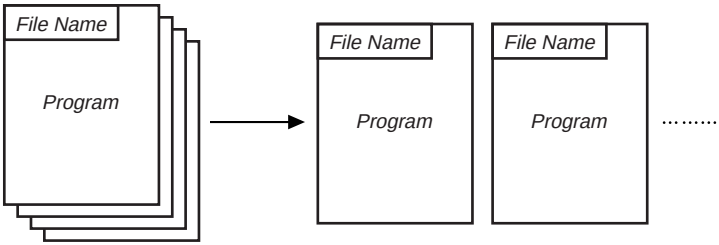
Chapter 21

Programming

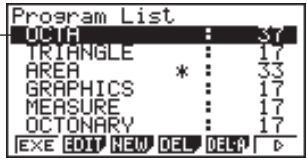
21-1 Before Programming

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)



- {**EXE**}/{**EDIT**} ... program {execute}/{edit}
- {**NEW**} ... {new program}
- {**DEL**}/{**DEL·A**} ... {specific program}/{all program} delete
- {**SRC**}/{**REN**} ... file name {search}/{change}

P.382
P.376

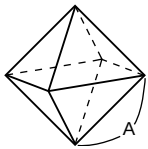
- If there are not programs stored in memory when you enter the PRGM Mode, the message "**No Programs**" appears on the display and only the NEW item (F3) is shown in the function menu.

  The values to the right of the program list indicate the number of bytes used up by each program.

21-2 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. Display the program list menu and press [F3] (NEW) to display a menu, which contains the following items.

P.374

- {**RUN**}/{**BASE**} ... {general calculation}/{number base} program input
- {**π0**} ... {password registration}
- {**SYBL**} ... {symbol menu}

2. Input the name of the file.

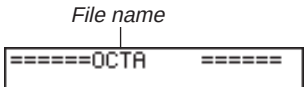
[O] [C] [T] [A]

```
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, *r*, *θ*, spaces, [,], {, }, ', ", ~, 0 through 9, ., +, –, ×, ÷
- Note, however, that [X,θ,T] and [•] cannot be input for the name of a program that contains binary, octal, decimal, or hexadecimal calculations.

- Use [F1] (RUN) to input a program for general calculations (a program to be executed in the COMP Mode). For programs that involve number system specifications, use [F2] (BASE). Note that programs input after pressing [F2] (BASE) are indicated by **B** to the right of the file name.
- Pressing [F6] (SYBL) displays a menu of symbols (', ", ~) that can be input.
- You can delete a character while inputting a file name by moving the cursor to the character you want to delete and pressing [DEL].

3. Press [EXE] to register the file name and change to the program input screen.



- 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 [EXIT].
- When you register the name of a program that contains binary, octal, decimal, or hexadecimal calculations, the indicator **B** is appended to the right of the file name.

●To input a program

P.379

The following items are included in the function menu of the program input screen, which is used for program input.

P.378

- {**TOP**}/{**BTM**} ... {top}/{bottom} of program
- {**SRC**} ... {search}
- {**MENU**} ... {mode menu}
- {**SYBL**} ... {symbol menu}

●To change modes in a program

- Pressing [F4] (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.

P.3

- {**STAT**}/{**MAT**}/{**LIST**}/{**GRPH**}/{**DYNA**}/{**TABL**}/{**RECR**}

  For details on each of these modes, see “To select an icon”, as well as the sections of this manual that describe what you can do in each mode.

- The following menu appears whenever you press [F4] (MENU) while inputting a program that involves number base specifications.
- {**d ~ o**}/{**LOG**}

- Pressing [F6] (SYBL) displays a menu of symbols (', ", ~, ∗, /, #) that can be input into a program.

- Pressing [SHIFT] [SET UP] displays a menu of commands that can be used to change set up screen settings inside a program.

- {**ANGL**}/{**COOR**}/{**GRID**}/{**AXES**}/{**LABL**}/{**DISP**}/{**P/L**}/{**DRAW**}/{**DERV**}/{**BACK**}/{**FUNC**}/{**SIML**}/{**S-WIN**}/{**LIST**}/{**LOCS**}/{**T-VAR**}/{Σ**DSP**}/{**RESID**}

P.5

For details on each of these commands, see “Set Up Screen Function Key Menus”.

The following function key menu appears if you press [SHIFT] [SET UP] while inputting a program that contains binary, octal, decimal, or hexadecimal calculation.

- {**Dec**}/{**Hex**}/{**Bin**}/{**Oct**}

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] [×] [SHIFT] [√] [3] [×] <value of A> [x^2] [EXE]
Volume V ..... [SHIFT] [√] [2] [÷] [3] [×] <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] [×] [SHIFT] [√] [3] [×] [ALPHA] [A] [x^2] [EXE]
Volume V ..... [SHIFT] [√] [2] [÷] [3] [×] [ALPHA] [A] [^] [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: ? → <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.

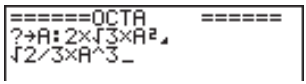
P.383

- 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] [F4](?) [→] [ALPHA] [A] [F6](▷) [F5](:)
[2] [×] [SHIFT] [√] [3] [×] [ALPHA] [A] [x^2]
[F6](▷) [F5](◢)

[SHIFT] [√] [2] [÷] [3] [×] [ALPHA] [A] [^] [3]



[SHIFT] [QUIT] or [EXIT] [EXIT]

●To run a program

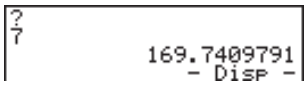
1. While the program list is on the display, use ▲ and ▼ to highlight the name of the program you want to run.
2. Press [F1] (EXE) or [EXE] to run the program.

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

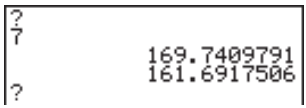
[F1] (EXE) or [EXE]

[7] [EXE]
(Value of A)

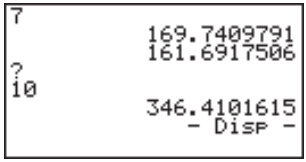


Intermediate result produced by ◢

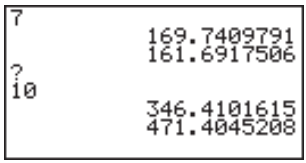
[EXE] [EXE]



[1] [0] [EXE]



[EXE]



P.392

- 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 occurs if the program specified by Prog "<file name>" cannot be found.

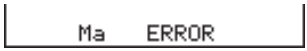
21-3 Debugging a Program

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.



P.450

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.

P.374

- 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

P.379

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

21-4 Calculating the Number of Bytes Used by a Program

This unit comes with 60 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.24

- You can check how much memory has been used and how much remains at any time by selecting the **SYS Mode** and entering the memory status screen. See “Memory Status” for details.

21-5 Secret Function

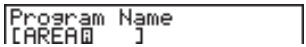
When inputting a program, you can protect it with a password that limits access to the program contents to those who know the password. Password protected programs can be executed by anyone without inputting the password.

●To register a password

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

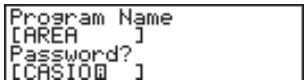
1. While the program list is on the display, press [F3] (NEW) and input the file name of the new program file.

   [F3](NEW)
   [A] [R] [E] [A]



2. Press [F5] (π0) and then input the password.

   [F5](π0)
   [C] [A] [S] [I] [O]



P.367

- 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.

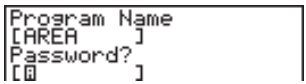
4. After inputting the program, press [SHIFT] [QUIT] to exit the program file and return to the program list. Files that are password protected are indicated by an asterisk to the right of the file name.

●To recall a program

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

1. In the program list, use ▲ and ▼ to move the highlighting to the name of the program you want to recall.

2. Press F2 (EDIT).



3. Input the password and press EXE to recall the program.
   - The message "**Mismatch**" appears if you input the wrong password.

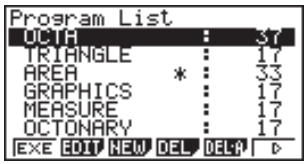
21-6 Searching for a File

There are three different methods for searching for a specific file name.

●To find a file using scroll search

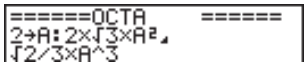
Example **To use scroll search to recall the program named OCTA**

1. While the program list is on the display, use ▲ and ▼ to scroll through the list of program names until you find the one you want.



F2

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



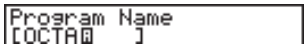
●To find a file using file name search

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

P.374

1. While the program list is on the display, press F3 (NEW) and input the name of the file you want to find.
   - If the file you are looking for is password protected, you should also input the password.

F3(NEW)
O C T A



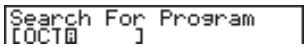
2. Press EXE to recall the program.
   - If there is no program whose file name matches the one you input, a new file is created using the input name.

●To find a file using initial character search

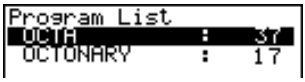
Example **To use initial character search to recall the program named OCTA**

1. While the program list is on the display, press F6 (▷) F1 (SRC) and input the initial characters of the file you want to find.

F6(▷)F1(SRC)
O C T



2. Press [EXE] 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 [EXIT] to clear the error message.

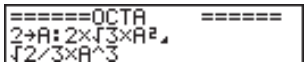
3. Use ▲ and ▼ to highlight the file name of the program you want to recall and then press [F2] (EDIT) to recall it.

21-7 Searching for Data Inside a Program

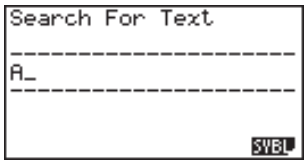
Example **To search for the letter “A” inside the program named OCTA**

1. Recall the program.

2. Press [F3] (SRC) and input the data you want to search for.



[F3](SRC)
[ALPHA] [A]

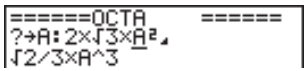


- You cannot specify the newline symbol (↵) or display command (◢) for the search data.

3. Press [EXE] to begin the search. The contents of the program appears on the screen with the cursor located at the first instance of the data you specified.

Indicates search operation is in progress

4. Press [EXE] to find the next instance of the data.



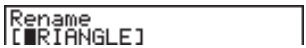
- If there is no match inside the program for the data you specified, the contents of the program appear with the cursor located at the point from which you started your search.
- Once the contents of the program are on the screen, you can use the cursor keys to move the cursor to another location before searching for the next instance of the data. Only the part of the program starting from the current cursor location is searched when you press [EXE].
- Once the search finds an instance of your data, inputting characters or moving the cursor causes the search operation to be cancelled (clearing the Search indicator from the display).
- If you make a mistake while inputting characters to search for, press [AC] to clear your input and re-input from the beginning.

21-8 Editing File Names and Program Contents

●To edit a file name

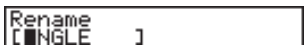
Example **To change the name of a file from TRIANGLE to ANGLE**

1. While the program list is on the display, use ▲ and ▼ to move the highlighting to the file whose name you want to edit and then press F6 (▷) F2 (REN).



2. Make any changes you want.

DEL DEL DEL



3. Press EXE to register the new name and return to the program list.
   - If the modifications you make result in a file name that is identical to the name of a program already stored in memory, the message "**Already Exists**" appears. When this happens, you can perform either of the following two operations to correct the situation.
   - Press ▶ or ◀ to clear the error and return to the file name input screen.
   - Press AC to clear the new file name and input a new one.

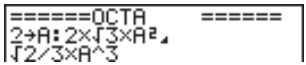
●To edit program contents

1. Find the file name of the program you want in the program list.
2. Recall the program.

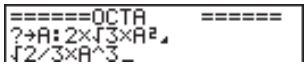
P.20

   - The procedures you use for editing program contents are identical to those used for editing manual calculations. For details, see "Editing Calculations".
   - The following function keys are also useful when editing program contents.

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



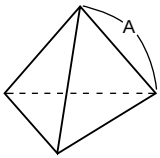
F2 (BTM) ..... Moves the cursor to the bottom of the program



P.367

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**

Use TETRA as the file name.



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] [F4](?) [→] [ALPHA] [A] [F6](▷) [F5](:)
Surface Area S ..... [SHIFT] [√] [3] [×] [ALPHA] [A] [x^2] [F6](▷) [F5](◢)
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] [F4](?) [→] [ALPHA] [A] [F6](▷) [F5](:)
Surface Area S ..... [2] [×] [SHIFT] [√] [3] [×] [ALPHA] [A] [x^2] [F6](▷) [F5](◢)
Volume V ..... [SHIFT] [√] [2] [÷] [3] [×] [ALPHA] [A] [^] [3]

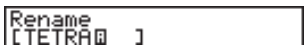
As you can see, you can produce the TETRA program by making the following changes in the OCTA program.

- Deleting [2] [×] (underlined using a wavy line above)
- Changing [3] to [1] [2] (underlined using a solid line above)

Let's edit OCTA to produce the TETRA program.

1. Edit the program name.

[F6](▷) [F2](REN) [T] [E] [T] [R] [A]



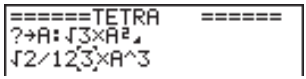
[EXE]

2. Edit the program contents.

[F2](EDIT)

[▶] [▶] [▶] [▶] [DEL] [DEL]

▼ ◀ SHIFT INS 1 2



DEL

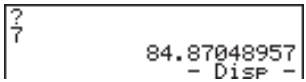
SHIFT QUIT

Let's try running the program.

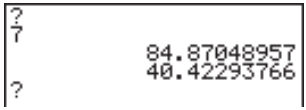
Length of One Side (A)	Surface Area (S)	Volume (V)
7 cm	84.87048957 cm^2	40.42293766 cm^3
10 cm	173.2050808 cm^2	117.8511302 cm^3
15 cm	389.7114317 cm^2	397.7475644 cm^3

F1 (EXE) or EXE

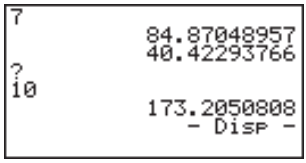
7 EXE
(Value of A)



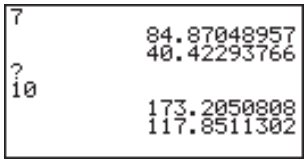
EXE EXE



1 0 EXE



EXE



21-9 Deleting a Program

There are two methods for deletion of a file name and its program.

●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 F4 (DEL).

3. Press F1 (YES) to delete the selected program or F6 (NO) to abort the operation without deleting anything.

●To delete all programs

1. While the program list is on the display, press F5 (DEL·A).

2. Press F1 (YES) to delete all the programs in the list or F6 (NO) to abort the operation without deleting anything.

P.26

- You can also delete all programs using the **SYS Mode**. See “Clearing Memory Contents” for details.

21-10 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.

- {**COM**}/{**CTL**}/{**JUMP**}/{**CLR**}/{**DISP**}/{**REL**}/{**I/O**}
- {**?**} ... {input command}
- {◢} ... {output command}
- {**:**} ... {multi-statement command}

■ COM (program command menu)

Selecting {COM} from the program menu displays the following function menu items.

- {**If**}/{**Then**}/{**Else**}/{**I-End**}/{**For**}/{**To**}/{**Step**}/{**Next**}/{**While**}/{**WEnd**}/{**Do**}/{**Lp-W**} ... {If}/{Then}/{Else}/{IfEnd}/{For}/{To}/{Step}/{Next}/{While}/{WhileEnd}/{Do}/{LpWhile} command

■ CTL (program control command menu)

Selecting {CTL} from the program menu displays the following function menu items.

- {**Prog**}/{**Rtrn**}/{**Brk**}/{**Stop**} ... {Prog}/{Return}/{Break}/{Stop} command

■ JUMP (jump command menu)

Selecting {JUMP} from the program menu displays the following function menu items.

- {**Lbl**}/{**Goto**} ... {Lbl}/{Goto} command
- {⇒} ... {jump command}
- {**Isz**}/{**Dsz**} ... {jump and increment}/{jump and decrement}

■ CLR (clear command menu)

Selecting {CLR} from the program menu displays the following function menu items.

- {**Text**}/{**Grph**}/{**List**} ... clears {text}/{graph}/{list}

■ DISP (display command menu)

Selecting {DISP} from the program menu displays the following function menu items.

- {**Stat**}/{**Grph**}/{**Dyna**} ... {statistical graph}/{graph}/{Dynamic Graph} draw

- {**F-Tbl**} ... {Table & Graph command menu}

The following are the items that appear in the above menu.

  - {**Tabl**}/{**G-Con**}/{**G-Plt**} ... {DispF-Tbl}/{DrawFTG-Con}/{DrawFTG-Plt} command

- {**R-Tbl**} ... {recursion calculation and recursion fomula}

The following are the items that appear in the above menu.

  - {**Tabl**}/{**Web**}/{**an-Cn**}/{Σ**a-Cn**}/{**an-Pl**}/{Σ**a-Pl**} ... {DispR-Tbl}/{DrawWeb}/{DrawR-Con}/{DrawRΣ-Con}/{DrawR-Plt}/{DrawRΣ-Plt} command

■ REL (conditional jump relational operator commands)

Selecting {REL} from the program menu displays the following function menu items.

- {**=**}/{≠}/{**>**}/{**<**}/{≥}/{≤} ... {=}/{≠}/{>}/{<}/{≥}/{≤} relational operators

■ I/O (input/output commands)

Selecting {I/O} from the program menu displays the following function menu items.

- {**Lcte**}/{**Gtky**}/{**Send**}/{**Recv**} ... {Locate}/{Getkey}/{Send()/{Receive()} command

- The appearance of the function menu differs slightly for a program that contains binary, octal, decimal, or hexadecimal calculation, but the functions in the menu are the same.

21-11 Command Reference

■ Command Index

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:

If <condition> (numeric expression) { ↵ : ◢ } Then <statement> [{ ↵ : ◢ } <statement>]

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.

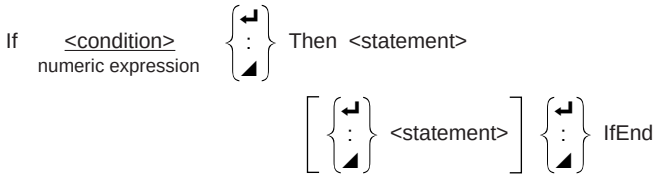
3. An If-condition must always be accompanied by a Then-statement. Omitting the Then-statement results in an error.

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:



Parameters: condition, numeric expression

Description:

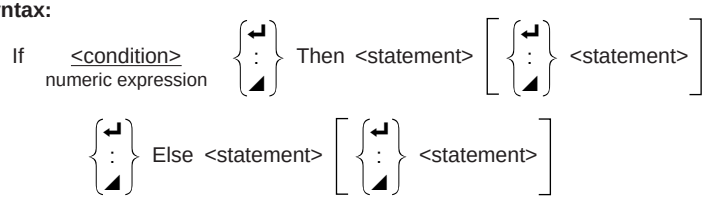
This command is almost identical to If~Then. The only difference is that the IfEnd-statement is always executed, regardless of whether the If-condition is true (non-zero) or false (0).

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:



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:

If <condition> (numeric expression) { ↵ : ◢ } Then <statement> [{ ↵ : ◢ } <statement>]

{ ↵ : ◢ } Else <statement> [{ ↵ : ◢ } I<statement>] { ↵ : ◢ } 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:
```
? → 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:

For <starting value> → <control variable name> To <ending value> { ↵ : ◢ }

[<statement> { ↵ : ◢ }] 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.

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:

For <starting value> → <control variable name> To <ending value> Step <step value> { ↵ / : / ◢ }

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:

Do { ↵ / : / ◢ } ~ LpWhile <expression>

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:

While <expression> { ↵ / : / ◢ } ~ 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" ↵

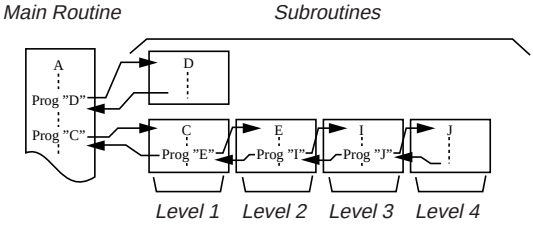
Example: Prog "ABC" ↵

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.

3. 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.



4. 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.

5. 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.

6. If a subroutine with the file name specified by the Prog command does not exist, an error occurs.

7. 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:

1. This command terminates program execution.

2. 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

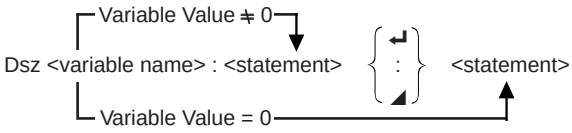
This program counts from 2 to 10. When the count reaches 5, however, it terminates execution and displays the message "STOP."

■ 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, r, θ

[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, r, θ)

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 occurs.

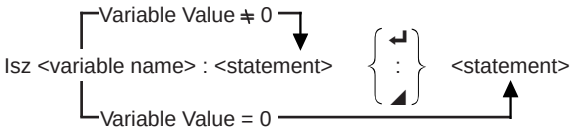
Example: ? → A : ? → B : Lbl 1 :
? → X : A × X + B ◢
Goto 1

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, r, θ

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

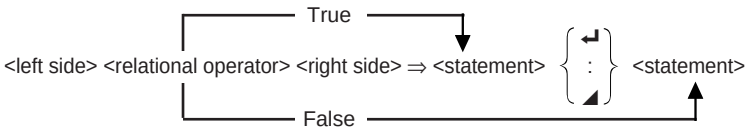
Description:

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:



P.401

Parameters:

left side/right side: variable (A to Z, r, θ), numeric constant, variable expression (such as: A × 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 ⇒ command. If the comparison returns a false result, execution jumps to the statements following the multi-statement command (:), display command (◢), or carriage return (↵).

Example:
```
Lbl 1 : ? → A :
A ≧ 0 ⇒ √ A ◢
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↵

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

ClrList

Function: This command clears list data.

Syntax: ClrList↵

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↵

Description: This command clears text from the screen during program execution.

■ Display Commands (DISP)

DispF-Tbl, DispR-Tbl

Function: These commands display numeric tables.

Syntax:

DispF-Tbl ↵

DispR-Tbl ↵

Description:

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

2. DispF-Tbl generates a function table, while DispR-Tbl generates a recursion table.

DrawDyna

Function: This command executes a Dynamic Graph draw operation.

Syntax: DrawDyna ↵

Description: This command performs a Dynamic Graph draw operation during program execution in accordance with the drawing conditions defined within the program.

DrawFTG-Con, DrawFTG-Plt

Function: These commands graph functions.

Syntax:

DrawFTG-Con ↵

DrawFTG-Plt ↵

Description:

1. These commands graph functions in accordance with conditions defined within the program.

2. DrawFTG-Con produces a connect type graph, while DrawFTG-Plt produces a plot type graph.

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.

DrawR-Con, DrawR-Plt

Function: These commands graph recursion expressions, with $a_n(b_n)$ as the vertical axis and n as the horizontal axis.

Syntax:

DrawR-Con ↵

DrawR-Plt ↵

Description:

1. These commands graph recursion expressions, with $a_n(b_n)$ as the vertical axis and n as the horizontal axis, in accordance with conditions defined within the program.

2. DrawR-Con produces a connect type graph, while DrawR-Plt produces a plot type graph.

DrawRΣ-Con, DrawRΣ-Plt

Function: These commands graph recursion expressions, with $\Sigma a_n(\Sigma b_n)$ as the vertical axis and n as the horizontal axis.

Syntax:

DrawRΣ-Con ↵

DrawRΣ-Plt ↵

Description:

1. These commands graph recursion expressions, with $\Sigma a_n(\Sigma b_n)$ as the vertical axis and n as the horizontal axis, in accordance with conditions defined within the program.

2. DrawRΣ-Con produces a connect type graph, while DrawRΣ-Plt produces a plot type graph.

DrawStat

Function: This draws a statistical graph.

Syntax:

DrawStat ↵

Description:

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

DrawWeb

Function: This command graphs convergence/divergence of a recursion expression (WEB graph).

Syntax: DrawWeb [name of recursion expression], [number of lines] ↵

Example: DrawWeb a_{n+1} (b_{n+1}), 5 ↵

Description:

1. This command graphs convergence/divergence of a recursion expression (WEB graph).

2. Omitting the number of lines specification automatically specifies the default value 30.

■ Input/Output Commands (I/O)

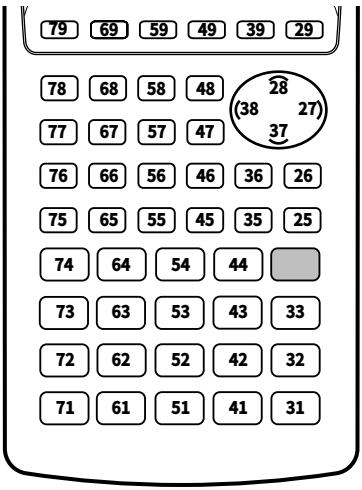
Getkey

Function: This command returns the code that corresponds to the last key pressed.

Syntax: Getkey ↵

Description:

1. This command returns the code that corresponds to the last key pressed.



2. A value of zero is returned if no key was pressed previous to executing this command.

3. This command can be used inside of a loop.

Locate

Function: This command displays alpha-numeric characters at a specific location on the text screen.

Syntax:

Locate <column number>, <line number>, <value>

Locate <column number>, <line number>, <variable name>

Locate <column number>, <line number>, "<string>"

[Example] Locate 1, 1, "AB"↵

Parameters:

- line number: number from 1 to 7
- column number: number from 1 to 21
- value: numeric value
- variable name: A to Z
- string: character string

Description:

1. This command displays values (including variable contents) or text at a specific location on the text screen.
2. The row is designated by a value from 1 to 7, which the column is designed by a value from 1 to 21.

Example: Cls↵
Locate 7, 1, "CASIO CFX"
This program displays the text "CASIO CFX" in the center of the screen.

- In some cases, the ClrText command should be executed before running the above program.

Receive (

Function: This command receives data from an external device.

Syntax: Receive (<data>)

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
   - Matrix data (all values - individual values cannot be specified)
   - List data (all values - individual values cannot be specified)
   - Picture data

Send (

Function: This command sends data to an external device.

Syntax: Send (<data>)

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
   - Matrix data (all values - individual values cannot be specified)
   - 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:

<left side> <relational operator> <right side> ⇒ <statement> { ↵ / : / ◢ } <statement>

Parameters:

left side/right side: variable (A to Z, r, θ), numeric constant, variable expression (such as: A × 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>

P.395

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

21-12 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 21 characters causes the text to move down to the next line. The screen scrolls automatically if the text causes the screen to become full.

21-13 Using Calculator Functions in Programs

P.80

■ Using Matrix Row Operations in a Program

These commands let you manipulate the rows of a matrix in a program.

- For this type of program, be sure to use the **MAT Mode** to input the matrix, and then switch to the **PRGM Mode** to input the program.

●To swap the contents of two rows (Swap)

Example 1 **To swap the values of Row 2 and Row 3 in the following matrix:**

$$\text{Matrix A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

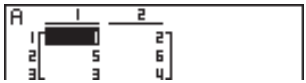
The following is the syntax to use for this program.

Swap A, 2, 3

Matrix name

Executing this program produces the following result.

(MAT Mode)



●To calculate a scalar product (∗Row)

Example 2 **To calculate the scalar product of Row 2 of the matrix in Example 1, multiplying by 4**

The following is the syntax to use for this program.

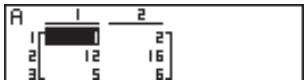
∗Row 4, A, 2

Matrix name

Multiplier

Executing this program produces the following result.

(MAT Mode)



●To calculate a scalar product and add the results to another row (∗Row+)

Example 3 **To calculate the scalar product of Row 2 of the matrix in Example 1, multiplying by 4, and add the result to row 3**

The following is the syntax to use for this program.

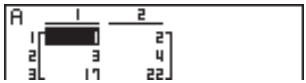
∗Row+ 4, A, 2, 3

Matrix name

Multiplier

Executing this program produces the following result.

(MAT Mode)



●To add two rows (Row+)

Example 4 **To add Row 2 to Row 3 of the matrix in Example 1**

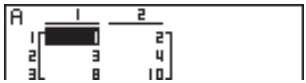
The following is the syntax to use for this program.

Row+ A, 2, 3

Matrix name

Executing this program produces the following result.

(MAT Mode)



P.112

■ 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 = Type ↵ ..... Specifies graph type.

"X^2 – 3" → Y1 ↵

• Graph draw operation

DrawGraph ↵

Example Program

① ClrGraph ↵

② View Window –10, 10, 2, –120, 150, 50 ↵

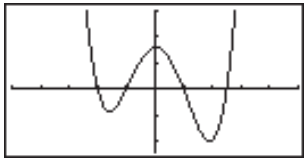
① SHIFT PRGM F6 F1 F2

② SHIFT F3 F1 EXIT

③Y = Type ↵
"X ^ 4–X ^ 3–24X² + 4X + 80" → Y1 ↵ (④)
⑤G SelOn 1 ↵
⑥Orange G1 ↵
⑦DrawGraph

③ F4 F4 F3 F1
④ VARS F4 F1 EXIT EXIT
⑤ F4 F4 F1 F1 EXIT
⑥ F4 F2
⑦ SHIFT PRGM F6 F2 F2

Executing this program produces the result shown here.



P.182

■ Using Dynamic Graph Functions in a Program

Using Dynamic Graph functions in a program makes it possible to perform repeat Dynamic Graph operations. The following shows how to specify the Dynamic Graph range inside a program.

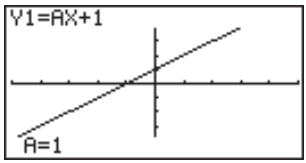
• Dynamic Graph range

1 → D Start ↵
5 → D End ↵
1 → D pitch ↵

Example Program

ClrGraph ↵
View Window –5, 5, 1, –5, 5, 1 ↵
Y = Type ↵
"AX + 1" → Y1 ↵ (①) — ① VARS F4 F1 EXIT EXIT
②D SelOn 1 ↵ — ② F4 F5 F1
③D Var A ↵ — ③ F3
1 → ④D Start ↵ — ④ VARS F5 F1
5 → ⑤D End ↵ — ⑤ F2
1 → ⑥D pitch ↵ — ⑥ F3
⑦DrawDyna — ⑦ SHIFT PRGM F6 F2 F3

Executing this program produces the result shown here.



⋮ ↑
↓ ⋮

P.206

■ 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

DispF-Tbl ↵

• Graph draw operation

Connect type: DrawFTG-Con ↵

Plot type: DrawFTG-Plt ↵

Example Program

ClrGraph ↵

ClrText ↵

View Window 0, 6, 1, –2, 106, 2 ↵

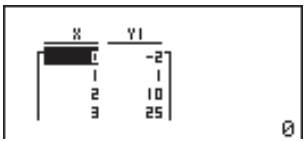
Y = Type ↵

"$3X^2 - 2$" → Y1 ↵

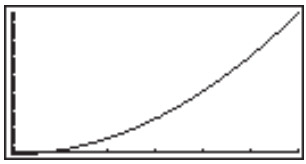
① T SelOn 1 ↵	① [F4] [F6] [F1] [F1]
0 → ② F Start ↵	② [VARS] [F6] [F1] [F1]
6 → ③ F End ↵	③ [F2]
1 → ④ F pitch ↵	④ [F3]
⑤ DispF-Tbl ◢	⑤ [SHIFT] [PRGM] [F6] [F2] [F4] [F1]
⑥ DrawFTG-Con	⑥ [SHIFT] [PRGM] [F6] [F2] [F4] [F2]

Executing this program produces the results shown here.

Numeric Table



Graph



P.218

■ Using Recursion Table & Graph Functions in a Program

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

• Recursion formula input

a_{n+1} Type ↵ .... Specifies recursion type.

"$3a_n + 2$" → a_{n+1} ↵

"$4b_n + 6$" → b_{n+1} ↵

• Table range setting

1 → R Start ↵

5 → R End ↵

1 → a_0 ↵

2 → b_0 ↵

1 → a_n Start ↵

3 → b_n Start ↵

• Numeric table generation

DispR-Tbl ↵

• Graph draw operation

Connect type: DrawR-Con ↵, DrawRΣ-Con ↵

Plot type: DrawR-Plt ↵, DrawRΣ-Plt ↵

• Statistical convergence/divergence graph (WEB graph)

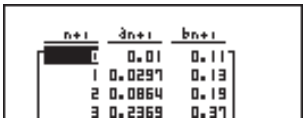
DrawWeb a_{n+1}, 10 ↵

Example Program

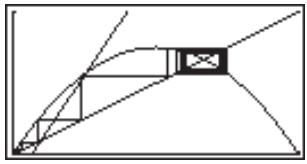
ClrGraph ↵	
View Window 0, 1, 1, 0, 1, 1 ↵	
① a_{n+1} Type ↵	① F4 F6 F2 F3 F2 EXIT
"$-3a_n^2 + 3a_n$" → a_{n+1} ↵	② F4 F2
"$3b_n - 0.2$" → b_{n+1} ↵	
0 → ③R Start ↵	③ VARS F6 F2 F2 F1
6 → R End ↵	
0.01 → a_0 ↵	
0.11 → b_0 ↵	
0.01 → a_n Start ↵	
0.11 → b_n Start ↵	
④DispR-Tbl ◢	④ SHIFT PRGM F6 F2 F5 F1
⑤DrawWeb ⑥a_{n+1}, 30	⑤ SHIFT PRGM F6 F2 F5 F2 EXIT EXIT EXIT ⑥ F4 F6 F2 F4 F3

Executing this program produces the results shown here.

Numeric Table



Recursion graph



P.231

■ Using List Sort Functions in a Program

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

• Ascending order

① ②
SortA (List 1, List 2, List 3)
Lists to be sorted (up to six can be specified)

① F4 F3 F1 EXIT ② OPTN F1 F1

• Descending order

SortD (List 1, List 2, List 3)
Lists to be sorted (up to six can be specified)

■ Using Solve Calculation Function in a Program

You can incorporate a solve calculation function into a program.

The following is the syntax for using the Solve function in a program.

Solve($f(x)$, n, a, b)
Upper limit
Lower limit
Initial estimated value

Example Program

① Solve($2X^2 + 7X - 9$, 1, 0, 1) ① OPTN F4 F1

- In the function $f(x)$, only X can be used as a variable in expressions. Other variables (A through Z, r, θ) are treated as constants, and the value currently assigned to that variable is applied during the calculation.
- Input of the closing parenthesis, lower limit a and upper limit b can be omitted.
- Solutions obtained using Solve may include errors.
- Note that you cannot use a Solve, differential, quadratic differential, integration, maximum/minimum value or Σ calculation expression inside of a Solve calculation term.

P.250

■ 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 “StatGraph”, 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
- Graph Color

P.252

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, Blue ↵

  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 normal probability plot.

  S-Gph1 DrawOn, NPPlot, List1, Square, Blue ↵

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

  S-Gph1 DrawOn, Hist, List1, List2, Blue ↵

  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.

P.254

  Histogram: ..... Hist

  Median Box: ..... MedBox

  Mean Box: ..... MeanBox

  Normal Distribution: ..... N-Dist

  Broken Line: ..... Broken

P.254

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

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

  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
  Cubic Regression: ..... Cubic
  Quartic Regression: ..... Quart
  Logarithmic Regression: .. Log
  Exponential Regression: Exp
  Power Regression: ..... Power

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

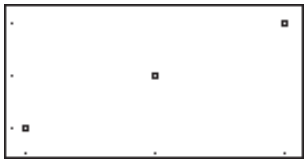
  S-Gph1 DrawOn, Sinusoidal, List1, List2, Blue ↵

Example Program

```
ClrGraph↵
①
S-Wind Auto↵
②
{1, 2, 3} → List 1↵
③
{1, 2, 3} → List 2↵
④ ⑤ ⑥ ⑦ ⑧
S-Gph1 DrawOn, Scatter, List1, List2, 1, Square, Blue ↵
⑨
DrawStat
```

① SHIFT SET UP F6 F6 F3 F1
② OPTN F1 F1
③ F1 EXIT EXIT
④ F4 F1 F2 F1 EXIT
⑤ F1 F1 EXIT
⑥ F2 F4 EXIT
⑦ EXIT F4 F1
⑧ EXIT F5 F1
⑨ SHIFT PRGM F6 F2 F1

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 F4 F1 F6 F1

```
1-Variable
x̄     =2.33333333
Σx    =14
Σx²   =36
xσn   =0.74535599
xσn-1 =0.81649658
n     =6
```

• Paired-variable statistical calculation

2-Variable List 1, List 2, List 3

Frequency data (Frequency)

y-axis data (YList)

x-axis data (XList)

```
2-Variable
 x̄   =2
 Σx  =6
 Σx² =14
 xσn =0.81649658
 xσn-1=1
 n   =3
```

• Regression statistical calculation

① LinearReg List 1, List 2, List 3

*Calculation type**

Frequency data (Frequency)

y-axis data (YList)

x-axis data (XList)

① F4 F1 F6 F6 F1

```
LinearReg
   a =1
   b =0
   r =1
   r²=1
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
CubicReg ..... cubic regression
QuartReg ..... quartic regression
LogReg ..... logarithmic regression
ExpReg ..... exponential regression
PowerReg ..... power regression

• Sine regression statistical calculation

SinReg List 1, List 2

y-axis data (YList)

x-axis data (XList)