRC)



2. Press to search.



- All files whose file names start with the characters you input are recalled.
  - If there is no program whose file name starts with the characters you input, the message **“Not Found”** appears on the display. If this happens, press to clear the error message.
3. Use and to highlight the file name of the program you want to recall and then press (EDIT) to recall it.

## 6. Editing Program Contents

●To edit program contents

1. Find the file name of the program you want in the program list.
2. Recall the program.
  - The procedures you use for editing program contents are identical to those used for editing manual calculations. For details, see “Making Corrections”.





- The following function keys are also useful when editing program contents.

**F1** (TOP) ..... Moves the cursor to the top of the program

```

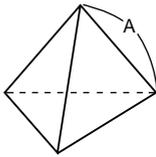
= OCTA =
2→A:2×√3×A²,
√2÷3×A³
    
```

**F2** (BTM) ..... Moves the cursor to the bottom of the program

```

= OCTA =
?→A:2×√3×A²,
√2÷3×A³_
    
```

**Example 2** To use the OCTA program to create a program that calculates the surface area and volume of regular tetrahedrons when the length of one side is known



Length of One Side (A)	Surface Area (S)	Volume (V)
7 cm	cm <sup>2</sup>	cm <sup>3</sup>
10 cm	cm <sup>2</sup>	cm <sup>3</sup>
15 cm	cm <sup>2</sup>	cm <sup>3</sup>

The following are the formulas used for calculating surface area S and volume V of a regular tetrahedron for which the length of one side is known.

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

Use the following key operations when inputting the program.

Length of One Side A .. **SHIFT** **PRGM** **▷** **F1** (?) **→** **ALPHA** **A** **▷** **F2** (:)

Surface Area S ..... **SHIFT** **✓** **3** **✕** **ALPHA** **A** **x<sup>2</sup>** **▷** **▷** **F2** (**▲**)

Volume V ..... **SHIFT** **✓** **2** **÷** **1** **2** **✕** **ALPHA** **A** **∧** **3**

Compare this with the program for calculating the surface area and volume of a regular octahedron.

Length of One Side A .. **SHIFT** **PRGM** **▷** **F1** (?) **→** **ALPHA** **A** **▷** **F2** (:)

Surface Area S ..... **2** **✕** **SHIFT** **✓** **3** **✕** **ALPHA** **A** **x<sup>2</sup>** **▷** **▷** **F2** (**▲**)

Volume V ..... **SHIFT** **✓** **2** **÷** **3** **✕** **ALPHA** **A** **∧** **3**

As you can see, you can produce the TETRA program by making the following changes in the OCTA program.

- Deleting **2** **✕** (underlined using a wavy line above)
- Changing **3** to **1** **2** (underlined using a solid line above)

Let's edit the program.

**F2**(EDIT)

```

= OCTA =
?→A:2×√3×A²,
√2÷3×A³
```

▶▶▶▶ **DEL** **DEL**

```

= OCTA =
?→A:√3×A²,
√2÷3×A³
```

▼◀ **SHIFT** **INS** **1** **2**

```

= OCTA =
?→A:√3×A²,
√2÷123×A³
```

**DEL**

```

= OCTA =
?→A:√3×A²,
√2÷12×A³
```

**QUIT**

Let's try running the program.

Length of One Side (A)	Surface Area (S)	Volume (V)
7 cm	84.87048957 cm <sup>2</sup>	40.42293766 cm <sup>3</sup>
10 cm	173.2050808 cm <sup>2</sup>	117.8511302 cm <sup>3</sup>
15 cm	389.7114317 cm <sup>2</sup>	397.7475644 cm <sup>3</sup>

```

Program List
OCTA
```

```

EXE EDIT NEW
```

**F1**

**F1** (EXE) or **EXE**

```

?
```

**7** **EXE**

(Value of A)

```

?
?
84.87048957
- DISP -
```

**EXE**

```

?
?
84.87048957
40.42293766
```

EXE

```
?
7
84.87048957
40.42293766
?
```

1 0 EXE

```
40.42293766
?
10
173.2050808
- Disp -
```

EXE

```
40.42293766
?
10
173.2050808
117.8511302
```

⋮

⋮

## 7. Deleting a Program

There are two different ways to delete a file name and its program.

- Specific program delete
- All program delete

### ●To delete a specific program

1. While the program list is on the display, use  $\blacktriangle$  and  $\blacktriangledown$  to move the highlighting to the name of the program you want to delete.

2. Press  $\blacktriangleright$  **F1** (DEL).

$\blacktriangleright$  **F1** (DEL)

```
YES NO
F1 F4
```

3. Press **F1** (YES) to delete the selected program or **F4** (NO) to abort the operation without deleting anything.

### ●To delete all programs

1. While the program list is on the display, press  $\blacktriangleright$  **F2** (DEL•A).



**F2** (DEL•A)



- Press **F1** (YES) to delete all the programs in the list or **F4** (NO) to abort the operation without deleting anything.
  - You can also delete all programs using the **MEM Mode**. See “Clearing Memory Contents” for details.

## 8. Useful Program Commands

In addition to calculation commands, this calculator also includes a variety of relational and jump commands that can be used to create programs that make repeat calculations quick and easy.

### Program Menu

Press **SHIFT** **PRGM** to display the program menu.

**SHIFT** **PRGM**



- F1** (COM) ..... Program command menu
- F2** (CTL) ..... Control command menu
- F3** (JUMP) ..... Jump command menu



- F1** (?) ..... Input command
- F2** (▲) ..... Output command
- F3** (CLR) ..... Clear command menu
- F4** (DISP) ..... Display command menu



- F1** (REL) ..... Conditional jump relational operator menu
- F2** (:) ..... Multi-statement command

Press to return to the previous menu.



**Program Command Menu (COM)**

While the program menu is on the display, press **F1** (COM) to display the program command menu.

**F1**(COM)



- F1** (If) ..... If command
- F2** (Then) ..... Then command
- F3** (Else) ..... Else command
- F4** (IfEnd) ..... IfEnd command

**▶**



- F1** (For) ..... For command
- F2** (To) ..... To command
- F3** (Step) ..... Step command
- F4** (Next) ..... Next command

**▶**



- F1** (While) ..... While command
- F2** (WhileEnd) ..... WhileEnd command
- F3** (Do) ..... Do command
- F4** (LpWhile) ..... LpWhile command

Press **▶** to return to the previous menu.

**Control Command Menu (CTL)**

While the program menu is on the display, press **F2** (CTL) to display the control command menu.

**F2**(CTL)



- F1** (Prog) ..... Prog command
- F2** (Rtrn) ..... Return command
- F3** (Brk) ..... Break command
- F4** (Stop) ..... Stop command

### Jump Command Menu (JUMP)

While the program menu is on the display, press **F3** (JUMP) to display the jump command menu.

**F3** (JUMP)



- F1** (Lbl) ..... Lbl command
- F2** (Goto) ..... Goto command
- F3** (=>) ..... => (jump) command

**F4**



- F1** (Lsz) ..... Lsz command
- F2** (Dsz) ..... Dsz command

Press **F4** to return to the previous menu.

### Clear Command Menu (CLR)

While the program menu is on the display, press **F3** (CLR) to display the clear command menu.

**F3** (CLR)



- F1** (Text) ..... ClrText command
- F2** (Grph) ..... ClrGraph command
- F3** (List) ..... ClrList command

### Display Command Menu (DISP)

While the program menu is on the display, press **F4** (DISP) to display the display command menu.

**F4** (DISP)



- F1** (Stat) ..... DrawStat command
- F2** (Grph) ..... DrawGraph command
- F3** (TABL) ..... Table & Graph command menu

Pressing **F3** (TABL) while the display command menu is on the display causes the Table & Graph command menu to appear.

**F3** (TABL)



**F1**    **F2**    **F3**

- F1** (Tabl) ..... DispTable command
- F2** (G•Con) .... DrawTG-Con command
- F3** (G•Plt) ..... DrawTG-Plt command

### Conditional Jump Relational Operator Menu (REL)

While the program menu is on the display, press **▶▶ F1** (REL) to display the conditional jump relational operator menu.

**▶▶ F1** (REL)



**F1**    **F2**    **F3**    **F4**    **▶**

- F1** (=) ..... Relational operator =
- F2** (≠) ..... Relational operator ≠
- F3** (>) ..... Relational operator >
- F4** (<) ..... Relational operator <

**▶**



**F1**    **F2**    **▶**

- F1** (≥) ..... Relational operator ≥
- F2** (≤) ..... Relational operator ≤

Press **▶** to return to the previous menu.

## 9. Command Reference

### ■ Command Index

Break .....	140
ClrGraph .....	144
ClrList .....	144
ClrText .....	144
DispTable .....	145
Do~LpWhile .....	139
DrawTG-Con, DrawTG-Plt .....	145
DrawGraph .....	145
DrawStat .....	144
Dsz .....	142
For~To~Next .....	138
For~To~Step~Next .....	138
Goto~Lbl .....	142
If~Then .....	136
If~Then~Else .....	137
If~Then~Else~IfEnd .....	137
If~Then~IfEnd .....	136
Isz .....	143
Prog .....	140
Return .....	141
Stop .....	141
While~WhileEnd .....	139
? (Input Command) .....	135
▲ (Output Command) .....	135
: (Multi-statement Command) .....	135
↵ (Carriage Return) .....	135
⇒ (Jump Code) .....	143
=, ≠, >, <, ≥, ≤ (Relational Operators) .....	145

The following are conventions that are used in this section when describing the various commands.

- Boldface Text** ..... Actual commands and other items that always must be input are shown in boldface.
- {Curly Brackets}** ..... Curly brackets are used to enclose a number of items, one of which must be selected when using a command. Do not input the curly brackets when inputting a command.
- [Square Brackets]** ..... Square brackets are used to enclose items that are optional. Do not input the square brackets when inputting a command.

Numeric Expressions . Numeric expressions (such as 10, 10 + 20, A) indicate constants, calculations, numeric constants, etc.

Alpha Characters ..... Alpha characters indicate literal strings (such as AB).

## ■ Basic Operation Commands

### ? (Input Command)

**Function:** Prompts for input of values for assignment to variables during program execution.

**Syntax:** ? → <variable name>

**Example:** ? → A ↵

**Description:**

1. This command momentarily interrupts program execution and prompts for input of a value or expression for assignment to a variable. When the input command is executed, "?" to appears on the display and the calculator stands by for input.
2. Input in response to the input command must be a value or an expression, and the expression cannot be a multi-statement.

### ▲ (Output Command)

**Function:** Displays and intermediate result during program execution.

**Description:**

1. This command momentarily interrupts program execution and displays alpha character text or the result of the calculation immediately before it.
2. The output command should be used at locations where you would normally press the **EXE** key during a manual calculation.

### : (Multi-statement Command)

**Function:** Connects two statements for sequential execution without stopping.

**Description:**

1. Unlike the output command (▲), statements connected with the multi-statement command are executed non-stop.
2. The multi-statement command can be used to link two calculation expressions or two commands.
3. You can also use a carriage return indicated by ↵ in place of the multi-statement command.

### ↵ (Carriage Return)

**Function:** Connects two statements for sequential execution without stopping.

**Description:**

1. Operation of the carriage return is identical to that of the multi-statement command.

- Using a carriage return in place of the multi-statement command makes the displayed program easier to read.

## ■ Program Commands (COM)

### If~Then

**Function:** The Then-statement is executed only when the If-condition is true (non-zero).

**Syntax:**

$$\text{If } \begin{array}{c} \text{<condition>} \\ \text{numeric expression} \end{array} \left\{ \begin{array}{c} \blacktriangleleft \\ \vdots \\ \blacktriangleright \end{array} \right\} \text{ Then } \text{<statement>} \left[ \left\{ \begin{array}{c} \blacktriangleleft \\ \vdots \\ \blacktriangleright \end{array} \right\} \text{<statement>} \right]$$

**Parameters:** condition, numeric expression

**Description:**

- The Then-statement is executed only when the If-condition is true (non-zero).
- If the condition is false (0), the Then-statement is not executed.
- An If-condition must always be accompanied by a Then-statement. Omitting the Then-statement results in an error (Syn ERROR).

**Example:** If A = 0  $\blacktriangleleft$   
 Then "A = 0"

### If~Then~IfEnd

**Function:** The Then-statement is executed only when the If-condition is true (non-zero). The IfEnd-statement is always executed: after the Then-statement is executed or directly after the If-condition when the If-condition is false (0).

**Syntax:**

$$\text{If } \begin{array}{c} \text{<condition>} \\ \text{numeric expression} \end{array} \left\{ \begin{array}{c} \blacktriangleleft \\ \vdots \\ \blacktriangleright \end{array} \right\} \text{ Then } \text{<statement>} \left[ \left\{ \begin{array}{c} \blacktriangleleft \\ \vdots \\ \blacktriangleright \end{array} \right\} \text{<statement>} \right] \left\{ \begin{array}{c} \blacktriangleleft \\ \vdots \\ \blacktriangleright \end{array} \right\} \text{ IfEnd}$$

**Parameters:** condition, numeric expression

**Description:**

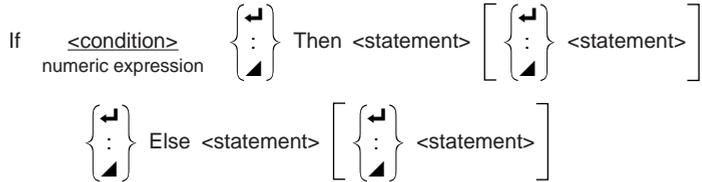
This command is almost identical to If~Then. The only difference is that the IfEnd-statement is always executed, regardless of whether the If-condition is true (non-zero) or false (0).

**Example:** If A = 0  $\blacktriangleleft$   
 Then "A = 0"  $\blacktriangleleft$   
 IfEnd  $\blacktriangleleft$   
 "END"

**If~Then~Else**

**Function:** The Then-statement is executed only when the If-condition is true (non-zero). The Else-statement is executed when the If-condition is false (0).

**Syntax:**



**Parameters:** condition, numeric expression

**Description:**

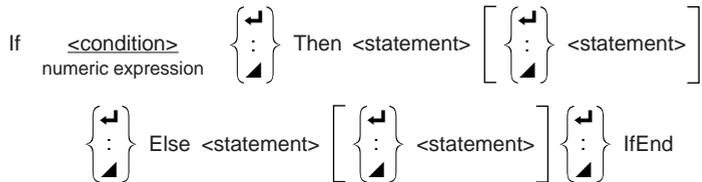
1. The Then-statement is executed when the If-conditions is true (non-zero).
2. The Else-statement is executed when the If-conditions is false (zero).

**Example:** If A = 0 ↵  
 Then "TRUE" ↵  
 Else "FALSE"

**If~Then~Else~IfEnd**

**Function:** The Then-statement is executed only when the If-condition is true (non-zero). The Else-statement is executed when the If-condition is false (0). The IfEnd-statement is always executed following either the Then-statement or Else-statement.

**Syntax:**



**Parameters:** condition, numeric expression

**Description:**

This command is almost identical to If~Then~Else. The only difference is that the IfEnd-statement is always executed, regardless of whether the If-condition is true (non-zero) or false (0).

**Example:** ? → A ↵  
 If A = 0 ↵  
 Then "TRUE" ↵  
 Else "FALSE" ↵  
 IfEnd ↵  
 "END"

**For~To~Next**

**Function:** This command repeats everything between the For-statement and the Next-statement. The starting value is assigned to the control variable with the first execution, and the value of the control variable is incremented by one with each execution. Execution continues until the value of the control variable exceeds the ending value.

**Syntax:**

$$\text{For } \langle \text{starting value} \rangle \rightarrow \langle \text{control variable name} \rangle \text{ To } \langle \text{ending value} \rangle \left\{ \begin{array}{l} \leftarrow \\ \vdots \\ \blacktriangle \end{array} \right\}$$

$$\left[ \langle \text{statement} \rangle \left\{ \begin{array}{l} \leftarrow \\ \vdots \\ \blacktriangle \end{array} \right\} \right] \text{ Next}$$

**Parameters:**

- control variable name: A to Z
- starting value: value or expression that produces a value (i.e.  $\sin x$ , A, etc.)
- ending value: value or expression that produces a value (i.e.  $\sin x$ , A, etc.)

**Description:**

1. When the starting value of the control variable is greater than the ending value, execution continues from the statement following Next, without executing the statements between For and Next.
2. A For-statement must always have a corresponding Next-statement, and the Next-statement must always come after its corresponding For-statement.
3. The Next-statement defines the end of the loop created by For~Next, and so it must always be included. Failure to do so results in an error (Syn ERROR).

**Example:** For 1 → A To 10 ↵

A × 3 → B ↵

B ▲

Next

**For~To~Step~Next**

**Function:** This command repeats everything between the For-statement and the Next-statement. The starting value is assigned to the control variable with the first execution, and the value of the control variable is changed according to the step value with each execution. Execution continues until the value of the control variable exceeds the ending value.

**Syntax:**

$$\text{For } \langle \text{starting value} \rangle \rightarrow \langle \text{control variable name} \rangle \text{ To } \langle \text{ending value} \rangle \text{ Step } \langle \text{step value} \rangle \left\{ \begin{array}{l} \leftarrow \\ \vdots \\ \blacktriangle \end{array} \right\}$$

Next

**Parameters:**

- control variable name: A to Z
- starting value: value or expression that produces a value (i.e.  $\sin x$ , A, etc.)
- ending value: value or expression that produces a value (i.e.  $\sin x$ , A, etc.)
- step value: numeric value (omitting this value sets the step to 1)

**Description:**

1. This command is basically identical to For~To~Next. The only difference is that you can specify the step.
2. Omitting the step value automatically sets the step to 1.
3. Making the starting value less than the ending value and specifying a positive step value causes the control variable to be incremented with each execution. Making the starting value greater than the ending value and specifying a negative step value causes the control variable to be decremented with each execution.

**Example:** For 1 → A To 10 Step 0.1 ↵

```
A × 3 → B ↵
B ▲
Next
```

**Do~LpWhile**

**Function:** This command repeats specific commands as long as its condition is true (non-zero).

**Syntax:**

$$\text{Do } \left\{ \begin{array}{c} \text{↵} \\ \vdots \\ \text{▲} \end{array} \right\} \sim \text{LpWhile } \langle \text{expression} \rangle$$

**Parameters:** expression

**Description:**

1. This command repeats the commands contained in the loop as long as its condition is true (non-zero). When the condition becomes false (0), execution proceeds from the statement following the LpWhile-statement.
2. Since the condition comes after the LpWhile-statement, the condition is tested (checked) after all of the commands inside the loop are executed.

**Example:** Do ↵

```
? → A ↵
A × 2 → B ↵
B ▲
LpWhile B >10
```

**While~WhileEnd**

**Function:** This command repeats specific commands as long as its condition is true (non-zero).

**Syntax:**

$$\text{While } \langle \text{expression} \rangle \left\{ \begin{array}{c} \text{↵} \\ \vdots \\ \text{▲} \end{array} \right\} \sim \text{WhileEnd}$$

**Parameters:** expression

**Description:**

1. This command repeats the commands contained in the loop as long as its condition is true (non-zero). When the condition becomes false (0), execution proceeds from the statement following the WhileEnd-statement.
2. Since the condition comes after the While-statement, the condition is tested (checked) before the commands inside the loop are executed.

**Example:** 10 → A ↵  
           While A > 0 ↵  
           A - 1 → A ↵  
           "GOOD" ↵  
           WhileEnd

## ■ Program Control Commands (CTL)

**Break**

**Function:** This command breaks execution of a loop and continues from the next command following the loop.

**Syntax:** Break ↵

**Description:**

1. This command breaks execution of a loop and continues from the next command following the loop.
2. This command can be used to break execution of a For-statement, Do-statement, and While-statement.

**Example:** While A>0 ↵  
           If A > 2 ↵  
           Then Break ↵  
           IfEnd ↵  
           WhileEnd ↵  
           A ▲ ←————— Executed after Break

**Prog**

**Function:** This command specifies execution of another program as a subroutine. In the RUN Mode, this command executes a new program.

**Syntax:** Prog "file name" ↵

**Example:** Prog "ABC" ↵

**Description:**

1. Even when this command is located inside of a loop, its execution immediately breaks the loop and launches the subroutine.
2. This command can be used as many times as necessary inside of a main routine to call up independent subroutines to perform specific tasks.



```

Example: For 2 → I To 10 ↵
            If I = 5 ↵
            Then "STOP" : Stop ↵
            IfEnd ↵
            Next
    
```

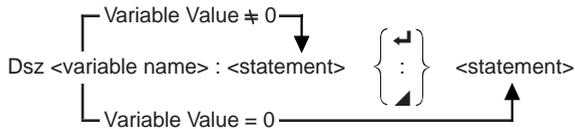
This program counts from 2 to 10. When the count reaches 5, however, it terminates execution and displays the message "STOP."

## ■ Jump Commands (JUMP)

### Dsz

**Function:** This command is a count jump that decrements the value of a control variable by 1, and then jumps if the current value of the variable is zero.

**Syntax:**



**Parameters:**

Variable Name: A to Z

[Example] Dsz B : Decrements the value assigned to variable B by 1.

**Description:**

This command decrements the value of a control variable by 1, and then tests (checks) it. If the current value is non-zero, execution continues with the next statement. If the current value is zero, execution jumps to the statement following the multi-statement command (:), display command (▲), or carriage return (↵).

```

Example: 10 → A : 0 → C :
            Lbl 1 : ? → B : B+C → C :
            Dsz A : Goto 1 : C ÷ 10
    
```

This program prompts for input of 10 values, and then calculates the average of the input values.

### Goto~Lbl

**Function:** This command performs an unconditional jump to a specified location.

**Syntax:** Goto <value or variable> ~ Lbl <value or variable>

**Parameters:** Value (from 0 to 9), variable (A to Z)

**Description:**

1. This command consists of two parts: Goto *n* (where *n* is a value from 0 to 9) and Lbl *n* (where *n* is the value specified for Goto). This command causes program execution to jump to the Lbl-statement whose value matches that specified by the Goto-statement.
2. This command can be used to loop back to the beginning of a program or to jump to any location within the program.

3. This command can be used in combination with conditional jumps and count jumps.
4. If there is no Lbl-statement whose value matches that specified by the Goto-statement, an error (Go ERROR) occurs.

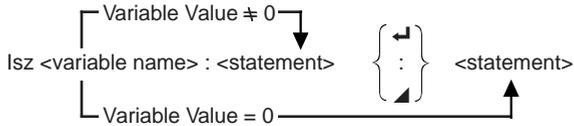
**Example:** ? → A : ? → B : Lbl 1 :  
 ? → X : A × X + B ▲  
 Goto 1

This program calculates  $y = AX + B$  for as many values for each variable that you want to input. To quit execution of this program, press **AC**.

**Isz**

**Function:** This command is a count jump that increments the value of a control variable by 1, and then jumps if the current value of the variable is zero.

**Syntax:**



**Parameters:**

Variable Name: A to Z

[Example] Isz A : Increments the value assigned to variable A by 1.

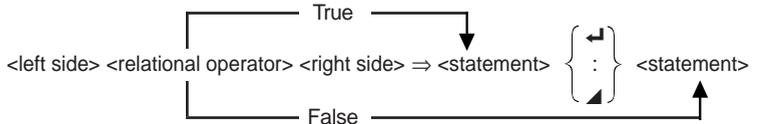
**Description:**

This command increments the value of a control variable by 1, and then tests (checks) it. If the current value is non-zero, execution continues with the next statement. If the current value is zero, execution jumps to the statement following the multi-statement command (:), display command (▲), or carriage return (↵).

**⇒ (Jump Code)**

**Function:** This code is used to set up conditions for a conditional jump. The jump is executed whenever the conditions are false.

**Syntax:**



**Parameters:**

left side/right side: variable (A to Z), numeric constant, variable expression (such as:  $A \times 2$ )

relational operator: =, ≠, >, <, ≥, ≤

**Description:**

1. The conditional jump compares the contents of two variables or the results of two expressions, and a decision is made whether or not to execute the jump based on the results of the comparison.
2. If the comparison returns a true result, execution continues with the statement following the  $\Rightarrow$  command. If the comparison returns a false result, execution jumps to the statements following the multi-statement command (:), display command ( $\blacktriangleleft$ ), or carriage return ( $\blacktriangledown$ ).

**Example:** Lbl 1 : ?  $\rightarrow$  A :

$A \geq 0 \Rightarrow \sqrt{A} \blacktriangleleft$

Goto 1

With this program, inputting a value of zero or greater calculates and displays the square root of the input value. Inputting a value less than zero returns to the input prompt without calculating anything.

**■ Clear Commands (CLR)**

**ClrGraph**

**Function:** This command clears the graph screen.

**Syntax:** ClrGraph  $\blacktriangledown$

**Description:** This command clears the graph screen during program execution.

**ClrList**

**Function:** This command clears list data.

**Syntax:** ClrList  $\blacktriangledown$

**Description:** This command clears the contents of the currently selected list (List 1 to List 6) during program execution.

**ClrText**

**Function:** This command clears the text screen.

**Syntax:** ClrText  $\blacktriangledown$

**Description:**

This command clears text from the screen during program execution.

**■ Display Commands (DISP)**

**DrawStat**

**Function:** This draws a statistical graph.

**Syntax:**

DrawStat  $\blacktriangledown$

**Description:**

This command draws a statistical graph in accordance with conditions defined within the program.

**DrawGraph**

**Function:** This command draws a graph.

**Syntax:** DrawGraph ↵

**Description:** This command draws a graph in accordance with the drawing conditions defined within the program.

**DispTable**

**Function:** These commands display numeric tables.

**Syntax:**

DispTable ↵

**Description:**

These commands generate numeric tables during program execution in accordance with conditions defined within the program.

**DrawTG-Con, DrawTG-Plt**

**Function:** These commands graph functions.

**Syntax:**

DrawTG-Con ↵

DrawTG-Plt ↵

**Description:**

1. These commands graph functions in accordance with conditions defined within the program.
2. DrawTG-Con produces a connect type graph, while DrawTG-Plt produces a plot type graph.

**■ Conditional Jump Relational Operators (REL)**

=, ≠, >, <, ≥, ≤

**Function:** These relational operators are used in combination with the conditional jump command.

**Syntax:**

<left side> <relational operator> <right side> ⇒ <statement>  $\left. \begin{array}{c} \text{↵} \\ \vdots \\ \text{↵} \end{array} \right\}$  <statement>  
 (With Jump Code)

**Parameters:**

left side/right side: variable (A to Z), numeric constant, variable expression (such as:  $A \times 2$ )

relational operator: =, ≠, >, <, ≥, ≤

**Description:**

1. The following six relational operators can be used in the conditional jump command

<left side> = <right side> : true when <left side> equals <right side>

<left side> ≠ <right side> : true when <left side> does not equal <right side>

<left side> > <right side> : true when <left side> is greater than <right side>

<left side> < <right side> : true when <left side> is less than <right side>

<left side> ≥ <right side> : true when <left side> is greater than or equal to <right side>

<left side> ≤ <right side> : true when <left side> is less than or equal to <right side>

2. See “⇒ (Jump Code)” for details on using the conditional jump.



## 10. Text Display

You can include text in a program by simply enclosing it between double quotation marks. Such text appears on the display during program execution, which means you can add labels to input prompts and results.

Program	Display
? → X	?
"X =" ? → X	X = ?

- If the text is followed by a calculation formula, be sure to insert a display command (▲) between the text and calculation.
- Inputting more than 13 characters causes the text to move down to the next line. The screen scrolls automatically if the text causes the screen to become full.

## 11. Using Calculator Functions in Programs

### ■ Using Graph Functions in a Program

You can incorporate graph functions into a program to draw complex graphs and to overlay graphs on top of each other. The following shows various types of syntax you need to use when programming with graph functions.

- View Window

View Window -5, 5, 1, -5, 5, 1 ◀



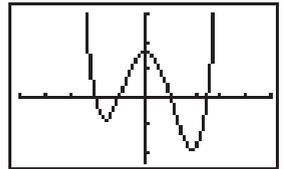
- Graph function input  
 $Y = \text{Type} \leftarrow \dots$  Specifies graph type.  
 $"X^2 - 3" \rightarrow Y1 \leftarrow$
- Graph draw operation  
 $\text{DrawGraph} \leftarrow$

**Example Program**

- ①  $\text{ClrGraph} \leftarrow$
- ② View Window  $-10, 10, 2, -120, 150, 50 \leftarrow$
- ③  $Y = \text{Type} \leftarrow$   
 $"X^4 - X^3 - 24X^2 + 4X + 80" \rightarrow Y1 \leftarrow$
- ⑤  $G \text{ SelOn } 1 \leftarrow$
- ⑥  $\text{DrawGraph} \leftarrow$

- ①  $\text{SHIFT} \text{ PRGM} \triangleright \text{F3} \text{ F2}$
- ②  $\text{SHIFT} \text{ F3} \text{ F1} \text{ QUIT}$
- ③  $\text{F3} \text{ F3} \text{ F2} \text{ F1} \text{ QUIT}$
- ④  $\text{VARS} \triangleright \text{F2} \text{ F1} \text{ QUIT}$
- ⑤  $\text{F3} \text{ F3} \text{ F1} \text{ F1}$
- ⑥  $\text{SHIFT} \text{ PRGM} \triangleright \text{F4} \text{ F2}$

Executing this program produces the result shown here.



**Using Table & Graph Functions in a Program**

Table & Graph functions in a program can generate numeric tables and perform graphing operations. The following shows various types of syntax you need to use when programming with Table & Graph functions.

- Table range setting  
 $1 \rightarrow \text{F Start} \leftarrow$   
 $5 \rightarrow \text{F End} \leftarrow$   
 $1 \rightarrow \text{F pitch} \leftarrow$
- Numeric table generation  
 $\text{DispTable} \leftarrow$
- Graph draw operation  
 $\text{Connect type: DrawTG-Con} \leftarrow$   
 $\text{Plot type: DrawTG-Plt} \leftarrow$

**Example Program**

- $\text{ClrGraph} \leftarrow$
- $\text{ClrText} \leftarrow$
- View Window  $0, 6, 1, -2, 106, 2 \leftarrow$
- $Y = \text{Type} \leftarrow$
- $"3X^2 - 2" \rightarrow Y1 \leftarrow$

- ① T SelOn 1 ↵
  - 0 → ② F Start ↵
  - 6 → ③ F End ↵
  - 1 → ④ F pitch ↵
  - ⑤ DispTable ▲
  - ⑥ DrawTG-Con
- ① **F3** **F4** **F1** **QUIT**
  - ② **VAR** **▶** **F3** **F1**
  - ③ **F2**
  - ④ **F3** **QUIT**
  - ⑤ **SHIFT** **PRGM** **▶** **F4** **F3** **F1** **QUIT**
  - ⑥ **SHIFT** **PRGM** **▶** **F4** **F3** **F2** **QUIT**

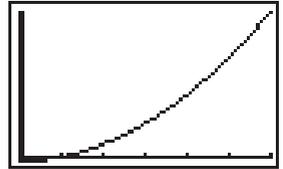
Executing this program produces the results shown here.

Numeric Table



Graph

**EXE**



P.80

### ■ Using List Sort Functions in a Program

These functions let you sort the data in lists into ascending or descending order.

- Ascending order

① SortA (② List 1, List 2, List 3)

Lists to be sorted (up to six can be specified)

① **F3** **F2** **F1** **QUIT**

② **OPTN** **F1** **F1**

- Descending order

SortD (List 1, List 2, List 3)

Lists to be sorted (up to six can be specified)



P.94

### ■ Using Statistical Calculations and Graphs in a Program

Including statistical calculations and graphing operations into program lets you calculate and graph statistical data.

#### ● To set conditions and draw a statistical graph

Following “StatGrph”, you must specify the following graph conditions:

- Graph draw/non-draw status (DrawOn/DrawOff)
- Graph Type
- x-axis data location (list name)
- y-axis data location (list name)
- Frequency data location (list name)
- Mark Type

The graph conditions that are required depends on the graph type. See “Changing Graph Parameters”.

- The following is a typical graph condition specification for a scatter diagram or *xy* line graph.

S-Gph1 DrawOn, Scatter, List1, List2, 1, Square ↵

In the case of an *xy* line graph, replace “Scatter” in the above specification with “*xy*Line”.

- The following is a typical graph condition specification for a single-variable graph.

S-Gph1 DrawOn, Hist, List1, List2 ↵

The same format can be used for the following types of graphs, by simply replacing “Hist” in the above specification with the applicable graph type.

Histogram: ..... Hist

Median Box: ..... MedBox

Normal Distribution: ..... N-Dist

- The following is a typical graph condition specification for a regression graph.

S-Gph1 DrawOn, Linear, List1, List2, List3 ↵

The same format can be used for the following types of graphs, by simply replacing “Linear” in the above specification with the applicable graph type.

Linear Regression: ..... Linear

Med-Med: ..... Med-Med

Quadratic Regression: .... Quad

Logarithmic Regression: .. Log

Exponential Regression: Exp

Power Regression : ..... Power

**Example Program**

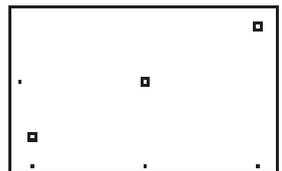
```

ClrGraph ↵
① S-WindAuto ↵
  {1, 2, 3} → ② List 1 ↵
  {1, 2, 3} → ③ List 2 ↵
④ S-Gph1 ⑤ DrawOn,
⑥ Scatter, List1, List2, 1, ⑦ Square ↵
⑧ DrawStat
    
```

```

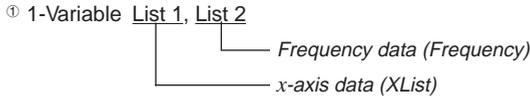
① SHIFT SETUP >>> F1 QUIT
② OPTN F1 F1
③ F1 QUIT
④ F3 F1 F2 F1 QUIT
⑤ F3 F1 F1 F1 QUIT
⑥ F3 F1 F2 > F1 QUIT
⑦ F3 F1 F4 F1 QUIT
⑧ SHIFT PRGM > F4 F1 QUIT
    
```

Executing this program produces the scatter diagram shown here.



## ■ Performing Statistical Calculations

- Single-variable statistical calculation

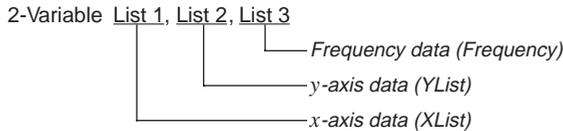


① **F3** **F1** **▶** **F1** **F1** **QUIT**

```

1-Variable
  x= 2.33333
  Σx= 14
  Σx²= 36
  x̄n= 0.74535
  
```

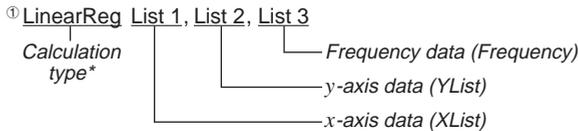
- Paired-variable statistical calculation



```

2-Variable
  x= 2.33333
  Σx= 14
  Σx²= 36
  x̄n= 0.74535
  
```

- Regression statistical calculation



① **F3** **F1** **▶** **F1** **▶** **F1** **QUIT**

```

LinearReg
  a= 0.64641
  b=-0.71186
  r= 0.87959
  y=ax+b
  
```

\* Any one of the following can be specified as the calculation type.

- LinearReg ..... linear regression
- Med-MedLine . Med-Med calculation
- QuadReg ..... quadratic regression
- LogReg ..... logarithmic regression
- ExpReg ..... exponential regression
- PowerReg ..... power regression