

Chapter

8

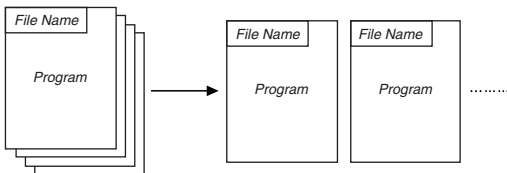


Programming

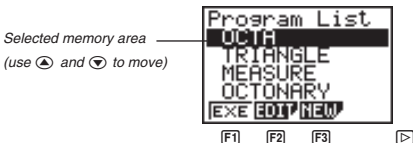
1. Before Programming
2. Programming Examples
3. Debugging a Program
4. Calculating the Number of Bytes Used by a Program
5. Secret Function
6. Searching for a File
7. Editing Program Contents
8. Deleting a Program
9. Useful Program Commands
10. Command Reference
11. Text Display
12. Using Calculator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



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.138

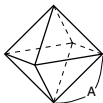
P.138

P.135

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 ²	cm ³
10 cm	cm ²	cm ³
15 cm	cm ²	cm ³

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)

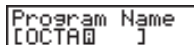


F3 (n0) Password registration

F4 (SYBL) Symbol menu

2. Input the name of the file.

O C T A



- 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, ., +, -, x, +

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

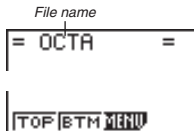
F4 (SYBL)



- 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.136

P.136

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

- Pressing **F4** displays a menu of symbols that can be input into a program.

F4



F4



Press **F4** 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> **x²** **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 ... $2 \times \text{SHIFT} \checkmark 3 \times \text{ALPHA} \Delta x^2 \text{EXE}$

Volume V $\text{SHIFT} \checkmark 2 \div 3 \times \text{ALPHA} \Delta \wedge 3 \text{EXE}$

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.

$\text{SHIFT} \text{PRGM} \triangleright \text{F1} (?) \rightarrow \text{ALPHA} \Delta \triangleright \text{F3} (:)$
 $2 \times \text{SHIFT} \checkmark 3 \times \text{ALPHA} \Delta x^2$
 $\triangleright \triangleright \text{F2} (\blacktriangle)$

$= \text{OCTA} =$
 $? \rightarrow A: 2 \times \sqrt{3} \times A^2,$
 $-$
 $?$ \blacktriangle CLR/DISP
 F1 F2

$\text{SHIFT} \checkmark 2 \div 3 \times \text{ALPHA} \Delta \wedge 3$

$= \text{OCTA} =$
 $? \rightarrow A: 2 \times \sqrt{3} \times A^2,$
 $\sqrt{2+3 \times A^3} -$

QUIT QUIT

Program List
 OCTA

•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 ²	161.6917506 cm ³
10 cm	346.4101615 cm ²	471.4045208 cm ³
15 cm	779.4228634 cm ²	1590.990258 cm ³

	<pre> Program List UCTH </pre>
	<pre> [EXE] [EDIT] [NEW] [F1] </pre>
[F1] (EXE) or [EXE]	<pre> ? </pre>
[7] [EXE] (Value of A)	<pre> ? ? 169.7409791 - DISP - </pre>
	<p style="text-align: center;"><i>Intermediate result produced by ▲</i></p>
[EXE]	<pre> ? ? 169.7409791 161.6917506 </pre>
[EXE]	<pre> ? ? 169.7409791 161.6917506 ? </pre>
[1] [0] [EXE]	<pre> 161.6917506 ? 10 346.4101615 - DISP - </pre>
[EXE]	<pre> 161.6917506 ? 10 346.4101615 471.4045208 </pre>
	...



P.149

- 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
```

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.

- Note that pressing ◀ or ▶ will not display the location of the error if the program is password protected.

•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.

4. Calculating the Number of Bytes Used by a Program

This unit comes with 20,000 bytes 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.

- 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.



P.200



P.133



P.135



P.37

5. Secret Function

When inputting a program, you can protect it with a password that limits access to the program contents to those who know the password. Password protected programs can be executed by anyone without inputting the password.

•To register a password

Example To create a program file under the name AREA and protect it with the password CASIO

1. While the program list is on the display, press **F3** (NEW) and input the file name of the new program file.

F3 (NEW)
A **R** **E** **A**



2. Press **F3** (π O) and then input the password.

F3 (π O)
C **A** **S** **I** **O**



- The password input procedure is identical to that used for file name input.

3. Press **EXE** to register the file name and password. Now you can input the contents of the program file.
 - Registration of a password uses 16 bytes of memory.
 - Pressing **EXE** without inputting a password registers the file name only, without a password.
4. After inputting the program, press **QUIT** to exit the program file and return to the program list. Files that are password protected are indicated by an asterisk to the right of the file name.

QUIT



•To recall a program

Example To recall the file named AREA which is protected by the password CASIO

1. In the program list, use \blacktriangle and \blacktriangledown to move the highlighting to the name of the program you want to recall.

2. Press F_2 (EDIT).

F_2 (EDIT)

```

Program Name
[AREA ]
Password?
[ ]
  
```

3. Input the password and press EXE to recall the program.

- The message "Mismatch" appears if you input the wrong password.

6. 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.

```

Program List
OCTA
TRIANGLE
AREA *
MEASURE
[EXE] [EDIT] [NEW]
  
```

F_2

2. When the highlighting is located at the name of the file you want, press F_2 (EDIT) to recall it.

F_2 (EDIT)

```

= OCTA =
2+A:2*√3*A²,
√2+3*A^3
  
```

●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

```
Program Name
[OCTA ]
```

2. Press **EXE** 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)
O C T

```
Search For
Program
[OCT ]
```

2. Press **EXE** to search.

EXE

```
Program List
OCTA
OCTONARY

[EXE EDIT NEW]
```

- 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 **OUT** to clear the error message.
3. Use **▲** and **▼** to highlight the file name of the program you want to recall and then press **F2** (EDIT) to recall it.

7. 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.

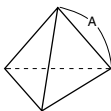
F1 (TOP) Moves the cursor to the top of the program

```
= 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 ²	cm ³
10 cm	cm ²	cm ³
15 cm	cm ²	cm ³

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** **▷** **F3** (:)

Surface Area S **SHIFT** **✓** **3** **✕** **ALPHA** **A** **x²** **▷** **▷** **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** **▷** **F3** (:)

Surface Area S **2** **✕** **SHIFT** **✓** **3** **✕** **ALPHA** **A** **x²** **▷** **▷** **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 X (underlined using a wavy line above)
- Changing 3 to 1 2 (underlined using a solid line above)

Let's edit the program.

F2 (EDIT)	<pre style="border: 1px solid black; padding: 5px;">= OCTA = 2→A:2×√3×A², √2÷3×A³</pre>
▶▶▶▶ DEL DEL	<pre style="border: 1px solid black; padding: 5px;">= OCTA = ?→A:√3×A², √2÷3×A³</pre>
▼◀ SHIFT INS 1 2	<pre style="border: 1px solid black; padding: 5px;">= OCTA = ?→A:√3×A², √2÷123×A³</pre>
DEL	<pre style="border: 1px solid black; padding: 5px;">= OCTA = ?→A:√3×A², √2÷12X)A³</pre>
QUIT	

Let's try running the program.

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

	<pre style="border: 1px solid black; padding: 5px;">Program List OCTA</pre>
	<pre style="border: 1px solid black; padding: 5px;">[EXE EDIT NEW] [F1]</pre>
F1 (EXE) or EXE	<pre style="border: 1px solid black; padding: 5px;">?</pre>
7 EXE (Value of A)	<pre style="border: 1px solid black; padding: 5px;">? 7 84.87048957 - DISP -</pre>

EXE

```
?
7
84.87048957
40.42293766
```

EXE

```
?
7
84.87048957
40.42293766
?
```

1 0 EXE

```
40.42293766
?
i0
173.2050808
- DISP -
```

EXE

```
40.42293766
?
i0
173.2050808
117.8511302
```

⋮

⋮

8. 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 **▲** and **▼** to move the highlighting to the name of the program you want to delete.
2. Press **▷** **F1** (DEL).

▷ **F1** (DEL)

```
YES NO
F1 F4
```

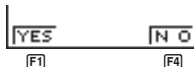
3. Press **F1** (YES) to delete the selected program or **F4** (NO) to abort the operation without deleting anything.

•To delete all programs

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



▶ **F2** (DEL•A)



2. 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.

9. 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 (I/O) Input/output command menu

F3 (:) 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)



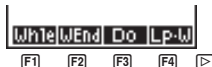
- F1** (If) If command
- F2** (Then) Then command
- F3** (Else) Else command
- F4** (If•End) IfEnd command

▶



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

▶



- F1** (While) While command
- F2** (WEnd) WhileEnd command
- F3** (Do) Do command
- F4** (Lp•W) 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)

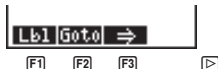


- F1** (Prog) Prog command
- F2** (Rtrn) Return command
- F3** (Brk) Break command
- F4** (Stop) Stop command

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

▶



F1 (Isz) Isz command

F2 (Dsz) Dsz command

Press **▶** 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** (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** (=) Relational operator =
- F2** (≠) Relational operator ≠
- F3** (>) Relational operator >
- F4** (<) Relational operator <

▶



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

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

Input/Output Commands Menu (I/O)

While the program menu is on the display, press **▶▶ F2** (I/O) to display the input/output command menu.

▶▶ F2 (I/O)



- F1** (Send) Send (command
- F2** (Recv) Receive (command

10. Command Reference

■ Command Index

Break	149
ClrGraph	153
ClrList.....	153
ClrText.....	153
DispTable	154
Do~LpWhile	148
DrawTG-Con, DrawTG-Plt.....	154
DrawGraph	154
DrawStat	153
Dsz	151
For~To~Next	147
For~To~Step~Next	147
Goto~Lbl	151
If~Then	145
If~Then~Else	146
If~Then~Else~IfEnd	146
If~Then~IfEnd	145
Isz	152
Prog	149
Receive(.....	154
Return	150
Send(.....	155
Stop	150
While~WhileEnd	148
? (Input Command)	144
▲ (Output Command)	144
: (Multi-statement Command)	144
↵ (Carriage Return)	144
⇒ (Jump Code)	152
=, ≠, >, <, ≥, ≤ (Relational Operators)	155

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.
2. 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:

$$\text{If } \begin{array}{l} \langle \text{condition} \rangle \\ \text{numeric expression} \end{array} \left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \text{ Then } \langle \text{statement} \rangle \left[\left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right]$$

Parameters: condition, numeric expression

Description:

1. The Then-statement is executed only when the If-condition is true (non-zero).
2. If the condition is false (0), the Then-statement is not executed.
3. 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 ↵
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:

$$\text{If } \begin{array}{l} \langle \text{condition} \rangle \\ \text{numeric expression} \end{array} \left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \text{ Then } \langle \text{statement} \rangle$$

$$\left[\left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right] \left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \text{ IfEnd}$$

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 ↵
Then "A = 0" ↵
IfEnd ↵
"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:

$$\text{If } \begin{array}{l} \langle \text{condition} \rangle \\ \text{numeric expression} \end{array} \left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \text{ Then } \langle \text{statement} \rangle \left[\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right]$$

$$\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \text{ Else } \langle \text{statement} \rangle \left[\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right]$$

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 \downarrow
 Then "TRUE" \downarrow
 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:

$$\text{If } \begin{array}{l} \langle \text{condition} \rangle \\ \text{numeric expression} \end{array} \left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \text{ Then } \langle \text{statement} \rangle \left[\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right]$$

$$\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \text{ Else } \langle \text{statement} \rangle \left[\left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \langle \text{statement} \rangle \right] \left\{ \begin{array}{l} \downarrow \\ : \\ \uparrow \end{array} \right\} \text{ IfEnd}$$

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: ? \rightarrow A \downarrow
 If A = 0 \downarrow
 Then "TRUE" \downarrow
 Else "FALSE" \downarrow
 IfEnd \downarrow
 "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:

$$\text{For } \langle \text{starting value} \rangle \rightarrow \langle \text{control variable name} \rangle \text{ To } \langle \text{ending value} \rangle \left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\}$$

$$\left[\langle \text{statement} \rangle \left\{ \begin{array}{c} \downarrow \\ \vdots \\ \uparrow \end{array} \right\} \right] \text{ Next}$$
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:

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

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.

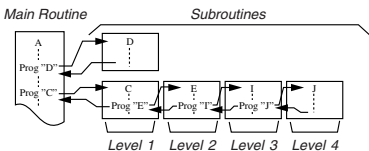
Syntax: Prog "file name" ↵

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.

- A subroutine can be used in multiple locations in the same main routine, or it can be called up by any number of main routines.



- Calling up a subroutine causes it to be executed from the beginning. After execution of the subroutine is complete, execution returns to the main routine, continuing from the statement following the Prog command.
- A Goto-Lbl command inside of a subroutine is valid inside of that subroutine only. It cannot be used to jump to a label outside of the subroutine.
- If a subroutine with the file name specified by the Prog command does not exist, an error (Go ERROR) occurs.
- In the **RUN Mode**, inputting the Prog command and pressing **EXE** launches the program specified by the command.

Return

Function: This command returns from a subroutine.

Syntax: Return ↵

Description:

Execution of the Return command inside a main routine causes execution of the program to stop.

Example:

Prog "A"	Prog "B"
1 → A ↵	For A → B To 10 ↵
Prog "B" ↵	B + 1 → C ↵
C ▲	Next ↵
	Return

Executing the program in File A displays the result of the operation (11).

Stop

Function: This command terminates execution of a program.

Syntax: Stop ↵

Description:

- This command terminates program execution.
- Execution of this command inside of a loop terminates program execution without an error being generated.

```

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

```

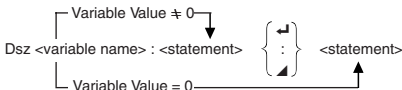
This program counts from 2 to 10. When the count reaches 5, however, it terminates execution and displays the message "STOP."

■ 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.

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

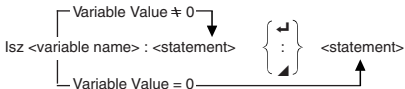
Example: ? → A : ? → B : Lbl 1 :
 ? → X : A × X + B ▲
 Goto 1

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.

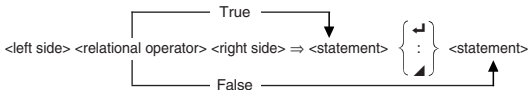
Description:

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 (\blacktriangle), or carriage return (\blacktriangledown).

Example: Lbl 1 : ? \rightarrow A :

A \geq 0 \Rightarrow $\sqrt{\square}$ A \blacktriangle

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.

■ Input/Output Commands (I/O)

Receive (

Function: This command receives data from an external device.

Syntax: Receive (<data>) (...ex. Receive (List 1))

Description:

1. This command receives data from an external device.
2. The following types of data can be received by this command.
 - Individual values assigned to variables
 - List data (all values - individual values cannot be specified)

Send (

Function: This command sends data to an external device.

Syntax: Send (<data>) (...ex. Send (List 1))

Description:

1. This command sends data to an external device.
2. The following types of data can be sent by this command.
 - Individual values assigned to variables
 - List data (all values - individual values cannot be specified)

■ Conditional Jump Relational Operators (REL)

=, ≠, >, <, ≥, ≤

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

Syntax:

$$\langle \text{left side} \rangle \langle \text{relational operator} \rangle \langle \text{right side} \rangle \Rightarrow \langle \text{statement} \rangle \left\{ \begin{array}{c} \leftarrow \\ : \\ \blacktriangleleft \end{array} \right\} \langle \text{statement} \rangle$$

(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

$\langle \text{left side} \rangle = \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ equals $\langle \text{right side} \rangle$

$\langle \text{left side} \rangle \neq \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ does not equal $\langle \text{right side} \rangle$

$\langle \text{left side} \rangle > \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ is greater than $\langle \text{right side} \rangle$

$\langle \text{left side} \rangle < \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ is less than $\langle \text{right side} \rangle$

$\langle \text{left side} \rangle \geq \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ is greater than or equal to $\langle \text{right side} \rangle$

$\langle \text{left side} \rangle \leq \langle \text{right side} \rangle$: true when $\langle \text{left side} \rangle$ is less than or equal to $\langle \text{right side} \rangle$

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



11. 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 (\blacktriangleleft) or multi-statement 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.

12. 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 \blacktriangleleft
- Graph function input
Y = Type \blacktriangleleft Specifies graph type.
"X² - 3" → Y1 \blacktriangleleft
- Graph draw operation
DrawGraph \blacktriangleleft

Example Program

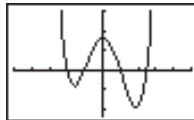
- ① ClrGraph \blacktriangleleft
- ② View Window -10, 10, 2, -120, 150, 50 \blacktriangleleft
- ③ Y = Type \blacktriangleleft
"X ^ 4 - X ^ 3 - 24X² + 4X + 80" → Y1 \blacktriangleleft
- ④ G SelOn 1 \blacktriangleleft
- ⑤ DrawGraph

- ① SHIFT PROG > F3 F2
- ② SHIFT F3 F1 QUIT
- ③ F3 F3 F2 F1 QUIT
- ④ VARS > F2 F1 QUIT
- ⑤ F3 F3 F1 F1
- ⑥ SHIFT PROG > F4 F2



P.48

Executing this program produces the result shown here.



P.74

■ 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 → F Start ↵
 - 5 → F End ↵
 - 1 → F pitch ↵
- Numeric table generation
 - DispTable ↵
- Graph draw operation
 - Connect type: DrawTG-Con ↵
 - Plot type: DrawTG-Plt ↵

Example Program

```

ClrGraph ↵
ClrText ↵
View Window 0, 6, 1, -2, 106, 20 ↵
Y = Type ↵
"3X2 - 2" → Y1 ↵
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 VARS > F3 F1

③ F2

④ F3 QUIT

⑤ SHIFT PRGM > F4 F3 F1 QUIT

⑥ SHIFT PRGM > F4 F3 F2 QUIT

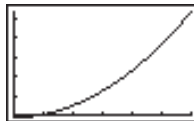
Executing this program produces the results shown here.

Numeric Table

X	Y1
0	-2
1	1

Graph

EXE



P.82

■ 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.96

■ 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 pie chart.

S-Gph1 DrawOn, Pie, List1, % (data display format) ↵

- The following is a typical graph condition specification for a stacked bar chart, bar graph, or line graph.

Stacked bar chart: S-Gph1 DrawOn, StackedBar, List1 ↵

Bar graph: S-Gph1 DrawOn, Bar, List1 ↵

Line graph: S-Gph1 DrawOn, LineG, List1 ↵

- The following is a typical graph condition specification for a superimposed bar and line graph.

S-Gph1 DrawOn, Both, List1 (bar graph list), List2 (line graph list),
Sep. G(AutoWin setting) ↵

- 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

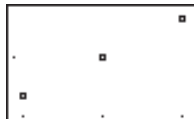
Exponential Regression: Exp

Power Regression : Power

Example Program

- | | |
|---|---|
| ClrGraph \downarrow | ① SHIFT SETUP ▷ ▷ F1 QUIT |
| ① S-WindAuto \downarrow | ② OPTN F1 F1 |
| {1, 2, 3} \rightarrow ② List 1 \downarrow | ③ F1 QUIT |
| {1, 2, 3} \rightarrow ③ List 2 \downarrow | ④ F3 F1 F2 F1 QUIT |
| ④ S-Gph1 ⑤ DrawOn, | ⑤ F3 F1 F1 F1 QUIT |
| ⑥ Scatter, List1, List2, 1, ⑦ Square \downarrow | ⑥ F3 F1 F2 ▷ F1 QUIT |
| ⑧ DrawStat | ⑦ F3 F1 F4 F1 QUIT |
| | ⑧ SHIFT PRGM ▷ F4 F1 QUIT |

Executing this program produces the scatter diagram shown here.

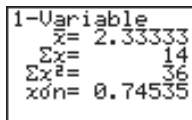


■ Performing Statistical Calculations

- Single-variable statistical calculation

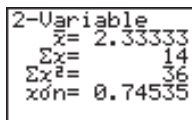
- ① 1-Variable List 1, List 2
- Frequency data (Frequency)
 - x -axis data (XList)

- ① **F3** **F1** **▷** **F1** **F1** **QUIT**

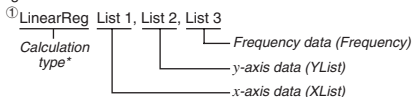


- Paired-variable statistical calculation

- 2-Variable List 1, List 2, List 3
- Frequency data (Frequency)
 - y -axis data (YList)
 - x -axis data (XList)



• 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

