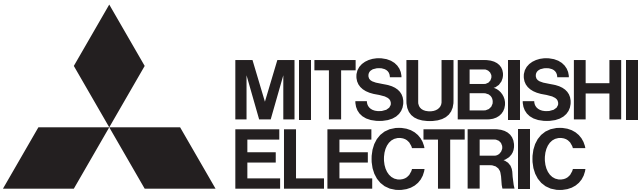


Programmable Controller

MELSEC iQ-R series MELSEC iQ-F series

MELSEC iQ-R/MELSEC iQ-F Structured Text (ST) Programming Guide Book

SAFETY PRECAUTIONS

(Read these precautions before using this product.)

Before using this product, please read this manual and the relevant manuals carefully and pay full attention to safety to handle the product correctly.

The precautions given in this manual are concerned with this product only. For the safety precautions of the programmable controller system, refer to the user's manual for the CPU module used.

In this manual, the safety precautions are classified into two levels: "⚠ WARNING" and "⚠ CAUTION".

⚠ **WARNING**	Indicates that incorrect handling may cause hazardous conditions, resulting in death or severe injury.
⚠ **CAUTION**	Indicates that incorrect handling may cause hazardous conditions, resulting in minor or moderate injury or property damage.

Under some circumstances, failure to observe the precautions given under "⚠ CAUTION" may lead to serious consequences.

Observe the precautions of both levels because they are important for personal and system safety.

Make sure that the end users read this manual and then keep the manual in a safe place for future reference.

[Design Precautions]

⚠ WARNING

- When data change, program change, or status control are performed from a personal computer to a running programmable controller, create an interlock circuit outside the programmable controller to ensure that the whole system always operates safely.

[Security Precautions]

⚠ WARNING

- To maintain the security (confidentiality, integrity, and availability) of the programmable controller and the system against unauthorized access, denial-of-service (DoS) attacks, computer viruses, and other cyberattacks from external devices via the network, take appropriate measures such as firewalls, virtual private networks (VPNs), and antivirus solutions.

CONDITIONS OF USE FOR THE PRODUCT

(1) MELSEC programmable controller ("the PRODUCT") shall be used in conditions;
i) where any problem, fault or failure occurring in the PRODUCT, if any, shall not lead to any major or serious accident; and
ii) where the backup and fail-safe function are systematically or automatically provided outside of the PRODUCT for the case of any problem, fault or failure occurring in the PRODUCT.

(2) The PRODUCT has been designed and manufactured for the purpose of being used in general industries.
MITSUBISHI ELECTRIC SHALL HAVE NO RESPONSIBILITY OR LIABILITY (INCLUDING, BUT NOT LIMITED TO ANY AND ALL RESPONSIBILITY OR LIABILITY BASED ON CONTRACT, WARRANTY, TORT, PRODUCT LIABILITY) FOR ANY INJURY OR DEATH TO PERSONS OR LOSS OR DAMAGE TO PROPERTY CAUSED BY the PRODUCT THAT ARE OPERATED OR USED IN APPLICATION NOT INTENDED OR EXCLUDED BY INSTRUCTIONS, PRECAUTIONS, OR WARNING CONTAINED IN MITSUBISHI ELECTRIC USER'S, INSTRUCTION AND/OR SAFETY MANUALS, TECHNICAL BULLETINS AND GUIDELINES FOR the PRODUCT.
("Prohibited Application")
Prohibited Applications include, but not limited to, the use of the PRODUCT in;
- Nuclear Power Plants and any other power plants operated by Power companies, and/or any other cases in which the public could be affected if any problem or fault occurs in the PRODUCT.
- Railway companies or Public service purposes, and/or any other cases in which establishment of a special quality assurance system is required by the Purchaser or End User.
- Aircraft or Aerospace, Medical applications, Train equipment, transport equipment such as Elevator and Escalator, Incineration and Fuel devices, Vehicles, Manned transportation, Equipment for Recreation and Amusement, and Safety devices, handling of Nuclear or Hazardous Materials or Chemicals, Mining and Drilling, and/or other applications where there is a significant risk of injury to the public or property.

Notwithstanding the above restrictions, Mitsubishi Electric may in its sole discretion, authorize use of the PRODUCT in one or more of the Prohibited Applications, provided that the usage of the PRODUCT is limited only for the specific applications agreed to by Mitsubishi Electric and provided further that no special quality assurance or fail-safe, redundant or other safety features which exceed the general specifications of the PRODUCTs are required. For details, please contact the Mitsubishi Electric representative in your region.

(3) Mitsubishi Electric shall have no responsibility or liability for any problems involving programmable controller trouble and system trouble caused by DoS attacks, unauthorized access, computer viruses, and other cyberattacks.

INTRODUCTION

Thank you for purchasing the Mitsubishi Electric MELSEC iQ-R series and MELSEC iQ-F series programmable controllers.
This manual describes the programming using Structured Text (ST) in GX Works3.
Before using this product, please read this manual and the relevant manuals carefully and develop familiarity with the functions and performance of the MELSEC iQ-R series and MELSEC iQ-F series programmable controllers to handle the product correctly.
When applying the program and circuit examples provided in this manual to an actual system, ensure the applicability and confirm that it will not cause system control problems.
Description in this manual is based on the use of MELSEC iQ-R series.
For the considerations when using MELSEC iQ-F series, refer to the following:
☞ Page 104 Considerations for Using the MELSEC iQ-F Series

CONTENTS

RELEVANT MANUALS

Manual name [manual number]	Description	Available form
MELSEC iQ-R/MELSEC iQ-F Structured Text (ST) Programming Guide Book [SH-081483ENG](this manual)	Programming using Structured Text (ST) in GX Works3. Fundamental operations and functions are explained using sample programs	Print book
		e-Manual PDF
GX Works2 Beginner's Manual (Structured Project) [SH-080788ENG]	Programming using Structured Text (ST) in GX Works2	Print book
		PDF
Structured Text (ST) Programming Guide Book [SH-080368E]	Programming using Structured Text (ST) in GX Developer	Print book
		PDF

Detailed specifications of Structured Text language are not described in this manual.

For details, refer to the following:

MELSEC iQ-R Programming Manual (Program Design)

MELSEC iQ-F FX5 Programming Manual (Program Design)

Point

e-Manual refers to the Mitsubishi Electric FA electronic book manuals that can be browsed using a dedicated tool.

e-Manual has the following features:

- Required information can be cross-searched in multiple manuals.
- Other manuals can be accessed from the links in the manual.
- The hardware specifications of each part can be found from the product figures.
- Pages that users often browse can be bookmarked.

TERMS

Unless otherwise specified, this manual uses the following terms.

Term	Description
Device	A variable of which name, type, and usage are defined by system.
Engineering tool	A tool for setting, programming, debugging, and maintaining programmable controllers.
Execution program	A program which has been converted. This program can be executed in a CPU module.
Function	A function that can be used as a POU. This function always outputs same result for the same input.
Function block	A function that can be used as a POU. This function retains values in the internal variables. It can be used by creating an instance.
Instance	An entity of a function block of which devices are assigned to the defined internal variables to be processed and executed. One or more instances can be created for one function block.
Label	A variable that is defined by user.
POU	A unit that configures a program. Units are categorized and provided in accordance with functions. Use of POUs enables dividing the lower-layer processing in a hierarchical program into some units in accordance with processing or functions, and creating programs for each unit.

GENERIC TERMS AND ABBREVIATIONS

Unless otherwise specified, this manual uses the following generic terms and abbreviations.

Generic term/abbreviation	Description
FB	An abbreviation for function block.
FUN	An abbreviation for function.
GX Developer	The product name of the software package for the MELSEC programmable controllers. A generic product name for SW8D5C-GPPW.
GX Works2	The product name of the software package for the MELSEC programmable controllers. A generic product name for SWnDND-GXW2 and SWnDNC-GXW2. ('n' indicates version.)
GX Works3	The product name of the software package for the MELSEC programmable controllers. A generic product name for SWnDND-GXW3. ('n' indicates version.)
LD	An abbreviation for Ladder Diagram.
ST	An abbreviation for Structured Text.

PART 1 ST PROGRAMMING

This part explains the ST programming with the MELSEC-iQ-R series and MELSEC iQ-F series.

1 WHAT IS STRUCTURED TEXT?

1.1 International Standard IEC 61131-3

IEC 61131-3 is the international standard for the PLC (Programmable Logic Controller) system, which is established by one of the international standard association, IEC (International Electrotechinical Commission).
In IEC 61131-3, five programming languages (IL/LD/ST/FBD/SFC) are regulated, and the guidelines for creating programs are described.

1.2 Features of Structured Text Language

Structured Text language is a description method for programming in text format. This language can be described in the same manner as a programming language for a personal computer such as C language. By using this language, the visibility of a program can be improved because the operation and data processing can be described simply.

Operation using mathematical expression

Arithmetic operations and comparison operations can be described in the same manner as the generic expression.

■Multiple operations can be described in one line

Since the operators (such as + and -) can be used, a program can be read much more easily than a ladder program.

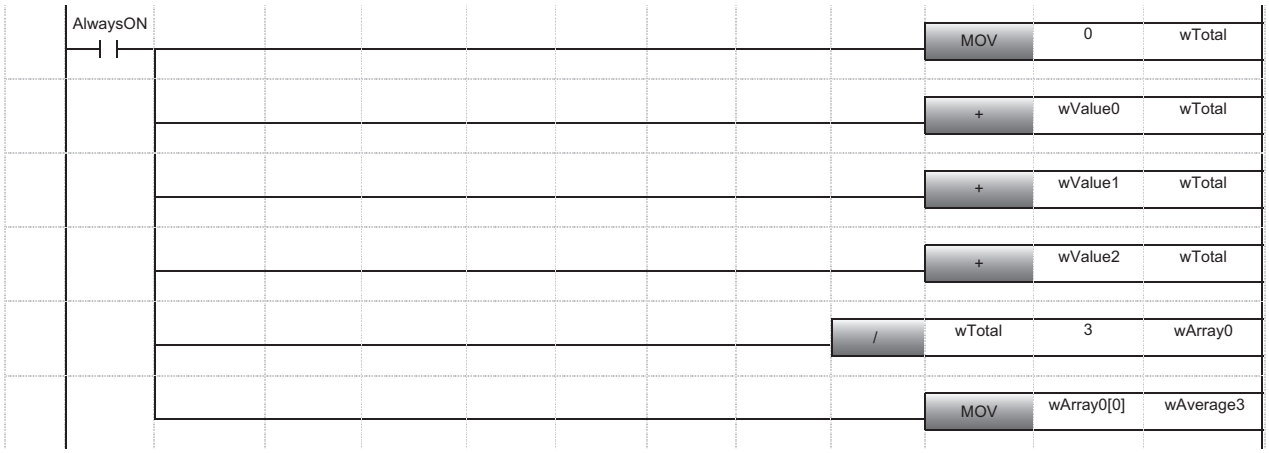
Program example

Assign the average from the wValue0 to the wValue2 in the wAverage3.
wAverage3 = (wValue0 + wValue1 + wValue2) ÷ 3

ST

```
wAverage3 := ( wValue0 + wValue1 + wValue2) / 3;
```

LD



Point

For details of the description method using Structured Text language, refer to the following:
☞ Page 18 Operational Expressions

■Description of expression regardless of data type

The description method for the expression of which value of decimal place is required to be considered is the same as the ordinary expression. The complex operations for real number type can be described simply.

Program example

Scaling analog data.

Scaling value = (After conversion.Upper limit - After conversion.Lower limit) ÷ (Before conversion.Upper limit - Before conversion.Lower limit) × (Measured value - Before conversion.Lower limit) + After conversion.Lower limit

ST

```
eScalingValue := ( stAfter.eUpperLimit - stAfter.eLowerLimit ) / (stBefore.eUpperLimit - stBefore.eLowerLimit) * (eMeasurements - stBefore.eLowerLimit) + stAfter.eLowerLimit;
```

Point

For details of the description method using Structured Text language, refer to the following:

☞ Page 29 Integer and Real Number

Complex information processing

By using a selection statement or an iteration statement, a complex processing of which execution contents branches depending on the condition, and the processing to be repeated can be described more easily than a ladder program.

Program example

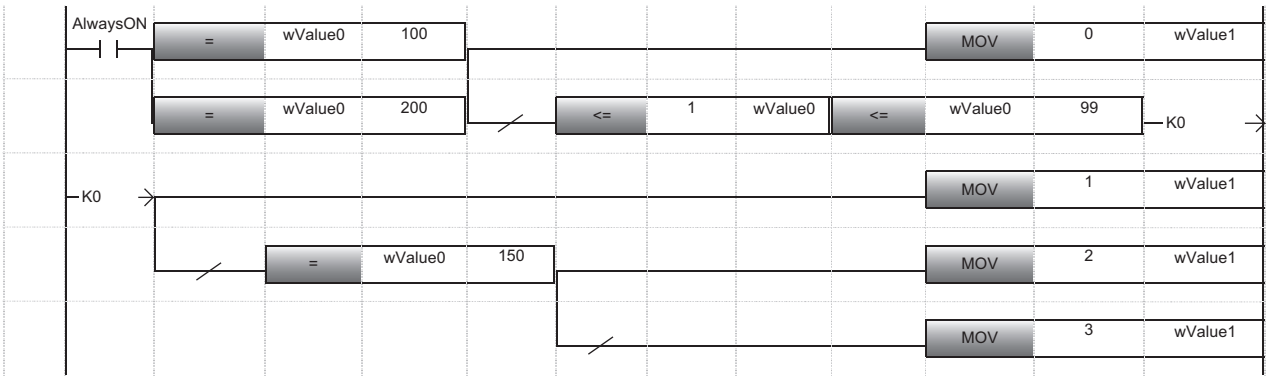
Set 0 to 3 to the wValue1 depending on the value of the wValue0.

- 100 or 200: 0
- 1 to 99: 1
- 150: 2
- Other than above: 3

ST

```
CASE wValue0 OF
    100, 200: wValue1 := 0;
    1..99: wValue1 := 1;
    150: wValue1 := 2;
    ELSE wValue1 := 3;
END_CASE;
```

LD



Point

For details of the description method using Structured Text language, refer to the following:

☞ Page 22 Selection, Page 25 Iteration

1.3 Proper Use for Programming Languages

The features of the five programming languages described in IEC 61131-3 are as follows:

Programming language		Features
IL	Instruction list	This language is suitable for high-speed processing or when CPU has a memory limitation. A compact program can be described.
LD	Ladder diagram	This language is suitable for simple relay sequence processing.
ST	Structured text	This language is suitable for operation using mathematical expressions or complex information processing. This language is familiar for engineers who are accustomed to a programming language for a personal computer.
FBD	Function block diagram	This language is suitable for continuous analog signal processing.
SFC	Sequential function chart	This language is suitable for step sequence based on the state transition.

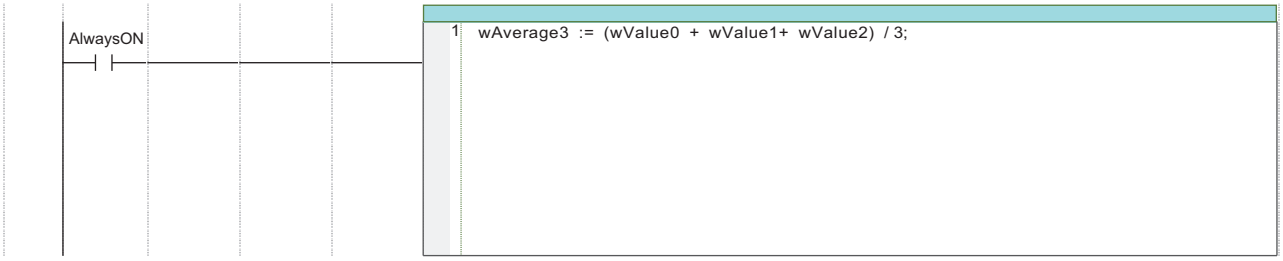
To take full advantage of the features of each language, create a program by combining each language in accordance with the process.

In GX Works3, the combination of the Structured Text and other languages can be used in the following functions.

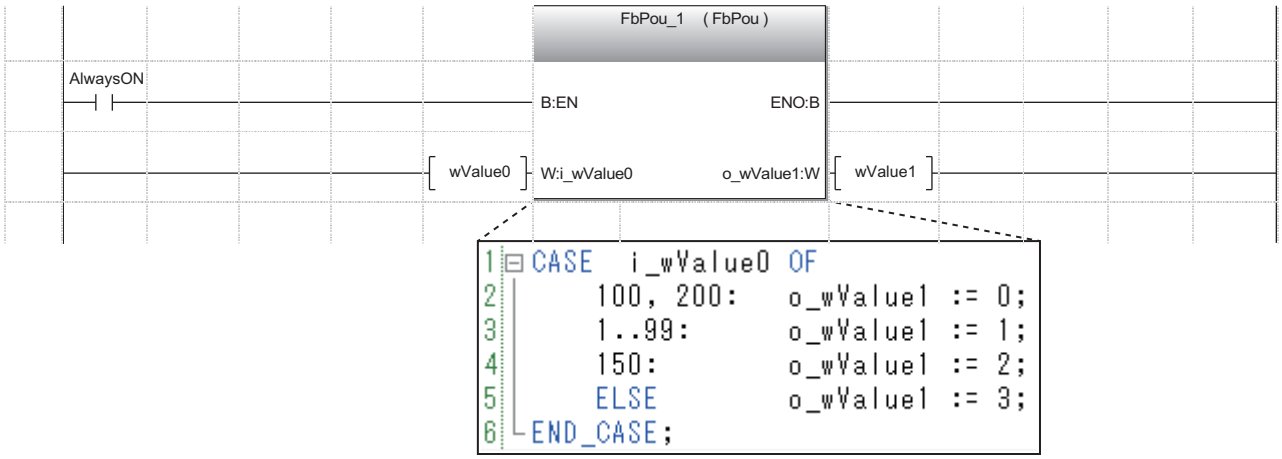
Function	Language	Description
Inline Structured Text	LD	Describe the part of a relay sequence processing using Structured Text.
FUN/FB	LD FBD	Define a sub program (subroutine) described in Structured Text as a POU, and call it from other programs such as LD.

By describing a simple relay sequence process and perspicuous process using the Ladder Diagram, and describing a part of complex process using the Structured Text of which process can be segmented, an easy-to-see program can be created.

LD with inline structured text



LD with FUN/FB



2 BASIC RULES FOR DESCRIPTION

This chapter explains the basic rules for describing programs in the Structured Text language.

2.1 Characters

The Structured Text is a programming language that can be described in text format. This section explains the characters and symbols that can be used in the Structured Text language.

Character code

GX Works3 supports the characters in the Unicode Basic Multilingual Plane (UTF-16).
Basic characters and symbols in multiple languages such as Japanese, English, and Chinese can be used for not only comments but also label names and data names.

Restriction

For the characters (reserved words) that cannot be used for a label name and data name, refer to the following:
GX Works3 Operating Manual

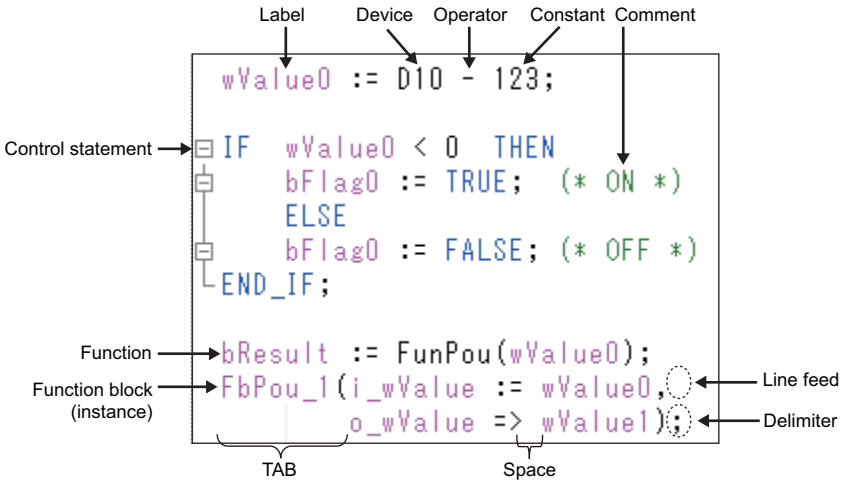
Basic component (Token)

A term or symbol that configures the program is referred to as a token.
In the Structured Text language describe the program using the following tokens.

Type		Example	Reference
Operator		+, -, <, >, =, &, NOT	Page 93 Operator
Keyword of control statement (defined standard identifier)		IF, CASE, WHILE, RETURN	Page 92 Statement
Identifier	Variable (Labels, devices, etc.)	X0, Y10, M100, D10, ZR0, bSwitch_A (arbitrary name)	Page 94 Device Page 96 Label
	POU (Function, function block)	BMOV, FunPou (arbitrary name), COUNTER_FB_M_1, FbPou_1 (arbitrary name)	Page 99 Function and function block
Constant		123, 'Character string', TRUE, FALSE	Page 97 Constant
Break character		;, (,)	—

A blank, line feed, and comment can be inserted among tokens.

Type	Example	Reference
Blank	Space, TAB	—
Line feed	Line feed	—
Comment	(**), /**/, //	Page 93 Comment



2.2 Instructions and Functions

The instructions which can be used in a ladder program is treated as functions in the Structured Text language.
The instructions and functions can be used in a similar format as function call of C language.

In GX Works3, the following functions and function blocks can be used.

Type	Instruction and function that corresponds to MELSEC-Q/L series	
	GX Works2	GX Developer
CPU module instruction Module dedicated instruction	Common instruction	MELSEC function
Standard function Standard function block	Application function	IEC function
Module FB	MELSOFT Library	—
MELSOFT Library (sample libraries)		—
Function and function block that are created by user	—	—

Restriction

The instructions which are not necessary for Structure Text language (such as contact) are not supported.
☞ Page 101 Instructions That Cannot be Used in ST Programs

Point

For details of the instructions, refer to the following:
📖MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks)
📖MELSEC iQ-R Programming Manual (Module Dedicated Instructions)
📖MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
📖 FB reference of each module

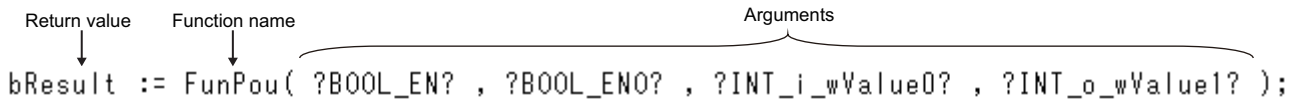
Function and function block

Function and function block are the POUs in which a subroutine to be called in a program is defined.

■Function

Function is a POU that outputs same result for the same input. Function is suitable for segmenting a simple and independent processing.

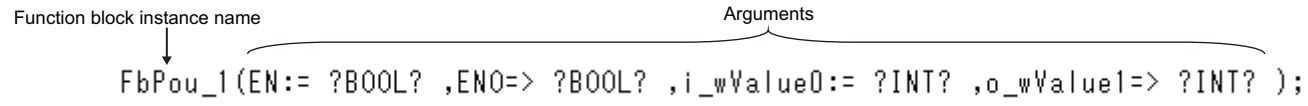
Describe the function name and arguments in an ST program as follows:



■Function block

Function block is a POU of which internal values can be used for operations. The different result is output for the same input depending on the value retained in each instance (entity). Function block is suitable for segmenting much more complex processing than a function, or for processing which is required to be executed repeatedly using the retained value.

To use a function block in an ST program, describe an instance and arguments as follows;



Point

For details of the description method using Structured Text language, refer to the following:

☞ Page 99 Function and function block

For details of the functions and function blocks, refer to the following:

📖MELSEC iQ-R Programming Manual (Program Design)

📖MELSEC iQ-F FX5 Programming Manual (Program Design)

2.3 Statement and Expression

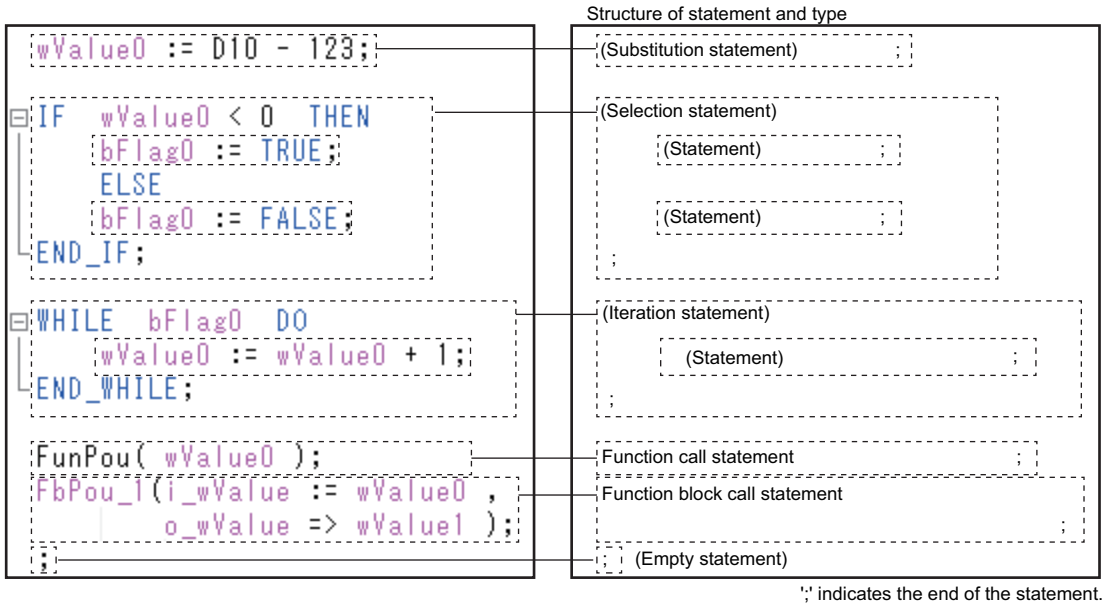
This section explains "statement" and "expression" which indicate units that configure an ST program.

Statement

A group of one execution process is referred to as "statement".

A program is described in "statement" units.

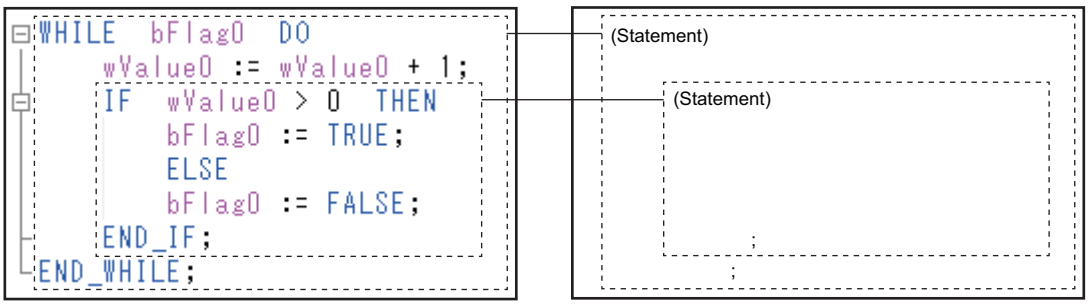
Each statement must end with a semicolon ';'.



The following shows the types of the statements.

Type		Description	Example
Assignment statement		Assigns the evaluation result in the right side to the variable in the left side.	Page 18 Assignment (:=)
Control statement	Selection statement (IF, CASE)	Selects an execution statement depending on the condition.	Page 22 Selection
	Iteration statement (FOR, WHILE, REPEAT)	Continues executing the execution statement for multiple times depending on the end condition.	Page 25 Iteration
	Exit of an iteration statement (EXIT)	Exits an iteration statement.	EXIT;
Subprogram control statement	Call statement	Calls a function or function block.	Page 99 Function and function block
	RETURN statement	Ends a process in the middle of a program.	RETURN;
Empty statement		Nothing is processed.	;

Hierarchization of a control statement can be performed. (Other statements such as a select statement or an iteration statement can be described among the statements.).

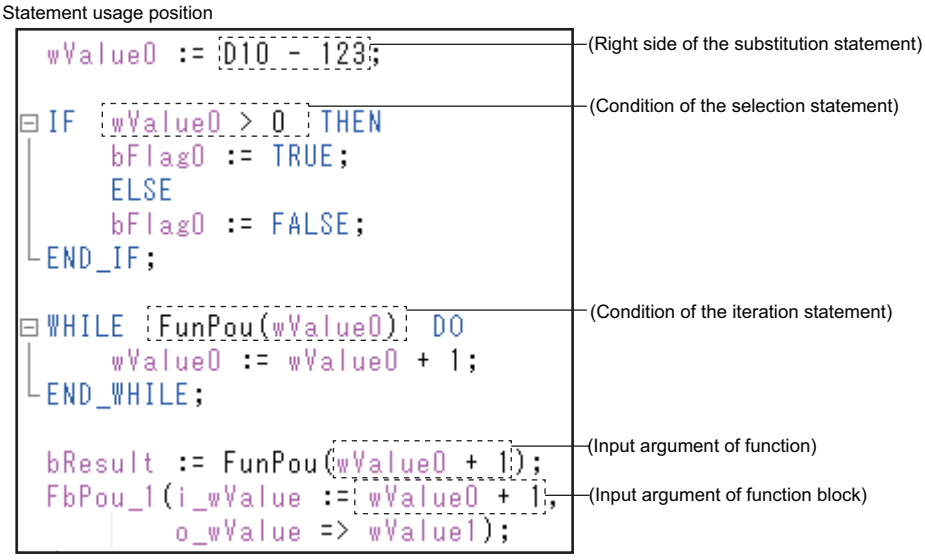


Expression

A description of values which are required for processing for a statement is referred to as "expression".
An expression is configured with variables and operators. The value of an expression is evaluated during the execution of a program.
The expressions are used in the following position in the statement.

- Right side of an assignment statement
- Execution result (EN) or input argument of a function and a function block
- Condition that is specified with a selection statement or iteration statement

Statement usage position



The data type of an expression is recognized at compilation (conversion). The value of an expression is evaluated during the execution of a program.
An operational expression such as an arithmetic operation and a comparison operation can be described by combining with constant or variable, and operator in the same manner as a generic expression.
A variable and constant can be used as an expression (primary expression) in the Structured Text language.
The types of expressions are as follows

Type		Data type of expression (operation result)	Example
Operational expression	Arithmetic expression	Integer, real number (depending on the operation target)	wValue0 + wValue1
	Logical expression	Boolean value (TRUE/FALSE)	bFlag0 OR bFlag1
	Comparison expression	Boolean value (TRUE/FALSE)	wValue0 > 0
Primary expression	Variable, constant	Defined data type	X0, wValue0, 123, TRUE
	Function call expression	Data type of return value	FunPou(wValue0, wValue1)

Point

A function which does not have a return value cannot be used as an expression.

3 DESCRIBING PERSPICUOUS PROGRAMS IN STRUCTURED TEXT

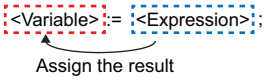
This chapter explains the process which makes it easy to read by describing using Structured Text language.

3.1 Operational Expressions

Complex operations which include decimal point or exponent can be described in the same manner as a general arithmetic expression using Structured Text language.

Assignment (:=)

The result of an evaluation can be stored to a variable using an assignment statement.
Describe an assignment statement using ':='. This operational expression stores the calculation result of the right side to the variable in the left side.



Point

Use ':=' in the Structured Text language instead of '=' which is used in a general expression.

Basic arithmetic operations (+, -, *, /)

Describe a basic arithmetic operation in the same manner as a general arithmetic expression using the operator (+, -, *, /).
The operation which cannot be described once in a ladder program can be described in one expression in the Structured Text language.

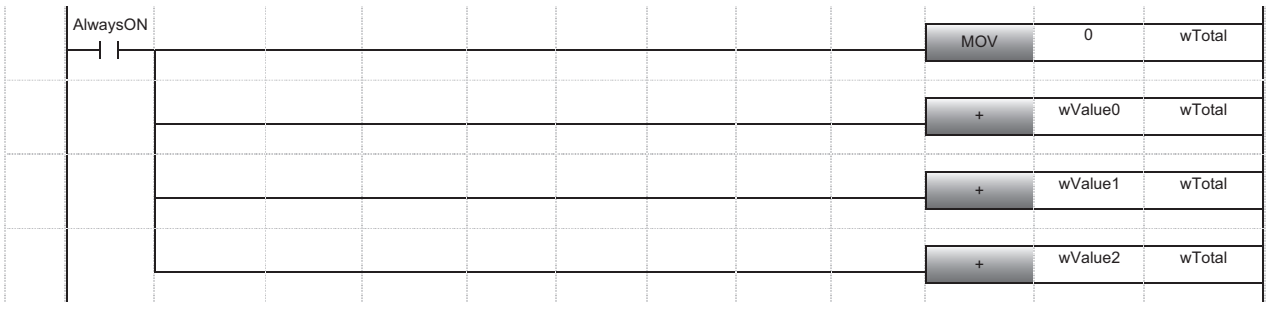
Program example

The sum from the wValue0 to the wValue2 is assigned to wTotal.
wTotal = wValue0 + wValue1 + wValue2

ST

```
wTotal := wValue0 + wValue1 + wValue2;
```

LD



Point

When multiple operational expressions are described in one statement, the operation is processed in order from high priority operation.

- Priority of basic arithmetic operations (high to low): Multiplication and division (*, /), addition and subtraction (+, -)

When some operators of which priority is the same are used in one statement, the operators are calculated in order from the left.

An operational expression in parentheses ( ) is calculated first.
Using the parentheses ( ) in complex basic arithmetic operations makes it easy to read the order of the expression.

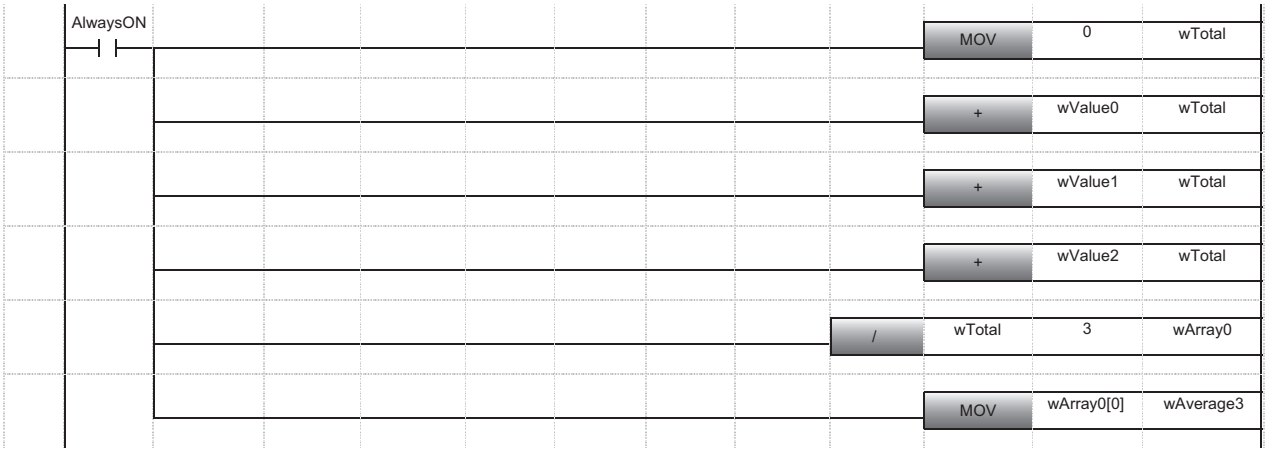
Program example

The average from the wValue0 to the wValue2 is assigned to the wAverage3.
wAverage3 = (wValue0 + wValue1 + wValue2) ÷ 3

ST

```
wAverage3 := ( wValue0 + wValue1 + wValue2) / 3;
```

LD



Advanced operations (exponent function, trigonometric function)

Describe exponent functions and trigonometric functions using standard functions.

Type		Function name	Example	
			General mathematical expression	ST
Absolute value		ABS	B = |A|	eValueB := ABS(eValueA)
Square root		SQRT	B = $\sqrt{A}$	eValueB := SQRT(eValueA);
Natural logarithm		LN	B = log e^A	eValueB := LN(eValueA);
Common logarithm		LOG	B = log 10^A	eValueB := LOG(eValueA);
Exponent		EXP	B = e^A	eValueB := EXP(eValueA);
Trigonometric functions	Sine, arcsine	SIN, ASIN	B = SIN A	eValueB := SIN (eValueA)
	Cosine, arccosine	COS, ACOS	B = COS A	eValueB := COS(eValueA);
	Tangent, arctangent	TAN, ATAN	B = TAN A	eValueB := TAN(eValueA);

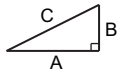
Describe an exponentiation using '**'.

Type	Operator	Example	
		General mathematical expression	ST
Exponentiation	**	B = C^A	eValueB := eValueC ** eValueA;

Program example

The length of a hypotenuse is obtained.

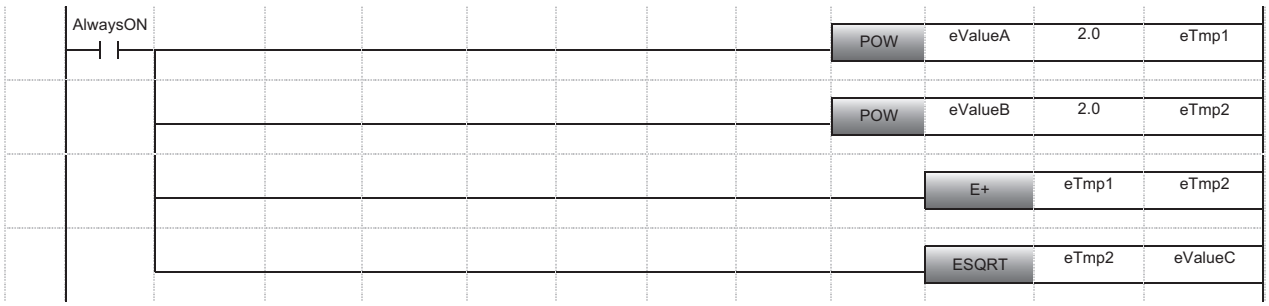
$C = \sqrt{(A^2 + B^2)}$



ST

```
eValueC := SQRT((eValueA ** 2.0) + (eValueB ** 2.0));
```

LD



Logical operation (AND, OR, XOR, NOT)

Describe a logical operation not with symbols (such as ∧, ∨, and ∀) but with an operator, which can easily be input and understood.

Program example

The logical product (AND) of the bFlag0 and the bFlag1 is assigned to the bResult.

ST

```
bResult := bFlag0 AND bFlag1;
```

LD

Point

AND operation can also be described using '&'.

When multiple operational expressions are described in one statement, the operation is processed in order from high priority operation.

- Priority of logical operator (high to low): NOT operation, AND operation (AND, &), XOR operation, OR operation

When some operators of which priority is the same are used in one statement, the operators are calculated in order from the left.

Comparison (<, >, <=, >=), equality/inequality (=,<>)

Describe a comparison operation using an equal sign or an inequality sign, which is the same symbol as a general arithmetic symbol.

Program example

The comparison result of the wValue0 and the wValue1 (equality: TRUE, inequality: FALSE) is assigned to the bResult.

ST

```
bResult := wValue0 = wValue1;
```

LD

Point

In the Structured Text language, '=' is regarded as an operator which compares if the left side and right side of the statements are equal.

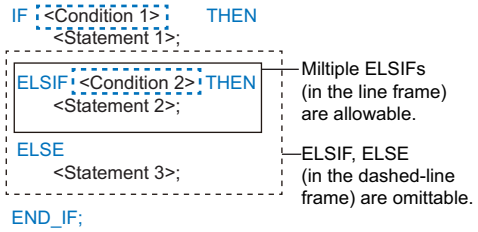
Describe an assignment statement using ':='.

3.2 Selection

In the Structured Text language, the process which branches off depending on the condition can be used in the same manner as a high-level programming language, such as C language.
By using the selection statement (IF, CASE), the case to be executed and the statement to be executed can be described.

Selection by boolean value (IF)

Describe the process which branches off depending on the condition (TRUE or FALSE) using an IF statement.
The following process is performed in an IF statement.



1. Judgment of IF

When Condition expression 1 is TRUE, the Statement 1 is executed.

2. Judgment of ELSIF

When the conditional expression above is FALSE, the condition is judged.
When the Condition expression 2 is TRUE, the Statement 2 is executed.

3. Judgment of ELSE

When all the conditions of 'IF' and 'ELSIF' are FALSE, the Statement 3 after 'ELSE' is executed.

Point

The following items can be specified in the conditional expression of IF statement.

- Operational expression of which result is to be a boolean value
- Variable of boolean type
- Function call expression of which return value is boolean type

Multiple selections using ELSIF (ELSIF<Condition>THEN<Statement>;) can be set.
Describe the selection using ELSIF or ELSE as necessary. (Omittable)

Program example

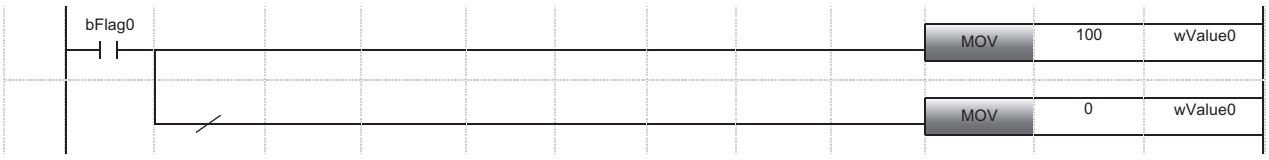
The different value is set to the wValue0 depending on the condition of the bFlag0.

- When ON (bFlag0 is TRUE): wValue0 = 100
- When OFF (bFlag0 is FALSE): wValue0 = 0

ST

```
IF bFlag0 THEN
    wValue0 := 100;
    ELSE
    wValue0 := 0;
END_IF;
```

LD



By using the expression in which a logical operation (AND, OR, etc.) or a comparison (<, >, =, etc.) are combined for a conditional expression of the selection statement, a program which can be branched with complex conditions can be described.

Program example

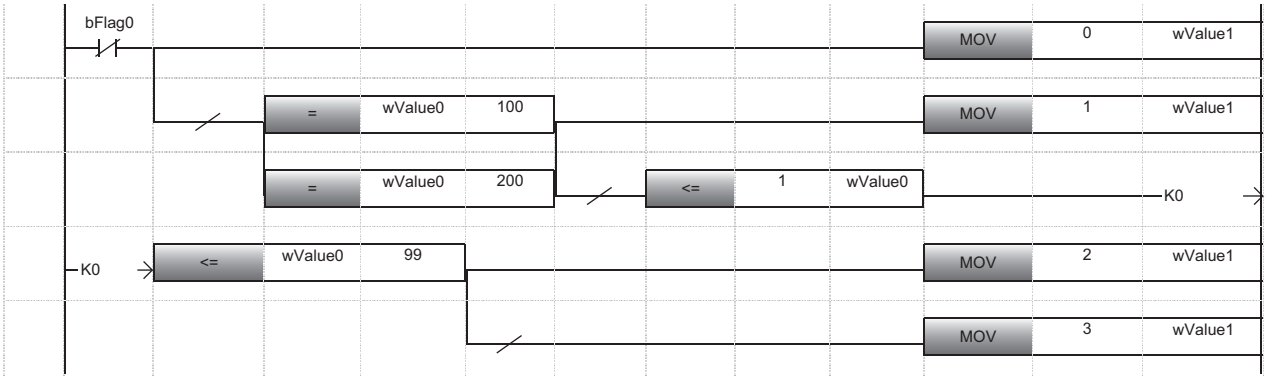
0, 1, 2, or 3 is set to the wValue1 depending on the value of the bFlag0 and the wValue0.

- When bFlag0 is FALSE: 0
- When the bFlag0 is TRUE and the wValue0 is 100 or 200: 1
- When the bFlag0 is TRUE and the wValue0 is 1 to 99: 2
- Other than above: 3

ST

```
IF NOT bFlag0 THEN
    wValue1 := 0;
    ELSIF (wValue0 = 100) OR (wValue0 = 200) THEN
    wValue1 := 1;
    ELSIF (1 <= wValue0) AND (wValue0<= 99) THEN
    wValue1 := 2;
    ELSE
    wValue1 := 3;
END_IF;
```

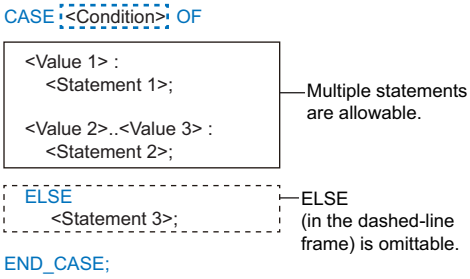
LD



Selection (CASE) by integer

Describe the process of which process branches using integer values with CASE statement.

The following process is performed in a CASE statement.



Selects an execution statement depending on the condition of the integer value.

1. Judgment of the Value 1

Execute the Statement 1 if the result of the conditional expression is equals to the Value 1.

2. Judgment of the integer value in the specified range

When specifying the range of the integer value to be judged, use '..'.

When the result of the conditional expression is within the range of the Value 2 to Value 3, the Statement 2 is executed.

3. Judgment of ELSE

When all the integer values or ranges are equal, the Statement 3 after 'ELSE' is executed.

Point

The following items can be specified in the conditional expression of CASE statement.

- Operational expression of which result is to be an integer (INT or DINT) value
- Variable of integer (INT or DINT) type
- Function call expression of which return value is integer (INT or DINT) type

Multiple selections (<Value>:<Statement>;) determined by an integer value can be set.

Describe the selection using ELSE (ELSE<Statement>) as necessary. (Omittable)

For an integer value to be judged, a label that satisfies the following conditions can be set: the data type is Word [Signed] or Double Word [Signed], and the class is VAR_CONSTANT.

Program example

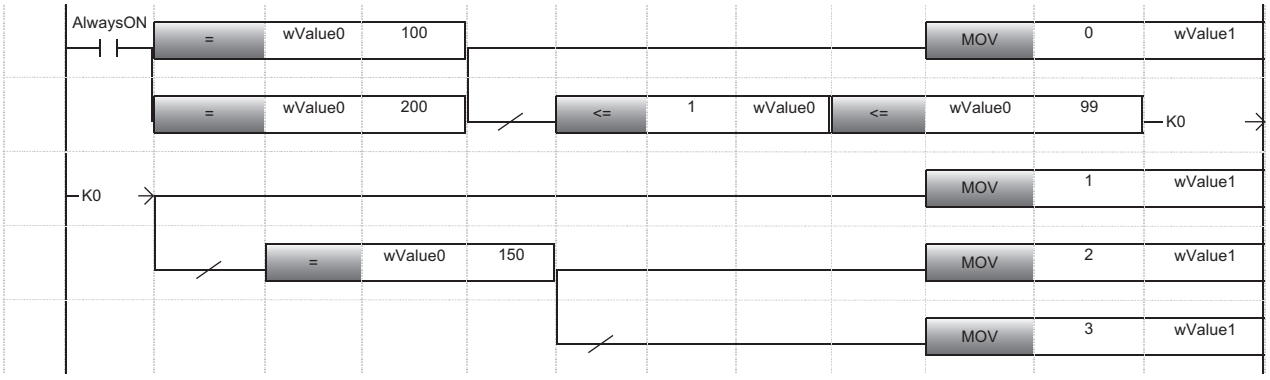
0, 1, 2, or 3 is set to the wValue1 depending on the value of the wValue0.

- When 100 or 200: 0
- When 1 to 99: 1
- When 150: 2
- Other than above: 3

ST

```
CASE wValue0 OF
    100, 200: wValue1 := 0;
    1..99: wValue1 := 1;
    150: wValue1 := 2;
    ELSE wValue1 := 3;
END_CASE;
```

LD



3.3 Iteration

By using the iteration statement (WHILE, REPEAT, FOR), the process can be executed for multiple times with the end condition specified.

Iteration by boolean condition (WHILE, REPEAT)

Describe the process to be repeated depending on the result of its condition (TRUE or FALSE) using WHILE statement or REPEAT statement.

The following process is performed in a WHILE statement.

```
WHILE <Condition>
  DO
    <Statement>;
END_WHILE;
```

Repeat until the result becomes FALSE

Execute the execution statement for multiple times depending on the end condition of the boolean value.

Repeat processing until the result of the conditional expression is FALSE.

1. Judgment of the condition

If a conditional expression is FALSE, the process is end.

2. Execution of the process

Execute the execution statement and repeat processing when the conditional expression is TRUE.

The following process is performed in a REPEAT statement.

```
REPEAT
  <Statement>;
  UNTIL <Condition>
END_REPEAT;
```

Repeat until the result becomes TRUE

Execute the execution statement for multiple times depending on the end condition of the boolean value.

Repeat processing until the result of the conditional expression is TRUE.

1. Execution of the process

Execute the execution statement.

2. Judgment of the condition

If a conditional expression is TRUE, the process is end.

If a conditional expression is FALSE, the process is repeated.

Point

The following items can be specified in the conditional expression of WHILE and REPEAT statement.

- Operational expression of which result is to be a boolean value
- Variable of boolean type
- Function call expression of which return value is boolean type

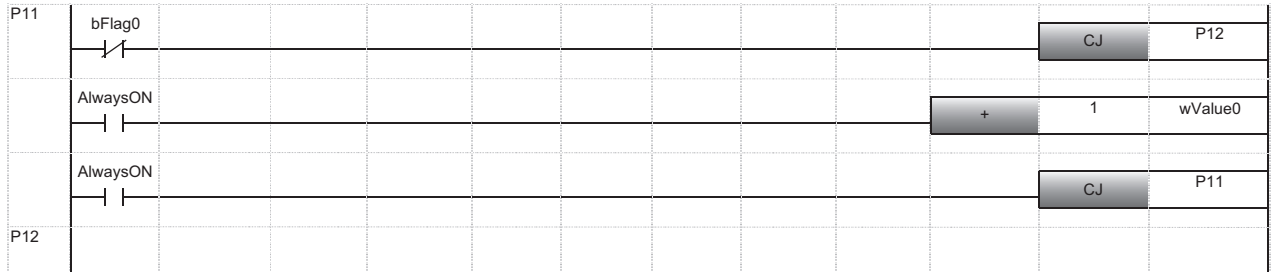
Program example

When the bFlag0 is TRUE, the wValue0 is incremented.

ST

```
WHILE bFlag0 DO
    wValue0 := wValue0 + 1;
END_WHILE;
```

LD



By using the expression in which a logical operation (AND, OR, etc.) and a comparison (<, >, =, etc.) are combined for a conditional expression of the iteration statement, a program of which end condition is complex can be described.

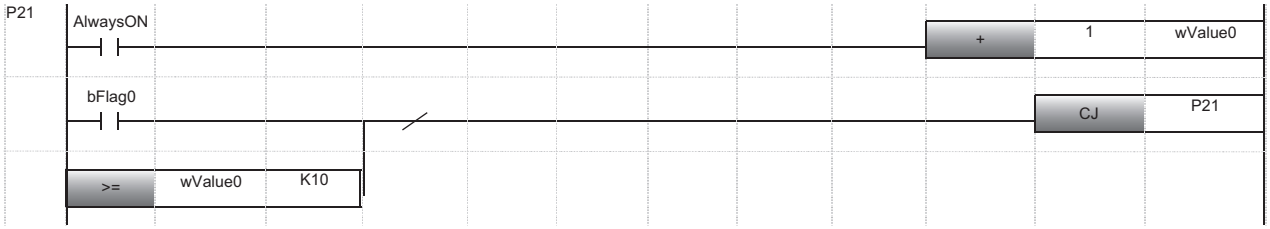
Program example

The wValue0 is incremented until the bFlag0 is TRUE or the wValue0 is 10 or more.

ST

```
REPEAT
    wValue0 := wValue0 + 1;
    UNTIL bFlag0 OR (wValue0 >= 10)
END_REPEAT;
```

LD



Iteration by integer value (FOR)

Describe the process which repeats processing until the variable of an integer type meets the condition using FOR statement.
The following process is performed in a FOR statement.

```
FOR <Variable> := <Initial (expression)>
  TO <End (expression)>
  BY <Increment (expression)>
  DO <Statement>;
END_FOR;
```

BY (in the dashed-line (thin) frame) is omittable.

Execute the execution statement for multiple times depending on the end condition of the integer value.
The execution is repeated until the variable of the integer type, to which the initial value has been set, is reached at the last value.

1. Initialization of variable

Set the initial value to the variable to be a condition.

2. Judgment of the condition

If the variable is reached at the last value, the process is end.

3. Execution of the process

Execute the execution statement.

4. Addition of increment value

Add the increment value to the variable, and repeat processing.

Point

The following items can be specified for the initial value, last value, and increment value of FOR statement.

- Operational expression of which result is to be an integer (INT or DINT) value
- Variable of integer (INT or DINT) type
- Function call expression of which return value is integer (INT or DINT) type

Convert the type in order that the value is to be an integer type.
If the increment value is 1, the addition process of the increment value (BY<Increment (expression)>;) can be omitted.
After the execution of the FOR statement, the variable set as a condition retains the value at the end of the FOR statement.

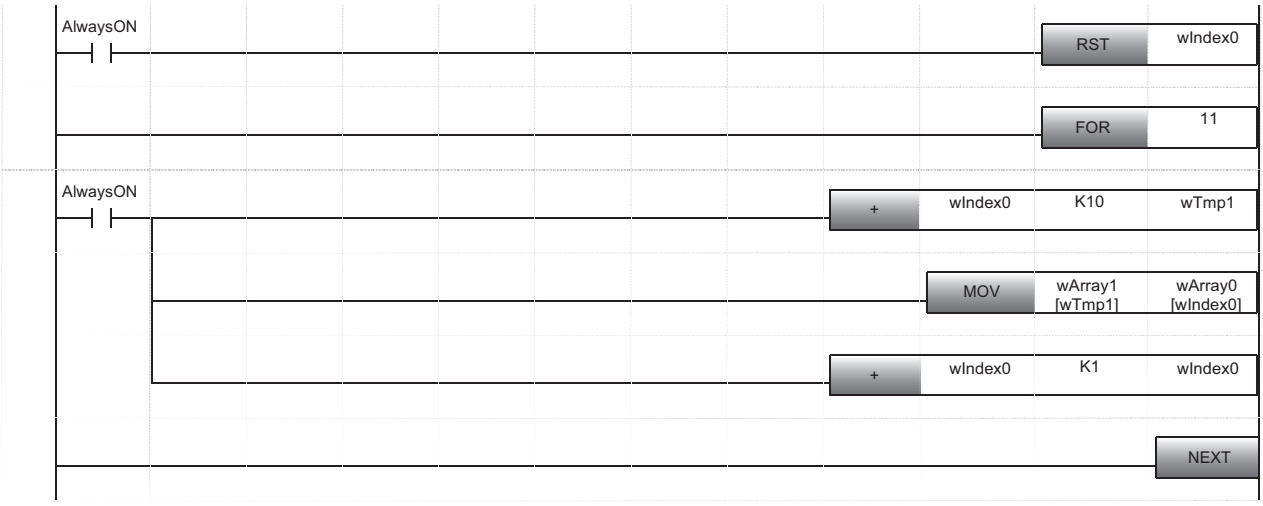
Program example

The element from 10 to 20 of the wArray1 are set to the element from 0 to 10 of the wArray0.

ST

```
FOR wIndex0 := 0 TO 10 BY 1 DO
    wArray0[wIndex0] := wArray1[wIndex0 + 10];
END_FOR;
```

LD



Point

By combining the array type data with iteration statement, the identical data processing can be performed for multiple array elements.

4 HANDLING VARIOUS DATA TYPES

This chapter explains the considerations when handling the variables of each data type, how to specify the data types, and how to convert using the Structured Text language.
Define the data type of labels at registration of labels. The data type of devices differ depending on the device type.

Point

When using the value with the different data type as the defined one, use a type conversion function of a standard function.

4.1 Boolean Value

Boolean is a 1 bit data type that holds 0 or 1.
Use this data type when indicating the ON/OFF for a bit device or TRUE/FALSE for an execution result.
A label to which a bit device or "Bit" data type is set is treated as a variable of boolean value.

4.2 Integer and Real Number

An integer type label, a real number type label, and a word device are treated as a variable of integer or real number.
The basic data type of a value used for each processing is integer type. When handling the value of decimal place, use a value of real number type.

Value of range

Depending on the type of variable, the effective digits differ. Specify the variable of the appropriate data type depending on the operation to be executed.
The following shows the ranges that can be handled in operation depending on the data type of the variable.

Data type			Range
Integer	Word [Unsigned]/Bit String [16-bit]	WORD	0 to 65535
	Double Word [Unsigned]/Bit String [32-bit]	DWORD	0 to 4294967295
	Word [Signed]	INT	-32768 to 32767
	Double Word [Signed]	DINT	-2147483648 to 2147483647
Real number	FLOAT [Single Precision]	REAL	-2^{128} to -2^{-126}, 0, 2^{-126} to 2^{128}
	FLOAT [Double Precision]	LREAL	-2^{1024} to -2^{-1022}, 0, 2^{-1022} to 2^{1024}

Type conversion which is performed automatically

When using the Structured Text language in GX Works3, data types of an integer and a real number are converted automatically with the following processing even when a different variable of data type or constant is described.

- Assignment statement
- Pass of the input argument to a function or function block
- Arithmetic operational expression

The data type conversion is performed automatically to the data type of which range is larger.

Ex.

Assignment between Word [Signed] (INT) and Double Word [Signed] (DINT)

ST	Operation result	
dValue0 := wValue1;	○	The type conversion from INT to DINT is performed automatically.
wValue1 := dValue0;	×	A conversion error occurs since the data may be lost at conversion from DINT to INT.
wValue1 := DINT_TO_INT(dValue0);	○	Perform a type conversion from DTIN to INT using a type conversion function. If the range of the value before conversion exceeds the range of INT type, an operation error occurs.

Point

The type conversion is not performed automatically under the following situation. In this case, use a type conversion function.

- Type conversion for the integer which has same data size and different sign
- Type conversion for the type of which data may be lost
- Type conversion other than integer type and real number type

Data type that can be converted automatically

The following shows the combinations of data types which are converted automatically.

Data type before conversion	Data type after conversion	Remarks
Word [Signed]	Double Word [Signed]	The value is converted to the sign-extended value automatically.
	FLOAT [Single Precision]	The value is converted to the same value before the conversion automatically.
	FLOAT [Double Precision]	
Word [Unsigned]/Bit String [16-bit]	Double Word [Signed]	The value is converted to the zero-extended value automatically.
	Double Word [Unsigned]/Bit String [32-bit]	
	FLOAT [Single Precision]	The value is converted to the same value before the conversion automatically.
	FLOAT [Double Precision]	
Double Word [Signed] Double Word [Unsigned]/Bit String [32-bit]	FLOAT [Double Precision]	
FLOAT [Single Precision]		

The data conversion is also performed automatically when passing the input argument to a standard function, standard function block, and instruction.

The following shows the combinations of data types which are converted automatically when the input argument is defined as the generic data type.

Data type of variable (before conversion) to be specified to argument	Data type definition of input argument	Data type after conversion
Word [Signed]	ANY32, ANY32_S	Double Word [Signed]
	ANY_REAL, ANY_REAL_32	FLOAT [Single Precision]
	ANY_REAL_64	FLOAT [Double Precision]
Word [Unsigned]/Bit String [16-bit]	ANY32, ANY32_U	Double Word [Unsigned]/Bit String [32-bit]
	ANY_REAL, ANY_REAL_32	FLOAT [Single Precision]
	ANY_REAL_64	FLOAT [Double Precision]
Double Word [Signed] Double Word [Unsigned]/Bit String [32-bit]	ANY_REAL, ANY_REAL_64	FLOAT [Double Precision]
FLOAT [Single Precision]	ANY_REAL_64	

Data type of the operation result of arithmetic expression

The operation result of an arithmetic operational expression (basic arithmetic operation) for the Structured Text language is the same data type as the variable and constant of the operation target. (For an exponentiation (**), the operation result will be a real number type.)

Point

If the data type of the operator on the right side and left side differ, the data type of the operation result will be a bigger data type.

- Priority of the data type of an operation result (high to low): FLOAT [Double Precision], FLOAT [Single Precision], Double Word, Word

[Singed] and [Unsigned] cannot be mixed in binary operation of the integer.

Considerations

If the value is outside the range of the data type which can be handled in the operation result, the accurate result (value) cannot be reflected to the process after the operation.
Convert the data type of the variable for the operation target to the data type within the range of the operation result in advance.

Ex.
Arithmetic operation of Word [Signed] (INT)

ST	Operation result	
dValue0 := wValue1 * 10;	×	If the operation result is out of the range of INT type (-32768 to 32767), the operation resulted in overflow or underflow is assigned.
dValue1 := INT_TO_DINT(wValue1); dValue0 := dValue1 * 10;	○	Overflow or underflow does not occur since the operation is processed with DINT type.
dValue1 := INT_TO_DINT(wValue1) * 10;	○	

Division of integer and real number

For a division, the operation result may differ depending on the type of variable.
For a division of an integer type variable, the value of decimal place is cut off.
For a division of a real number type variable, the value of decimal place is calculated.

- FLOAT [Single Precision]: 7 effective digits (6 digits of decimal places)
- FLOAT [Double Precision]: 15 effective digits (14 digits of decimal places)

Ex.
Assign the operation result of the expression '(2 ÷ 10) × 10' to D0

Data type		ST	Operation result
Integer	Word [Signed]	D0 := (2 / 10) * 10;	0
Real number	FLOAT [Single Precision]	D0:E := (2.0 / 10.0) * 10.0;	2.0

Point

For word devices, a data type can be specified by adding ':E' in the Structured Text language.
(☞ Page 94 Type specification of word device)

Remainder of division (MOD)

Calculate the remainder of division by modulus operation (MOD).

Ex.
Calculate the lower two-digits of a five-digits integer

Data type		ST	Operation result
Integer	Word [Signed]	D0 := (-32368 MOD 100);	-68

4.3 Character String

Use a character string processing instruction and standard function (character string function) for the processing of the variable of a character string type.

Program example

The length of a character string is calculated.

ST

```
wLen0 := LEN(sString0);
```

Assignment of character string

Describe a character string type variable using an assignment statement.

Program example

The character string 'ABC' is assigned to the sString0 (character string variable) and the wsString1 (character string [Unicode] type variable).

ST

```
sString0 := 'ABC';
wsString1 := "ABC";
```

Point

Enclose ASCII character string constant in single quotes (').
Enclose Unicode character string constant in double quotes (").

Comparison of character strings

Describe a comparison operation (comparison, equality, inequality) of a character string type variable using an operator.

Program example

After the comparison of the character strings, the sString1 is assigned to the sString0 if the characters are not equal.

ST

```
IF sString0 <> sString1 THEN
    sString0 := sString1;
END_IF;
```

Point

For the character string type variable, a comparison operator (<, >, <=, >=) compares the values using the value of ASCII code or Unicode.
Depending on the first character code number of which inequality is detected, the comparison result of the character string is determined.
The details of the comparison conditions are the same as the one as the character string comparison instruction (LD$<, etc.)

4.4 Time

Time data is used for the time type variable or clock data (array data used for CPU module instructions).

Time type variable

By using the time type variable, an easy-to-see program in which time data in milliseconds is included can be created.
Describe the time type of constants as follows:

- T#23d23h59m59s999ms

Time unit	d (day)	h (hour)	m (minute)	s (second)	ms (millisecond)
Range	0 to 24	0 to 23	0 to 59	0 to 59	0 to 999

The time type variable can be set in the range from T#-24d20h31m23s648ms to T#24d20h31m23s647ms.
For details on the description method of the time length, refer to the following:
☞ Page 98 Description method of the time length

Assignment of time

Describe a time type variable using as assignment statement.

Program example

The value that indicates 1 hour 30 minutes is assigned to the tmData0 (time type variable).

ST

```
tmData0 := T#1h30m;
```

Comparison of time

Describe a comparison operation (comparison, equality, inequality) of a time type variable using an operator.

Program example

If the time data is one day or more, the processing is ended.

ST

```
IF tmData0 >= T#1d THEN
    RETURN;
END_IF;
```

Basic arithmetic expression for time

Describe a multiplication and division of the time type variable using a standard function (time data type function).
An addition or subtraction can be described using an operator.

Program example

The value in which 10 ms is added and doubled is assigned to the tmData0 (time type variable).

ST

```
tmData0 := MUL_TIME(tmData0 + T#10ms, 2);
```

Clock data (date and time)

Clock data is an array data that is used for CPU module instructions.

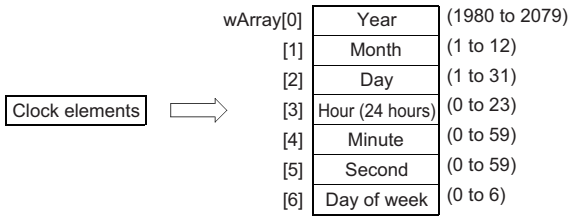
- Clock data: An array that stores year, month, day, hour, minute, second, and day of week
- Extended clock data: An array of which millisecond unit is added to a clock data
- Date data: An array only for year, month, and day of clock data
- Time data: An array only for hour, minute, and second of clock data

Clock instruction and clock data

Use the clock instruction of the application instruction to process a clock data (array of Word [Signed] data).

Program example

"year, month, day, hour, minute, second, and day of week" are read from the clock element of the CPU module.



ST

```
DATERD(TRUE, wArray);
```

Comparison of date data and time data

The comparison instruction for date data and time data cannot be used in ST programs. (☞ Page 101 Instructions That Cannot be Used in ST Programs)

To compare a date data and time data, compare each element of the array or convert the data to the time type variable.

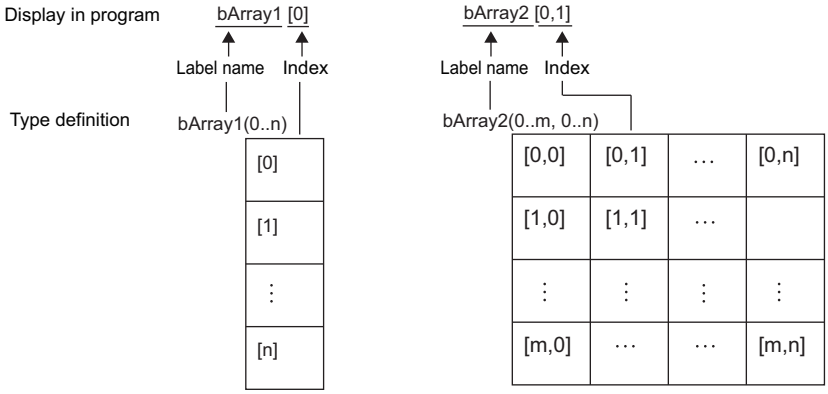
4.5 Array and Structure

An array and structure are the data format that can handle multiple data at once.

- Array: A consecutive aggregation of same data type variables
- Structure: A aggregation of different data type variables

Array

Describe only the label name when indicating whole array type variables.
When indicating each element of the array, describe the element number to be specified enclosing with '[ ]' as an index after the label name.



Assignment of array

Describe an array type variable using the assignment statement to assign the value to the element specified by the index.
By describing the element without index, the value is assigned (copied) to the whole array elements.

Program example

The following data is assigned to the wArray1 (array type variable) of Word [Signed] data.

- Array 0: 10
- Array 1: Array [0,1] of two-dimensional array, wArray2

ST

```
wArray1[0] := 10;
wArray1[1] := wArray2[0, 1];
```

Program example

All the elements of wArray0 are assigned to all the elements of the wArray1 (array type variable of Word [Signed] data).
(This program example can be described when the data type and number of array data on the right side and left side are the same.)

ST

```
wArray1 := wArray0;
```

Expression of array

Describe the elements of each array using an operational expression as a variable of the defined data type. (Comparison and basic arithmetic expression can be described.)
However, the array type variable with no index cannot be used in the operational expression.
Array element can be described in an expression in the Structured Text language.

Program example

The following data is assigned to the wArray1 (array type variable) of the Word [Signed] data.

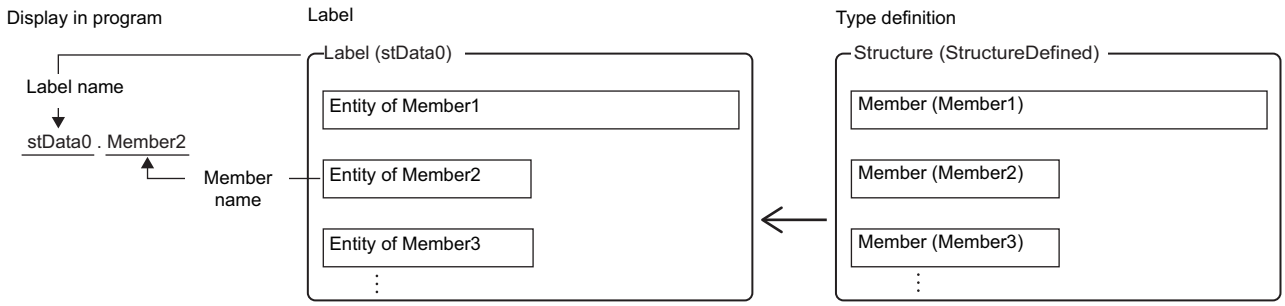
- Element wIndex0+1: Sum of the element 0 and element 1 in the wArray0

ST

```
wArray1[wIndex0 +1] := wArray0[0] + wArray0[1];
```

Structure

Define the structure combining the multiple different data types as one data type.
Structure can be used for data management because the arbitrary name of structure type definition and name of each member can be set.
When indicating whole structure type variables, describe only the label name.
When indicating each member of the structure, list the member names by adding a dot '.' after the label name.



Assignment of structure

Describe a structure type variable using the assignment statement to assign the value to the specified member.
By describing the structure without specifying the member, the value is assigned (copied) to the whole structures.

Program example

The following data is assigned to the member of the stData0 (structure type variable).

- Member1 (FLOAT [Single Precision]): 10.5
- Member2 (Bit): TRUE

ST

```
stData0.Member1 := 10.5;
stData0.Member2 := TRUE;
```

Program example

All the elements of stData0 are assigned to all the elements of the stData1 (structure type variable).
(This program example can be described when the data type of the structure (structure type definition) on the right side and left side are the same.)

ST

```
stData1 := stData0;
```

Data type combined with structure and array

A structure which includes array in the member and a structure type array can be used in ST programs.

Program example

The wArray1 (array type variable) is assigned to the wArray (array member) of the stData0 (structured type variable). (Both of them are Word [signed] type and same size.)

ST

```
stData0.wArray := wArray1;
```

Program example

The member of the element number 1 and the member of element number 0 in the stArray0 (structured array type variable) are compared.

ST

```
bResult := stArray0[1].Member1 > stArray0[0].Member1;
```

Point

Structure arrays can be used in the assignment statement in GX Works3.

5 DESCRIBING LADDER PROGRAM IN STRUCTURED TEXT

This chapter explains how to describe ladder symbols and sequence instructions in the Structure Text language.

5.1 Describing Contacts and Coils

The program using contacts and coils of a ladder program can be described with the logical operations such as AND operation, OR operation, and NOT operation and assignment statements in the Structured Text language.

Open contact and coil

Describe a program configured with an open contact and coil using an assignment statement.

Program example

The bResult0 turns ON/OFF according to the ON/OFF statue of the bFlag0.

ST

```
bResult0 := bFlag0;
```

LD

Point

Describe an assignment statement using ':='. The calculation result of a right side is stored to a variable on the left side.

Closed contact (NOT)

Describe a closed contact with the operator of a NOT operation.

Program example

The bResult0 turns OFF when the bFlag0 is ON, and the bResult0 turns ON when the bFlag0 is OFF.

ST

```
bResult0 := NOT bFlag0;
```

LD

Series connection, parallel connection (AND, OR)

Describe the condition described in a series connection and parallel connection using AND operation(AND, &) or OR operation.

Program example

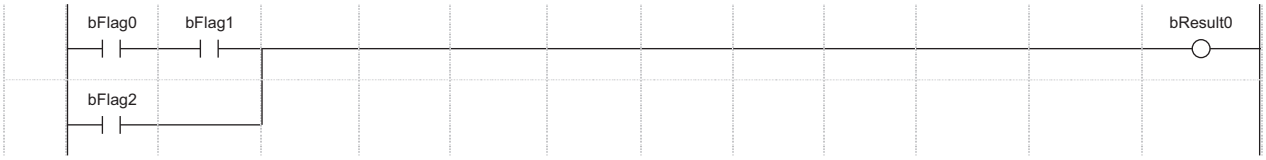
The bResult0 turns ON when any of the following conditions is met.

- Condition 1: The bFlag0 is ON and the bFlag1 is ON
- Condition 2: The bFlag2 is ON

ST

```
bResult0 := bFlag0 AND bFlag1 OR bFlag2;
```

LD



Point

When multiple operational expressions are described in one statement, the operation is processed in order from high priority operation.

- Priority of logical operator (high to low): AND operation(AND, &), XOR operation, OR operation

When some operators of which priority is the same are used in one statement, the operators are operated in order from the left.

Contact and coil of which execution order are complicated

The complex combination of contacts and coils can be described in the Structured Text language. By using the parentheses ( ), the statements of a complex execution order can be described clearly.

Program example

The bResult0 turns ON when the following condition 1 and condition 2 are met.

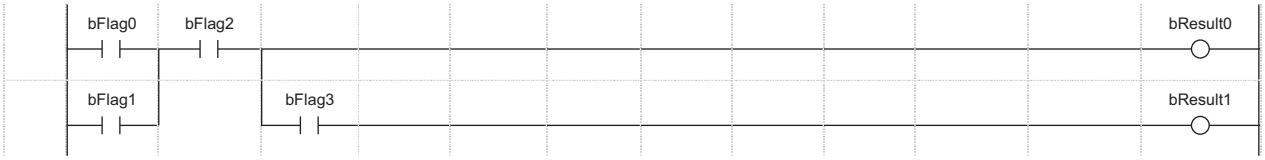
The bResult1 turns ON when the following condition 1, condition 2, and condition 3 are met.

- Condition 1: Either the bFlag0 or the bFlag1 is ON
- Condition 2: The bFlag2 is ON
- Condition 3: The bFlag3 is ON

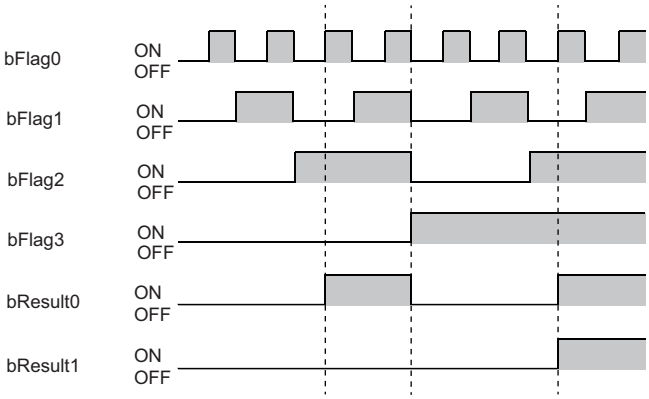
ST

```
bResult0 := (bFlag0 OR bFlag1) AND bFlag2;
bResult1 := bResult0 AND bFlag3;
```

LD



When the above program is executed, the ON/OFF timing of each device are as follows;



5.2 Describing Instructions

In the Structured Text language, an instruction which can commonly be used with Ladder Diagram is handled as a function. Some instructions which are not supported by the Structure Text language (☞ Page 101 Instructions That Cannot be Used in ST Programs) can be described using the format for the Structured Text language such as operators.

Instructions that can be used in ladder program and ST program

Basically, every instructions can be used in a ladder program and ST program. (☞ Page 14 Instructions and Functions)
In accordance with the definitions of each instruction, describe the instruction as a function.

Program example

The bResult0 turns ON for one scan when the bFlag0 is turned ON from OFF.

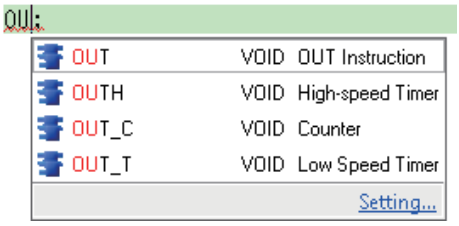
ST

```
PLS(bFlag0, bResult0);
```

LD

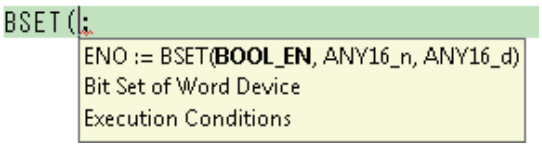
Considerations when entering instructions

Despite the same function, some instruction names differ between Ladder Diagram and Structured Text. (Low-speed timer instruction (OUT_T) of output instruction, etc.)
When entering an instruction name in the ST editor, the list of the instructions start with the character of the input instruction name is displayed.
Check if the instruction name can be used in the ST program.



Considerations when entering arguments

The order of arguments for the same instruction may differ between Ladder Diagram and Structured Text.
The format of an instruction is displayed in tooltip when entering an argument in an ST editor.
Input the arguments in accordance with the tooltip.



For details on the instructions, refer to the programming manual. Press the F1 key with the cursor on the instruction to display the page of the instruction.

Instructions that can be described using assignment statements

Describe a data conversion instruction using the type conversion function of a standard function and assignment statement.
Describe a character string transfer instruction ($MOV) using the assignment statement of a string type label.
☞ Page 101 Instructions that can be described in assignment statement

Program example

The eValue0 (FLOAT [Single Precision]) is converted to the wValue1 (16-bit binary data with sign).

ST

```
wValue1 := REAL_TO_INT( eValue0 );
```

LD

Program example

The sString0 (character string type variable) is transferred (assigned) to the sString1.

ST

```
sString1 := sString0;
```

LD

Instructions that can be described using operator

Describe basic instructions such as comparison operation instructions and arithmetic operation instructions using operators.
☞ Page 101 Instructions that can be described with operator

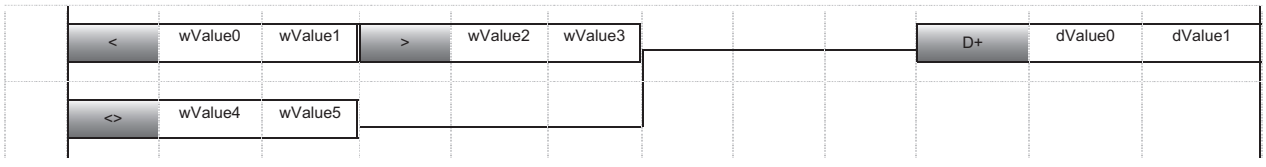
Program example

Depending on the comparison result of the Word [Signed] (INT) valuable, the two values of Double Word [Signed] (DINT) are added.

ST

```
IF (wValue0 < wValue1 ) AND (wValue2 > wValue3 ) OR (wValue4 <> wValue5 ) THEN
    dValue1 := dValue0 + dValue1;
END_IF;
```

LD



Instructions that can be described in control statement and FUN/FB

Describe FOR to NEXT instructions of the structured instructions using an iteration statement (☞ Page 25 Iteration).
A program in which a pointer is specified using a subroutine program instruction (such as CALL) or a pointer branch instruction (such as CJ, SCJ, and JMP) cannot be used in ST program. Structure the program using a selection statement, function ,or function block.
☞ Page 103 Instructions that can be described with control statement or function

Point

END instruction is not required for the Structured Text language. (☞ Page 103 Unnecessary instructions for ST program)

5.3 Describing Statements of Ladder and Notes

Describe a statement and note of a ladder program as a comment.
The comment of the Structured Text language is more useful than statement and note since it can be described in arbitrary position.

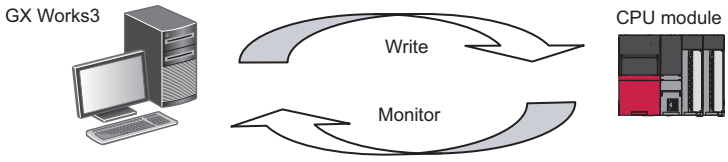
```
(* The processing is performed depending on the ten-key input. *)
IF G_wTenKey <> c_wNONE THEN
        CASE G_wTenKey OF
                0..9 : (* For number-key input (0 to 9) *)
                (* Add the input numeric value to the end of the display value. *)
                IF G_eDecimal = 0.0 THEN
                        (* For integer part *)
                        G_eDisplayValue := (G_eDisplayValue * 10) + G_wTenKey;
                        ELSE
                        (* For after decimal point *)
                        G_eDisplayValue := G_eDisplayValue + (G_eDecimal * G_wTenKey);
                        G_eDecimal := G_eDecimal * 0.1;
                END_IF;
                10: (* For input of decimal point key *)
                G_eDecimal := 0.1;
                11..14: (* For input of addition, subtraction, multiplication, or division key (11 to 14) *)
                (* Retain the operation type *)
                G_wOperation := G_wTenKey - 10;
                (* Move the display value to the previous operation value and then reset the displayed value *)
                G_eLastValue := G_eDisplayValue;
                G_eDisplayValue := 0.0;
                G_eDecimal := 0.0;
                15: (* For equal-key input *)
                (* Add, subtract, multiply, or divide the displayed value to/from the current value. *)
                G_eLastValue := Calculation(G_eLastValue, G_wOperation, G_eDisplayValue);
                (* Assign the rounding result to the display value. *)
                G_eDisplayValue := FractionProcessing(G_eLastValue, G_wSwitch1, G_wSwitch2);
                G_wOperation := 0; (* Clear the operation type *)
                G_eDecimal := 0.0;
        END_CASE;
        (* Clear the key input*)
        G_wTenKey := c_wNONE;
END_IF;
```

6 PROGRAM CREATION PROCEDURE

6.1 Overview of Procedure

This section explains the creation procedure of a program.

1. Open an ST editor.
2. Edit the ST program.
3. Convert and debug the program.
4. Check the program in the CPU module.



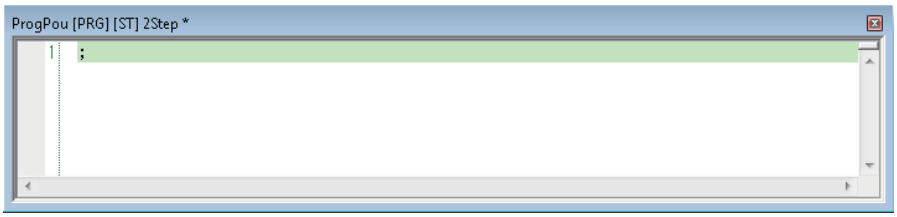
Point

For details on the operations of GX Works3, refer to the following:
GX Works3 Operating Manual

6.2 Opening ST Editor

Create an ST program using the programming function of the engineering tool (GX Works3).
Select "ST" in "Program Language" and create a new project and program.

ST editor



6.3 Editing ST Programs

This section explains a series of steps from creating a simple program to execute it.
Create the following program as an example.

Program example

```
wValue0 := D10 - 123;

IF  wValue0 < 0  THEN
    bFlag0 := TRUE;  (* ON *)
    ELSE
    bFlag0 := FALSE; (* OFF *)
END_IF;

bResult := FunPou(wValue0);
FbPou_1(i_wValue := wValue0,
        o_wValue => wValue1);
```

Entering texts

Enter texts in the same manner as a generic text editor in an ST editor.

Operating procedure

```
wValue0 := D10 - 123;
```

1. Enter 'wValue0 := D10 - 123;'.

Point

By pressing the Ctrl + Shift + = keys, the assignment symbol (:=) can be entered.
Other than the operation from the [Edit] menu, the following generic key operations can be used.

- Shift + Delete / Shift + Insert: Cut/paste
- Alt + Back space / Ctrl + Back space: Undo/redo

Copying/pasting data among editors

Text data can be copied/pasted among the generic text editors.

- Utilizing the created programs by copying texts from other text editor or PDF and pasting them to an ST editor.
- Creating documents by copying the programs created in an ST editor and pasting them to other text editor.

Point

In GX Works3, the rectangle range can be selected by dragging the range while pressing the Alt key on the ST editor.

Entering control statement

Enter the control statement (IF statement).

Operating procedure

```
IF
```

1. Enter 'IF'.
2. The syntax templates are displayed by pressing the Ctrl + F1 keys.

```
IF  ?Condition?  THEN
    ?Statement? ;
ELSE
    ?Statement? ;
END_IF;
```

```
IF  wValue0 < 0  THEN
    ?Statement? ;
ELSE
    ?Statement? ;
END_IF;
```

3. Move the cursor position on '?Condition?'.
(The cursor can be moved by pressing the Ctrl + Alt + ← keys.)
4. Enter "wValue0 < 0" as a conditional expression.

```
IF  wValue0 < 0  THEN
    bFlag0 := TRUE;
ELSE
    bFlag0 := FALSE;
END_IF;
```

5. Move the cursor position on '?Statement?'.
(The cursor can be moved by pressing the Ctrl + Alt + → keys.)
6. Enter "bFlag0 := TRUE;" as an execution statement.
   - If "TRUE" is entered in lower case, it will be changed to upper case automatically.

Enter "bFlag0 := FALSE;" as well.

Point

The following functions are used in the operation procedure above.

- [Edit] ⇨ [Display Template] (Ctrl + F1)
- [Edit] ⇨ [Mark Template (Left)] / [Mark Template (Right)] (Ctrl + Alt + ← / →)

Collapsing statements

Collapse the control statement (⊞IF...END_IF;) by clicking the icon (⊟) displayed on the left side of the control statement.

Entering comment

Enter a comment in the ST program.

Operating procedure

```
⊟IF  wValue0 < 0  THEN
     bFlag0 := TRUE;  ON
     ELSE
     bFlag0 := FALSE;
 END_IF;
```

```
⊟IF  wValue0 < 0  THEN
⊟    bFlag0 := TRUE;  (* ON *)
     ELSE
⊟    bFlag0 := FALSE; (* OFF *)
 END_IF;
```

1. Enter a comment in the arbitrary position in the program. (In the figure on the left, enter a comment after 'bFlag0 := TRUE;'.)

- A space or TAB can be inserted anywhere.

2. Enter the delimiter ('(*' and '*)') before and after the comment.

The range enclosed with the symbols is regarded as a comment.

Enter "(* OFF *)" after "FALSE" as well.

Point

In GX Works3, '/**/' and '//', which are the same symbols as C language, can be used for comment.

By adding '//' in front of the statement, the statement is regarded as a comment.

By using the following functions, the comment out/disable comment out function can be performed in the selected range.

- [Edit] ⇨ [Comment Out of Selected Range](Ctrl + Shift + S)
- [Edit] ⇨ [Disable Comment Out of Selected Range](Ctrl + Shift + D)

```
⊟IF  wValue0 < 0  THEN
⊟    bFlag0 := TRUE;  (* ON *)
 //  ELSE
 //  bFlag0 := FALSE; (* OFF *)
 END_IF;
```

Range that statement is regarded as a comment by the line unit.

Collapsing comments

Collapse the comment ((**)) by clicking the icon (⊟) displayed on the left side of the comment.

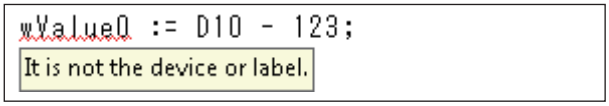
Using labels

Register a label and use it in an ST program.

Registering labels

Register a label from an ST editor.

Operating procedure

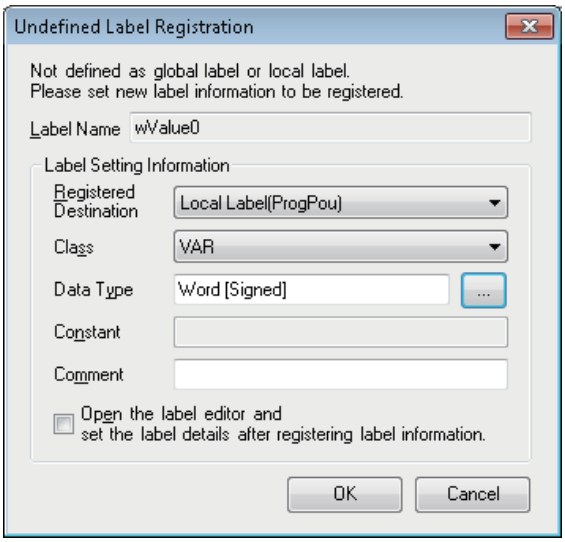


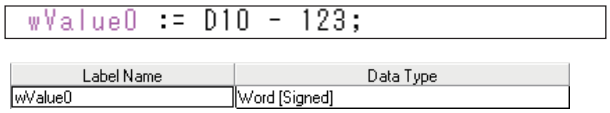
Unregistered label name is displayed as an error.

1. By pressing the F2 key on the label name, the "Undefined Label Registration" screen is displayed.

2. Set the following items to register the label, wValue0.

- Registered Destination: Local label
- Class: VAR
- Data Type: Word [Signed]





Label Name	Data Type
wValue0	Word [Signed]

The registered label is displayed with the color for labels.

3. The registered label is set on the label setting screen.

Point

The following function is used in the operation procedure above.

- [Edit] ⇨ [Register Label](F2)

The colors for each configuration element on the ST editor can be set using the following function.

- [View] ⇨ [Color and Font]

Registering labels on the label editor

Register labels on the label editor.

Operating procedure

Label Name	Data Type
wValue0	Word [Signed]
bFlag0	Bit

```
IF  wValue0 < 0  THEN
    bFlag0 := TRUE;  (* ON *)
    ELSE
    bFlag0 := FALSE; (* OFF *)
END_IF;
```

1. Display the "Local Label Setting" screen.
2. Set the following items to register labels.
   - Label Name: bFlag0
   - Data Type: Bit
   - Class: VAR (automatically set)
3. The registered label is displayed with the color for labels on the ST editor.

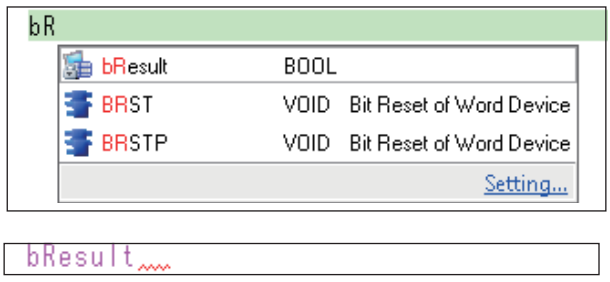
Set the following items for the local label as the labels to be used in the program example.

Label Name	Data Type	Class
wValue0	Word [Signed]	VAR
wValue1	Word [Signed]	VAR
bFlag0	Bit	VAR
bResult	Bit	VAR

Using registered labels on the ST editor

Enter a label on the ST editor by selecting a label registered on the label editor.

Operating procedure



1. Register the bit type label, bResult.
2. Enter the first two letters ,"bR". The corresponding data such as registered label names are displayed in a list.
3. Select a label name by pressing the ↓ key and press the Enter key to enter the selected label name.

Creating functions and function blocks

Create a new POU (function and function block) to be used in a program.

Creating function

Create the new data of a function in the project, and define the function program and its local label.

■Creating new data

Create the new data of a function in the project.

Operating procedure

1. Select [Project] ⇨ [Data Operation] ⇨ [New Data].
2. Set the following items on the "New Data" screen.

Data Type	Item	Description	Setting value in the program example
Function	Data Name	An identifier for calling a function.	FunPou
	Program Language	A language to describe a function program.	ST
	Result Type	Set the data type to be returned after the completion of an execution.	Bit
	Use EN/ENO	When "Yes" is selected, EN and ENO are added to the arguments. • EN: A boolean type input argument to which an execution condition is to be set. • ENO: A boolean type output argument to which an execution result is to be returned.	No
	FUN File of Add Destination	Set the name of a file to which a function is to be created is stored.	FUNFILE

Point

The setting items of the created function can be checked on the "Properties" screen.

The "Properties" screen can be displayed with the following operation.

• Select and right-click the data on the Navigation window and select [Property] from the shortcut menu.

■Setting argument and internal variable

Define the arguments and internal variables used in the function.

Set the arguments and internal variables in the local label of the function. Set the following items on the label setting screen.

Label Name	Data Type	Class
i_wValue	Word [Signed]	VAR_INPUT
bFlag	Bit	VAR

Point

A local label of which class is set to "VAR_INPUT" is an input argument.

A local label of which class is set to "VAR_OUTPUT" is an output argument.

The order of arguments when executing a function call will be the order defined in the local label setting.

Creating function block

Create the new data of a function block in the project, and define the function block program and its local label.

■Creating new data

Create new data of function block in the project.

Operating procedure

1. Select [Project] ⇨ [Data Operation] ⇨ [New Data].
2. Set the following items on the "New Data" screen.

Data Type	Item	Description	Setting value in the program example
Function Block	Data Name	A data type name of a function block to be defined.	FbPou
	Program Language	A language to describe a function block program.	ST
	Use EN/ENO	When "Yes" is selected, EN and ENO are added to the arguments. • EN: A boolean type input argument to which an execution condition is to be set. • ENO: A boolean type output argument to which an execution result is to be returned.	No
	FB Type	Set the conversion type of a function block. Macro type or subroutine type can be selected.	Subroutine Type
	FB File of Add Destination	Set the name of a file to which a function block is to be created is stored.	FBFILE

Point

The setting items of the created function block can be checked on the "Properties" screen.
The "Properties" screen can be displayed with the following operation.
• Select and right-click the data on the Navigation window and select [Property] from the shortcut menu.

■Setting argument and internal variable

Define the arguments and internal variables to be used in the function block.
Set the arguments and internal variables in the local label of the function block. Set the following items on the label setting screen.

Label Name	Data Type	Class
i_wValue	Word [Signed]	VAR_INPUT
o_wValue	Word [Signed]	VAR_OUTPUT

Point

A local label of which class is set to "VAR_INPUT" is an input argument.
A local label of which class is set to "VAR_OUTPUT" is an output argument.

Entering function

Enter a function.

Point

A function with a return value can be used as an expression. (Function call expression)
A function with no return value must be described as a call statement. (Function call statement)

```
FunPou(wValue0);  ← Function call statement
```

Entering function call expression to assignment statement

■Assigning return value to variable

To assign the return value to the variable, describe a function call statement to the right side of the assignment statement.

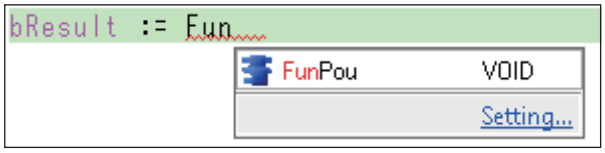
```
bResult :=
```

1. Enter a variable to which a returned value is to be assigned.
2. Enter the assignment symbol, ":=".

■Entering function name

Select and enter a function name from the undefined functions.

Operating procedure




```
bResult := FunPou
```

1. Enter the first three letters ,"Fun". The corresponding functions are displayed in a list.
2. Select a label name by pressing the ↓ key and press the Enter key to enter the selected function name.

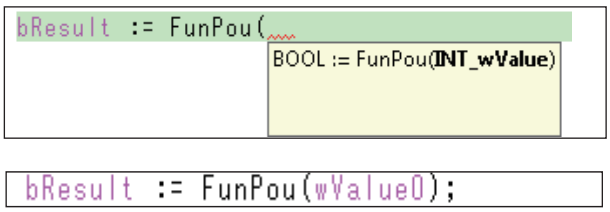
Point

Instructions can also be entered from the list. (☞ Page 41 Describing Instructions)
By pressing the F1 key on the instruction, the details of the instruction can be checked on the e-Manual Viewer.
(To check the instructions, the files of the corresponding programming manuals are required to be registered to e-Manual Viewer.)

■Entering arguments

Enter the argument in accordance with the tooltip.

Operating procedure




```
bResult := FunPou(wValue0);
```

1. By entering "(", the format of the instruction is displayed in the tooltip.
2. Enter the argument in accordance with the tooltip.

Delimit the multiple argument using commas ','.

3. Enter ")" at the end of the argument.
4. Enter ";" which indicates the end of the call statement.

Point

For the defined function, the arguments can also be inserted using the template.
By pressing the Ctrl + F1 keys, the template of the selected function name is displayed.

```
bResult := FunPou( ?INT_wValue? );
```

The argument can be selected by the following functions.

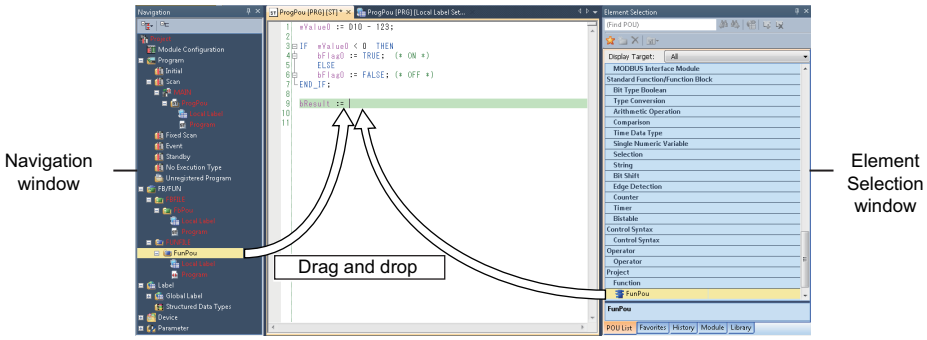
- [Edit] ⇨ [Display Template](Ctrl + F1)
- [Edit] ⇨ [Mark Template (Left)]/[Mark Template (Right)](Ctrl + Alt + ←/→)

Selecting functions from the Navigation window or the Element Selection window

Insert a function by selecting it from the Navigation window or the Element Selection window.

Point

Insert a function from the Navigation window or the Element Selection window by dragging or dropping it to the ST editor.



Entering function block

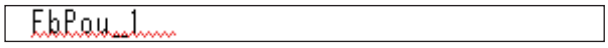
Enter a function block.

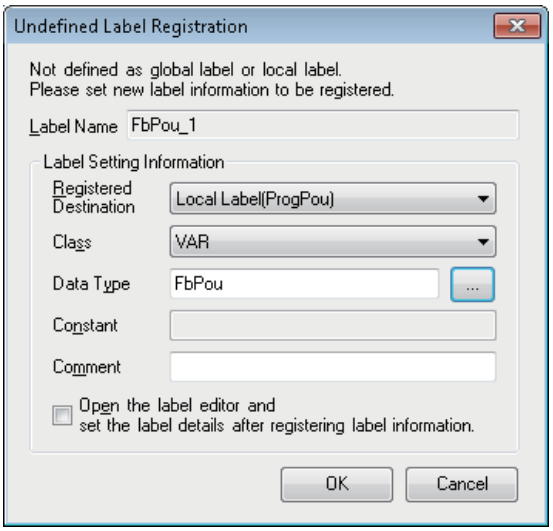
Entering function block call statement

■Entering instance of function block

Enter the defined function block.

Operating procedure





1. Enter the instance name, "FbPou_1" (example).
2. By pressing the F2 key on the instance name, the "Undefined Label Registration" screen is displayed.
3. Set the following items and register the instance.
   - Registered Destination: Local label
   - Class: VAR
   - Data Type: FbPou

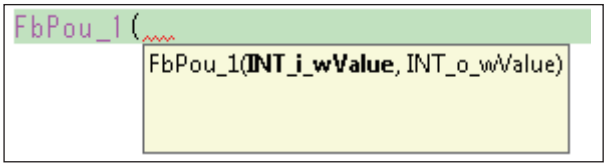
   By selecting "Function Block" in "Type Category" at data type selection, the defined function block can be selected.

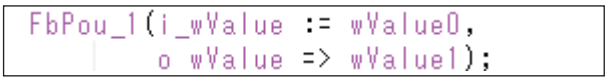
The registered instance name is displayed with the color for labels.

■Entering arguments

Enter the argument in accordance with the tooltip.

Operating procedure





1. By entering "(", the format of function block is displayed in the tooltip.
2. Enter the argument in accordance with the tooltip.

Delimit the multiple argument using commas ','.

3. Enter ")" at the end of the argument.
4. Enter ";" which indicates the end of the call statement.

Point

For the defined function block, the arguments can also be inserted using the template.
By pressing the Ctrl+F1 keys, the template of the selected instance name is displayed.

```
FbPou_1(i_wValue:= ?INT? ,o_wValue=> ?INT? );
```

The argument can be selected by the following functions.

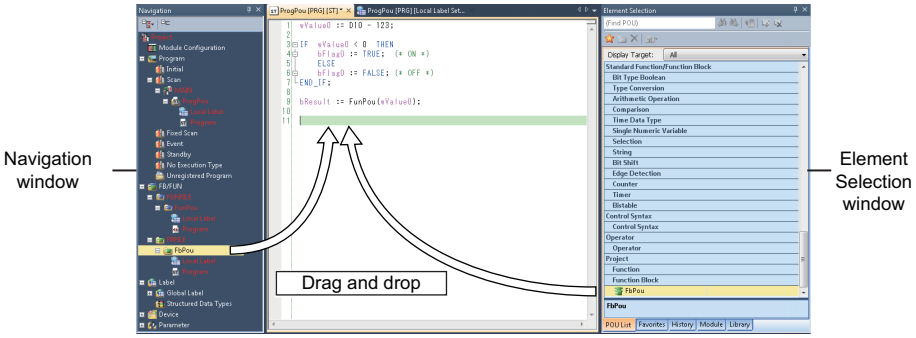
- [Edit] ⇨ [Display Template](Ctrl+F1)
- [Edit] ⇨ [Mark Template (Left)]/[Mark Template (Right)](Ctrl+Alt+←/→)

Selecting function blocks from the Navigation window or the Element Selection window

Insert a function block by selecting it from the Navigation window or the Element Selection window.

Point

Insert a function block from the Navigation window or the Element Selection window by dragging or dropping it to the ST editor.



6.4 Converting and Debugging Programs

The created program is required to be converted to the code which can be executed in the CPU module of a programmable controller (execution program).
An illegal program is checked at the conversion. Modify the program according to the displayed message.

Converting programs

Convert the created program to the executable code.

Operating procedure

1. Execute [Convert] ⇨ [Convert] ([F4]).

Point

By performing the conversion ([F4]), only the added or changed program is converted.
When converting all programs including the converted programs, perform the following function.
• [Convert] ⇨ [Rebuild All] ([Shift]+[Alt]+[F4])

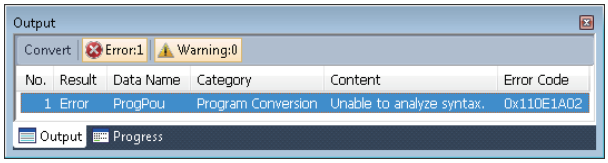
Checking error/warning

An illegal program is checked at the conversion, and an error/warning message is displayed.
Convert the following program as an example.

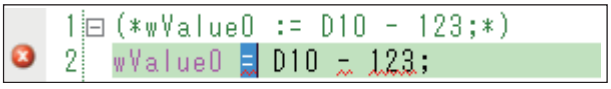
Program example

```
1  (*wValue0 := D10 - 123;*)
2   wValue0 = D10 - 123;          <-- The assignment symbol is different.
3
4  IF  wValue0 < 0  THEN
5      bFlag0 := TRUE;  (* ON *)
6      ELSE
7      bFlag0 := FALSE; (* OFF *)
8  (*END_IF;*)
9   END_IF                        <-- ';' does not exist at the end of statement.
10
11 (*bResult := FunPou( wValue0 );*)
12  FunPou( wValue0 );            <-- A return value is not used.
13  FbPou_1(i_wValue := wValue0,
14     (*  o_wValue => wValue1);*)
15         o_wValue := wValue1);  <-- The output variable is ':='.
```

Operating procedure



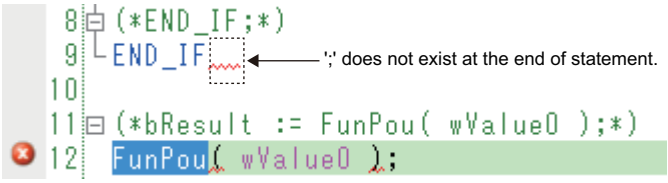
1. Double-click the error/warning message displayed on the Output window.



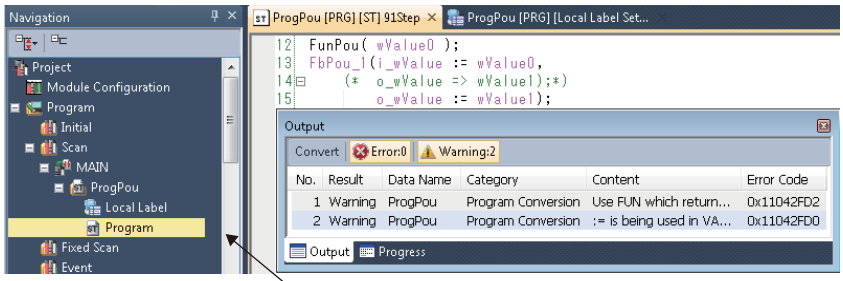
2. The cursor is moved to the corresponding error location. Modify the program according to the message.

Point

If the program cannot be analyzed, check the statement above and below.
An error location is underlined.

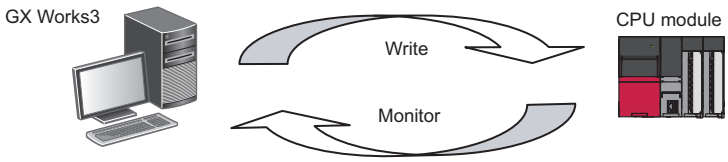


If only a warning occurs, the conversion is completed.



6.5 Checking Execution on CPU Module

Execute the converted execution program by writing it to the CPU module of a programmable controller. Check if the program is running properly by monitoring the running program.



Executing programs in the programmable controller

The following shows the procedure to execute an execution program in a CPU module.

Operating procedure

1. Connect a personal computer to the CPU module, and set the connection target in the engineering tool (GX Works3).
2. Change the operation status of the CPU module to 'STOP'.
3. Select [Online] ⇨ [Write to PLC] to write "Program".

- Write "Parameter" according to change of the system or device settings.
- When using global labels, write "Global Label Setting".
- When using devices/labels of which initial values are set, write "Local Label Initial Value", "Global Label Initial Value", or "Device Initial Value".

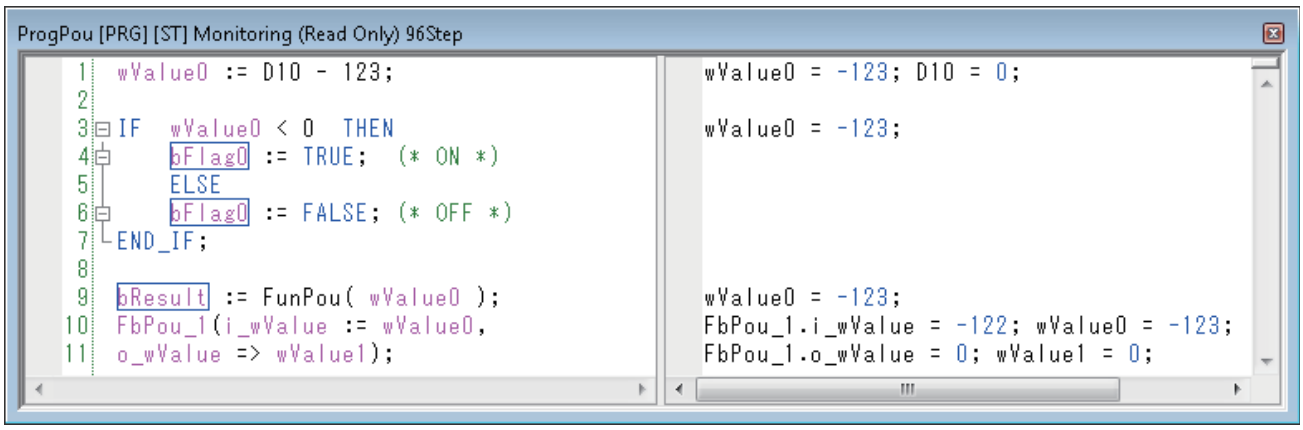
4. Reset the CPU module.
5. Change the operation status of the CPU module to 'RUN'.

Checking the running program

The following shows how to check the running program on the program editor.

Operating procedure

Select [Online] ⇨ [Monitor] ⇨ [Start Monitoring](F3)/[Stop Monitoring](Alt+F3).



Display of current values

The following shows the display of the monitoring value on the ST editor.

■Bit type

The monitoring values of bit types are displayed on the program as follows;

- TRUE: bFlag0
- FALSE: bFlag0

■Other than bit type

The monitoring values other than bit types are displayed on the right side of the split window.

Point

Place the cursor on the device/label name to display a monitoring value on the tooltip.

Changing current values

The current value of the devices and labels can be changed with the following method.

Operating procedure

1. Select the device/label of which current value is to be changed on the ST editor.
2. Click the Shift+Enter keys. (Or double-click while pressing the Shift key)

The devices/labels other than bit type are registered to the Watch window. Change the current value on the Watch window.

■Changing current values on the Watch window

The current values of devices/labels registered to the Watch window can be changed by the following method.

Operating procedure

1. Select [Online] ⇨ [Watch] ⇨ [Start Watching](Shift+F3)/[Stop Watching](Shift+Alt+F3).
2. Enter a value to which "Current Value" is to be changed directly while monitoring.

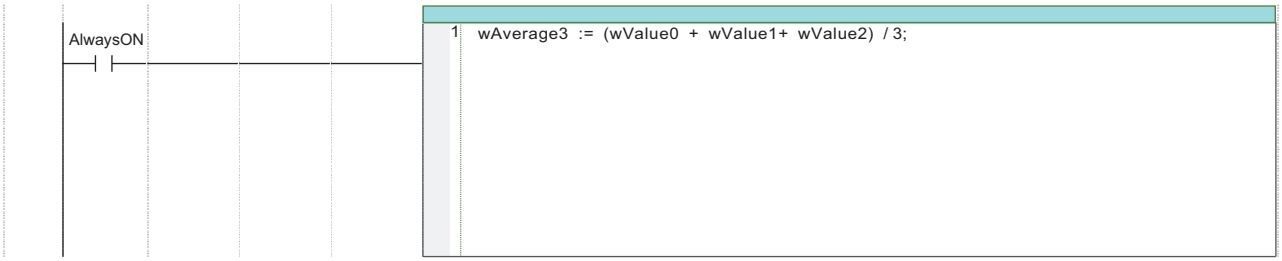
Point

Devices/labels can be registered to the Watch window by any of the following operations.

- Drag and drop the devices/labels from the ST editor to the Watch window.
- [Tool] ⇨ [Options] ⇨ "Monitor" ⇨ "ST Editor" ⇨ "Setting for Automatic Registration to Watch Window"

6.6 Inserting ST Program in Ladder Program (Inline structured text)

When describing a part of a ladder program using an ST program, use the Inline structured text function. An ST program can be described instead of the ladder instructions by using an inline structured text.



Operating procedure

1. Select [Edit] ⇨ [Inline Structured Text] ⇨ [Insert Inline Structured Text Box](Ctrl+B).

2. Edit the ST program in the inline structured text box.

For details on how to edit the program in an inline structured text box is the same as that of the ST editor.

3. Convert the ladder program.

Inline structured text is compiled (converted) as a part of the ladder program.

Point

Enter 'STB' on the ladder entry dialog to insert an inline structured text box.

PART 2 PROGRAM EXAMPLES

This part shows the examples of ST programming with simple functions.

7 OVERVIEW OF PROGRAM EXAMPLE

This chapter explains the list of program examples and their usage.

7.1 List of Program Example

In the part 2, the following program examples are described using the simple applications.

Item	Description		Reference
Calculator processing	Basic arithmetic expression Selection	Selection statement (IF) Assignment of integer, real number, and boolean value	Page 64 Initialization Program: Initialization
		Selection statement (CASE) Basic arithmetic expression	Page 65 Basic Arithmetic Operation (FUN): Calculation
		Round down/round up/rounds off of integer	Page 66 Rounding Processing (FUN): Rounding
		Operation of integer and real number Type conversion of integer and real number Hierarchization of selection statement	Page 67 Fraction Processing (FUN): FractionProcessing Page 69 Calculator Program: Calculator Page 71 Post-Tax Price Calculation: IncludingTax
Positioning processing	Exponent function Trigonometric function Structure	Trigonometric function (TAN^{-1})	Page 73 Rotation Angle Calculation (FUN): GetAngle
		Structure type argument Operation of structure Exponent function (exponentiation, square root)	Page 74 Distance Calculation (FUN): GetDistance
		Trigonometric function (COS, SIN) Structure type return value	Page 75 X, Y-Coordinate Calculation (FUN): GetXY
		Operation of integer and real number	Page 76 Command Pulse Calculation (FB): PulseNumberCalculation
		Assignment of structure Argument specification of structure	Page 77 Positioning Control: PositionControl
Sort of defective products	Array Iteration processing	Two dimensional array Operation of array Array type argument Hierarchization of iteration statement and selection statement	Page 80 Product Check (FB): ProductCheck
		Structure which includes array type member Structure array	Page 82 Sorting Product Data (FB): Assortment
		Initialization of array element Argument specification of array Iteration statement (FOR, WHILE, REPEAT)	Page 83 Product Data Management: DataManagement
Measurement of running time	Time Character string	Time type data Contact, coil, and current value of timer	Page 86 Operating Time Management: OperatingTime
		Timer Operation of boolean value	Page 87 Flicker Timer (FB): FlickerTimer Page 88 Lamp ON/OFF: LampOnOff
		Clock data (INT type array data)	Page 89 Conversion from Sec. to Hour/Min/Sec: SecondsToTimeArray
		Time type data String data	Page 90 Conversion from Time to String: TimeToString

7.2 Applying Program Example in GX Works3

This manual shows the program examples to be created in the Structured Text language using GX Works3.

Considerations

The program examples described in this manual do not guarantee the actual operation.
When applying program examples to the actual system, make sure to match the program example with the system and its required operation.
Assign the devices for the labels used in the program example according to the equipment to be used as necessary.
Set the parameters in accordance with the equipment or devices to be used as necessary.

Application procedure of sample program

Apply the sample program by the following procedure.

1. Set the global labels on the label editor. (☞ Page 48 Registering labels on the label editor)

- When using a structure, create a new structure definition and set the structure.

2. Create a POU.

- Set the local labels. (☞ Page 48 Registering labels on the label editor)
- Apply a program example to the program. (☞ Page 45 Copying/pasting data among editors)

When a POU defined in this manual is used in the sample program, create the function or the function block in the same project. (☞ Page 49 Creating new data)

3. Convert the program. (☞ Page 55 Converting and Debugging Programs)

Point

When reading this manual in PDF or e-Manual format, a sample program can easily be applied by copying and pasting it.

8 CALCULATOR PROCESSING (BASIC ARITHMETIC OPERATION AND SELECTION)

This chapter shows the program examples for the processing such as basic arithmetic operations and selections in the program that enables calculator processing.

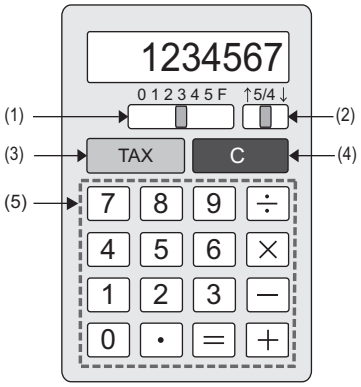
Create the following POUs in the program example.

Data name	Data type	Description	Reference
Initialization	Program block	Initializes variables when a clear instruction is issued.	Page 64 Initialization Program: Initialization
Calculation	Function	Adds, subtracts, multiplies, or divides two values.	Page 65 Basic Arithmetic Operation (FUN): Calculation
Rounding	Function	Rounds down/rounds up/rounds off a variable.	Page 66 Rounding Processing (FUN): Rounding
FractionProcessing	Function	Rounds a value right after the specified decimal place.	Page 67 Fraction Processing (FUN): FractionProcessing
Calculator	Program block	Operates values in accordance with the input.	Page 69 Calculator Program: Calculator
IncludingTax	Program block	Calculates post-tax price and amount of tax, and displays the post-tax price when a post-tax calculation instruction is issued.	Page 71 Post-Tax Price Calculation: IncludingTax

Overview of function

The following processing are performed.

- When the clear key is input, the current value (previous calculation result) and displayed value are initialized.
- The displayed value is updated according to the input of the ten key. The value after decimal point is rounded depending on the setting of the slide switches.
- When the post-tax calculation key is input, a post-tax price is displayed.

Device	Number	Name		Description
	(1)	Decimal part specification switch	0 to 5	Specifies the number of digits after a decimal point to be displayed. The values after the specified digits are processed with the setting of the Rounding processing switch.
			Floating point (F)	Displays values without rounding.
	(2)	Rounding processing switch	Round up (↑)	Adds 1 to the specified digit if the digit right after the specified digit is other than 0.
			Round off (5/4)	Rounds off the value right after the specified digit.
			Round down (↓)	Rounds down the value under the specified digits.
	(3)	Post-tax calculation key		Calculates post-tax price.
	(4)	Clear key		Initializes current value (previous calculation result) and displayed value.
	(5)	Ten key	0 to 9, decimal point (.)	Enters a numerical value.
			+, -, *, /	Specifies the type of basic arithmetic operation.
			=	Executes the specified basic arithmetic operation.

Global labels to be used

The following shows the global labels and structure definitions used in the program example.

Set the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_eDisplayValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Display value
G_eLastValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Current value (Previous calculation result)
G_wSwitch1	Word [Signed]	VAR_GLOBAL	—	Setting value of switch (0 to 5: Number of decimal part, 6: Floating point)

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_wSwitch2	Word [Signed]	VAR_GLOBAL	—	Setting value of switch (0: Round down, 1: Round up, 2: Round off)
G_bTax	Bit	VAR_GLOBAL	—	Post-tax calculation key input
G_bClear	Bit	VAR_GLOBAL	—	Clear-key input
G_wTenKey	Word [Signed]	VAR_GLOBAL	—	Ten-key input (0 to 9: Numerical value, 10: Decimal point, 11 to 14: Basic arithmetic operation, 15: =)
G_wOperation	Word [Signed]	VAR_GLOBAL	—	Operation type
G_eDecimal	FLOAT [Single Precision]	VAR_GLOBAL	Initial Value: 0	Operation for decimal part

■Structure

Do not use.

8.1 Initialization Program: Initialization

This program initializes variables when a clear instruction is issued.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	Initialization	ST	Initialization

Program example

```
(* Initialize the variables when a clear instruction is issued. *)
IF G_bClear THEN
    G_eDisplayValue := 0.0;
    G_eLastValue := 0.0;
    G_wOperation := 0;
    G_eDecimal := 0.0;
    G_bClear := FALSE;
END_IF;
```

Point

The values of each variable at first execution depend on the label settings or the setting of the assigned devices.

Global labels can be used in all programs in the project.

Statements in an IF statement are executed when its conditional expression is TRUE. It is not executed when the conditional expression is FALSE.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_bClear	Bit	VAR_GLOBAL	—	Clear-key input
G_eDisplayValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Display value
G_eLastValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Current value (Previous calculation result)
G_wOperation	Word [Signed]	VAR_GLOBAL	—	Operation type
G_eDecimal	FLOAT [Single Precision]	VAR_GLOBAL	Initial Value: 0	Operation for decimal part

■Local label

Do not use.

POU

Do not use.

8.2 Basic Arithmetic Operation (FUN): Calculation

This function performs basic arithmetic operation for two values according to the specified operation type.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	Calculation	ST	FLOAT [Single Precision]	No	Basic arithmetic operation

Program example

```
(* Add, subtract, multiply, or divide value 2 to/from value 1. *)
CASE i_wOperation OF
    1: (* Operation type is addition: Operation result = Value 1 + Value 2 *)
    Calculation := i_eValue1 + i_eValue2;
    2: (* For subtraction *)
    Calculation := i_eValue1 - i_eValue2;
    3: (* For multiplication *)
    Calculation := i_eValue1 * i_eValue2;
    4: (* For division *)
    IF i_eValue2 = 0.0 THEN (* If the value 2 is 0, do not operate. *)
        Calculation := i_eValue1;
        ELSE
        Calculation := i_eValue1 / i_eValue2;
    END_IF;
END_CASE;
```

Point

CASE statement selects an execution statement depending on the condition of the integer value. The statement which does not match the condition is not executed.
Selection statement (IF statement, CASE statement) can be hierarchized.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eValue1	FLOAT [Single Precision]	VAR_INPUT	—	Value 1
i_wOperation	Word [Signed]	VAR_INPUT	—	Operation type
i_eValue2	FLOAT [Single Precision]	VAR_INPUT	—	Value 2

■Result type

Identifier	Data Type	Description
Calculation	FLOAT [Single Precision]	Operation result

POU

Do not use.

8.3 Rounding Processing (FUN): Rounding

This function rounds down/rounds up/rounds off the first digit of an integer variable.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	Rounding	ST	Double Word [Signed]	No	Rounding processing

Program example

```
(* Round the first digit with the specified rounding method. *)
CASE i_wType OF
    0: (* Round down *)
    Rounding := i_dValue / 10 * 10; (* Round down the first decimal place *)
    1: (* Round up *)
    Rounding := (i_dValue + 9) / 10 * 10; (* Round up the first decimal place *)
    2: (* Round off *)
    Rounding := (i_dValue + 5) / 10 * 10; (* Round off the first decimal place *)
    ELSE (* Input value is returned when the value is other than specified value. *)
    Rounding := i_dValue;
END_CASE;
```

Point

For the division for an integer, the values after decimal point are rounded down.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_wType	Word [Signed]	VAR_INPUT	—	Rounding method (0: Round down, 1: Round up, 2: Round off)
i_dValue	Double Word [Signed]	VAR_INPUT	—	Input value

■Result type

Identifier	Data Type	Description
Rounding	Double Word [Signed]	Operation result

POU

Do not use.

8.4 Fraction Processing (FUN): FractionProcessing

This function rounds a value right after the specified decimal place.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	FractionProcessing	ST	FLOAT [Single Precision]	No	Fraction processing

Program example

```
(* Round a value right after the specified decimal place. *)
(* If the specified digit is out of the range, return the input value. *)
IF (i_wDigits <= 0) OR (i_wDigits > c_wMAX) THEN
    FractionProcessing := i_eValue;
    RETURN; (* End processing *)
END_IF;
(* Move the decimal point in order that the specified value becomes the first place. *)
wDigits := i_wDigits + 1;
eValue := i_eValue * (10.0 ** wDigits) - 0.5;
(* Perform the specified rounding processing for the first digit of an integer value. *)
dValue := Rounding(i_wType, REAL_TO_DINT(eValue));
(* Change the value to the real number, and move the decimal point back to its original position. *)
FractionProcessing := DINT_TO_REAL(dValue) / (10.0 ** wDigits);
```

Point

When converting a real number type to an integer type, use the type conversion function such as REAL_TO_DINT.

The call expression of the type conversion function can be described for the terms and arguments of an operational expression.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eValue	FLOAT [Single Precision]	VAR_INPUT	—	Input value
i_wDigits	Word [Signed]	VAR_INPUT	—	Number of decimal part to be displayed (Decimal part: 0 to 5 digits)
i_wType	Word [Signed]	VAR_INPUT	—	Rounding method (0: Round down, 1: Round up, 2: Round off)
c_wMAX	Word [Signed]	VAR_CONSTANT	Constant: 5	Maximum value of decimal place specification
wDigits	Word [Signed]	VAR	—	Digits of decimal part for processing (Decimal part: 1 to 6 digits)
eValue	FLOAT [Single Precision]	VAR	—	Real number value (for internal calculation)
dValue	Double Word [Signed]	VAR	—	Integer value (for internal calculation)

■Result type

Identifier	Data Type	Description
FractionProcessing	FLOAT [Single Precision]	Operation result

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
Rounding	Function	Rounds down/rounds up/rounds off a variable.	Page 66 Rounding Processing (FUN): Rounding

Data name	Data type	Description	Reference
REAL_TO_DINT	Standard function	Conversion of REAL type → DINT type Converts a value from REAL type data to DINT type data.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
DINT_TO_REAL	Standard function	Conversion of DINT type → REAL type Converts a value from DINT type data to REAL type data.	

8.5 Calculator Program: Calculator

This program updates the displayed value according to the input of the ten key. The values after decimal point are rounded depending on the setting of the Decimal place digit specification switch.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	Calculator	ST	Calculator

Program example

```
(* The processing is performed depending on the ten-key input. *)
IF G_wTenKey <> c_wNONE THEN
    CASE G_wTenKey OF
        0..9 : (* For number-key input (0 to 9) *)
        (* Add the input numeric value to the end of the display value. *)
        IF G_eDecimal = 0.0 THEN
            (* For integer part *)
            G_eDisplayValue := (G_eDisplayValue * 10) + G_wTenKey;
            ELSE
            (* For after decimal point *)
            G_eDisplayValue := G_eDisplayValue + (G_eDecimal * G_wTenKey);
            G_eDecimal := G_eDecimal * 0.1;
        END_IF;
        10: (* For input of decimal point key *)
        G_eDecimal := 0.1;
        11..14: (* For input of addition, subtraction, multiplication, or division key (11 to 14) *)
        (* Retain the operation type *)
        G_wOperation := G_wTenKey - 10;
        (* Move the display value to the previous operation value and then reset the displayed value *)
        G_eLastValue := G_eDisplayValue;
        G_eDisplayValue := 0.0;
        G_eDecimal := 0.0;
        15: (* For equal-key input *)
        (* Add, subtract, multiply, or divide the displayed value to/from the current value. *)
        G_eLastValue := Calculation(G_eLastValue, G_wOperation, G_eDisplayValue);
        (* Assign the rounding result to the display value. *)
        G_eDisplayValue := FractionProcessing(G_eLastValue, G_wSwitch1, G_wSwitch2);
        G_wOperation := 0; (* Clear the operation type *)
        G_eDecimal := 0.0;
    END_CASE;
    (* Clear the key input *)
    G_wTenKey := c_wNONE;
END_IF;
```

Point

If the data type of the operators on the right side and left side differ, the data type of the operation result will be a bigger data type. (The operation result between INT type (Word [Signed]) and REAL type (FLOAT [Single Precision]) will be the REAL type (FLOAT [Single Precision]).)

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_wTenKey	Word [Signed]	VAR_GLOBAL	—	Ten-key input (0 to 9: Numerical value, 10: Decimal point, 11 to 14: Basic arithmetic operation, 15: =)
G_eDecimal	FLOAT [Single Precision]	VAR_GLOBAL	Initial Value: 0	Operation for decimal part
G_wOperation	Word [Signed]	VAR_GLOBAL	—	Operation type
G_eDisplayValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Display value
G_eLastValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Current value (Previous calculation result)
G_wSwitch1	Word [Signed]	VAR_GLOBAL	—	Setting value of switch (0 to 5: Number of decimal part, 6: Floating point)
G_wSwitch2	Word [Signed]	VAR_GLOBAL	—	Setting value of switch (0: Round down, 1: Round up, 2: Round off)

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
c_wNONE	Word [Signed]	VAR_CONSTANT	Constant: -1	No ten-key input

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
Calculation	Function	Adds, subtracts, multiplies, or divides a current value to an input value.	Page 65 Basic Arithmetic Operation (FUN): Calculation
FractionProcessing	Function	Calculates a value by selecting round down/round up/ round off at the specified decimal place.	Page 67 Fraction Processing (FUN): FractionProcessing

8.6 Post-Tax Price Calculation: IncludingTax

This program calculates post-tax price and amount of tax, and displays the post-tax price when a post-tax calculation instruction is issued.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	IncludingTax	ST	Post-tax price calculation

Program example

```
(* When a post-tax calculation instruction is issued, display the post-tax price. *)
IF G_bTax THEN
    (* Calculate the amount of tax. (Calculate the values after decimal point as well.) *)
    eTaxAmount := G_eDisplayValue * c_eTaxRate / 100.0;
    (* Calculate the post-tax price. (Calculate the values after decimal point as well.) *)
    G_eLastValue := G_eDisplayValue + eTaxAmount;
    (* Set the post-tax price as a display value. (Values after decimal point are rounded off) *)
    dPrice := REAL_TO_DINT(G_eLastValue);
    G_eDisplayValue := DINT_TO_REAL(dPrice);
    (* Clear the key input *)
    G_bTax := FALSE;
END_IF;
```

Point

When converting a real number type to an integer type, use the type conversion function such as REAL_TO_DINT.

The data after type conversion using REAL_TO_DINT will be the value of which first decimal place is rounded off of REAL type data.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_eDisplayValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Display value
G_eLastValue	FLOAT [Single Precision]	VAR_GLOBAL	—	Current value (Previous calculation result)
G_bTax	Bit	VAR_GLOBAL	—	Post-tax calculation key input

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
c_eTaxRate	FLOAT [Single Precision]	VAR_CONSTANT	Constant: 8.0	Rate of tax (%)
eTaxAmount	FLOAT [Single Precision]	VAR	—	Amount of tax
dPrice	Double Word [Signed]	VAR	—	Post-tax price

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
REAL_TO_DINT	Standard function	Converting REAL to DINT Converts a value from REAL type data to DINT type data.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
DINT_TO_REAL	Standard function	Converting DINT to REAL Converts a value from DINT type data to REAL type data.	

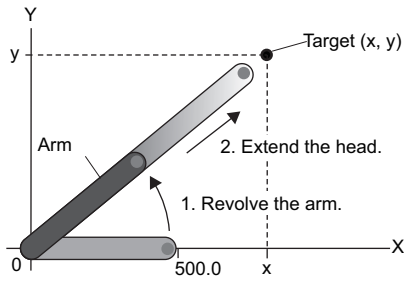
9 POSITIONING PROCESSING (EXPONENT FUNCTION, TRIGONOMETRIC FUNCTION AND STRUCTURE)

This chapter shows the examples of programs for calculating position on an X, Y-coordinate and amount of rotation of a device, and data processing using an exponent function, arithmetic operation of trigonometric function, or structure.
Create the following POUs in the program example.

Data name	Data type	Description	Reference
GetAngle	Function	Calculates a rotation angle for the target X, Y-coordinate.	Page 73 Rotation Angle Calculation (FUN): GetAngle
GetDistance	Function	Calculates the distance between two points from X, Y-coordinate.	Page 74 Distance Calculation (FUN): GetDistance
GetXY	Function	Calculates X, Y-coordinate from radius and angle.	Page 75 X, Y-Coordinate Calculation (FUN): GetXY
PulseNumberCalculation	Function block	Calculates number of command pulses to a motor from the movement amount.	Page 76 Command Pulse Calculation (FB): PulseNumberCalculation
PositionControl	Program block	Calculates the angle and length of the arm to be moved from the target X, Y-coordinate.	Page 77 Positioning Control: PositionControl

Overview of function

This function calculates the rotation angle of the arm and number of command pulses of the motor for adjusting the length of the arm in order to move the head of the arm to the target X, Y-coordinate with the following procedure.



1. Rotate the arm towards the target specified in X, Y-coordinate.
2. Extend the head of the arm towards the target using a stepping motor.

Global labels to be used

The following shows the global labels and structure definitions used in the program example.
Set the following items in GX Works3.

■Global label

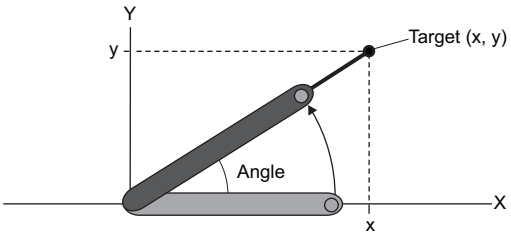
Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_stTarget	stPosition	VAR_GLOBAL	—	Target (X, Y-coordinate)
G_stArm	stPosition	VAR_GLOBAL	—	Head of the arm (X, Y-coordinate)
G_eAngle	FLOAT [Single Precision]	VAR_GLOBAL	—	Rotation angle (Degree)
G_ePulses	FLOAT [Single Precision]	VAR_GLOBAL	—	Number of command pulses of the motor
G_bOneScanOnly	Bit	VAR_GLOBAL	Assign (Device/Label): SM402	Turn ON for only 1 scan after RUN

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stPosition	eXcoordinate	FLOAT [Single Precision]	0.0	X-coordinate
	eYcoordinate	FLOAT [Single Precision]	0.0	Y-coordinate

9.1 Rotation Angle Calculation (FUN): GetAngle

This function calculates the rotation angle (0 to 180) for the target X, Y-coordinate.



Angle (radian)

$$\theta = \mathrm{ATAN}\left(\frac{y}{x}\right)$$

Angle (degree)

When x = 0: = 90

When x > 0: $= \theta \times \frac{180}{\pi}$

When x < 0: $= \theta \times \frac{180}{\pi} + 180$

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	GetAngle	ST	FLOAT [Single Precision]	Yes	Rotation angle calculation

Program example

```
(* Calculate radian from X, Y-coordinate, and convert the radian to degree. *)
(* If X-coordinate is 0, 90 degree (End processing without calculation.) *)
IF i_eXcoordinate = 0.0 THEN
    GetAngle := 90.0;
    ENO := TRUE;
    RETURN; (* End processing. *)
END_IF;
(* Calculate radian from X-coordinate and Y-coordinate. *)
eAngleRad := ATAN(i_eYcoordinate / i_eXcoordinate); (* Angle (radian) rad = ATAN (Y-coordinate/X-coordinate) *)
(* Error end if the data which cannot be handled with ATAN instruction is included *)
IF SD0 = H3402 THEN (* Error code : 3402H (Operation error) *)
    ENO := FALSE;
    RETURN; (* End processing *)
    ELSE
    ENO := TRUE;
END_IF;
(* Convert radian to degree. *)
GetAngle := eAngleRad * 180.0 / c_ePi; (* Angle (degree) = Angle (radian) rad * 180/pi *)
(* If X-coordinate is negative, add 180 degree. *)
IF i_eXcoordinate < 0.0 THEN
    GetAngle := GetAngle + 180.0;
END_IF;
```

Point

DEG instruction can be used for the conversion from single-precision real number radian to angle.
SD0 is a device for error check. The latest error code will be stored to it.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eXcoordinate	FLOAT [Single Precision]	VAR_INPUT	—	X-coordinate
i_eYcoordinate	FLOAT [Single Precision]	VAR_INPUT	—	Y-coordinate
eAngleRad	FLOAT [Single Precision]	VAR	—	Angle (radian)
c_ePi	FLOAT [Single Precision]	VAR_CONSTANT	Constant: 3.14159	Circular constant

■Result type

Identifier	Data Type	Description
GetAngle	FLOAT [Single Precision]	Angle (degree): 0 to 180

POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
ATAN	Standard function	TAN^{-1} operation Outputs the arc tangent value (TAN^{-1}) of an input value.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

9.2 Distance Calculation (FUN): GetDistance

This function calculates the distance between two points from X, Y-coordinate.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	GetDistance	ST	FLOAT [Single Precision]	No	Distance calculation

Program example

```
(* Calculate distance between two points from X, Y-coordinate. *)
GetDistance := SQRT((i_stPosition0.eXcoordinate - i_stPosition1.eXcoordinate) ** 2.0
          + (i_stPosition0.eYcoordinate - i_stPosition1.eYcoordinate) ** 2.0);
```

Point

The argument of a structure can be specified for a function/function block.

Structure member can be specified to the operand (operation target value) of the operational experssion, and left side and right side of the iteration statement.

**' is the operator of an exponentiation.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_stPosition0	stPosition	VAR_INPUT	—	Position 0 (X, Y-coordinate)
i_stPosition1	stPosition	VAR_INPUT	—	Position 1 (X, Y-coordinate)

■Result type

Identifier	Data Type	Description
GetDistance	FLOAT [Single Precision]	Distance between two points

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stPosition	eXcoordinate	FLOAT [Single Precision]	0.0	X-coordinate
	eYcoordinate	FLOAT [Single Precision]	0.0	Y-coordinate

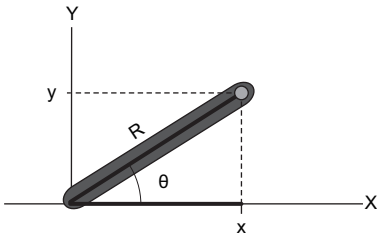
POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
SQRT	Standard function	Square root Outputs the square root of an input value.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

9.3 X, Y-Coordinate Calculation (FUN): GetXY

This function calculates X, Y-coordinate from the specified radius and angle.



Angle (radian)

$\theta = \text{Angle (degree)} \times \frac{\pi}{180}$

X,Y coordinate

$x = \text{radius } R \times \cos(\theta)$

$y = \text{radius } R \times \sin(\theta)$

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Result Type	EN/ENO	Title
Function	GetXY	ST	stPosition	No	X, Y-coordinate calculation

Program example

```
(* Convert the unit from degree to radian. *)
eAngleRad := i_eAngle * c_ePi / 180.0; (* Angle (radian) rad = Angle (degree) * pi/180 *)
(* Calculate X, Y-coordinate. *)
GetXY.eXcoordinate := i_eRadius * COS(eAngleRad); (* X-coordinate = Radius * COS(rad) *)
GetXY.eYcoordinate := i_eRadius * SIN(eAngleRad); (* Y-coordinate = Radius * SIN (rad) *)
```

Point

A structure type can be specified to the Result Type of a function.

RAD instruction can be used for the conversion from single-precision real number angle to radian.

Variable

Define the labels by setting the following items in GX Works3. Result type can be set in the property of the function.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eRadius	FLOAT [Single Precision]	VAR_INPUT	—	Radius (mm)
i_eAngle	FLOAT [Single Precision]	VAR_INPUT	—	Angle (degree): 0 to 180
eAngleRad	FLOAT [Single Precision]	VAR	—	Angle (radian)
c_ePi	FLOAT [Single Precision]	VAR_CONSTANT	Constant: 3.14159	Circular constant

■Result type

Identifier	Data Type	Description
GetXY	stPosition	X, Y-coordinate

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stPosition	eXcoordinate	FLOAT [Single Precision]	0.0	X-coordinate
	eYcoordinate	FLOAT [Single Precision]	0.0	Y-coordinate

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
COS	Standard function	COS operation Outputs COS (cosine) of an input value.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
SIN	Standard function	SIN operation Outputs SIN (sine) of an input value.	

9.4 Command Pulse Calculation (FB): PulseNumberCalculation

This function block calculates number of command pulses to a motor from the movement amount.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	EN/ENO	Title
Function Block	PulseNumberCalculation	ST	No	Command pulse calculation

Program example

```
(* Calculate the number of command pulses. *)
IF e1round <> 0.0 THEN
    (* Number of command pulses = Motor resolution*(Target movement amount/Movement amount for one motor rotation) *)
    o_ePulses := eResolution * ( i_eDistance / e1round );
END_IF;
```

Variable

Define the labels by setting the following items in GX Works3.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eDistance	FLOAT [Single Precision]	VAR_INPUT	—	Target movement amount (mm)
o_ePulses	FLOAT [Single Precision]	VAR_OUTPUT	—	Number of command pulses of motor (pulse)
e1round	FLOAT [Single Precision]	VAR	—	Movement amount for one motor rotation (mm/rev)
eResolution	FLOAT [Single Precision]	VAR	—	Motor resolution (Pulse/rev)

Point

Set the initial values to the e1round and the eResolution in the local label setting in the used-side program.

☞ Page 78 Local label

The setting for initial values of labels does not exist for the MELSEC iQ-F series; therefore, set the initial values by using a program or a watch window of an engineering tool.

POU

Do not use.

9.5 Positioning Control: PositionControl

This program calculates the angle and length of the arm to be moved from the target X, Y-coordinate.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	PositionControl	ST	Positioning control

Program example

```
(* Calculate the rotation angle and number of command pulses of the motor for adjusting the length of the arm in order to move the head of the arm to the target X, Y-coordinate. *)
(* Set the initial value to the coordinate of the arm only at the first time after RUN. *)
IF G_bOneScanOnly THEN
    G_stArm.eXcoordinate := 500.0;
    G_stArm.eYcoordinate := 0.0;
END_IF;
(* The position of the target will be set to the global variable, G_stTarget in other program. [The program is omitted in this program example.] *)
(* Calculate the moving angle of the arm for the target X, Y-Coordinate. *)
eTargetAngle := GetAngle( TRUE, bResult1, G_stTarget.eXcoordinate, G_stTarget.eYcoordinate );
eArmAngle := GetAngle( bResult1, bResult2, G_stArm.eXcoordinate, G_stArm.eYcoordinate );
IF bResult2 THEN
    G_eAngle := eTargetAngle - eArmAngle;
    ELSE
    RETURN; (* Error end *)
END_IF;
(* Calculate the current length of the arm (distance from origin) from the X, Y-coordinate of the head of the arm. *)
eDistance := GetDistance( G_stArm, stOrigin ); (* Input variable is a function call of structure *)
(* Calculate the X, Y-coordinate of the head of the arm after the rotation. *)
G_stArm := GetXY( eDistance, G_eAngle); (* Return value is a function call of structure *)
(* Calculate the distance between two points from the X, Y-coordinate of the target and head of the arm. *)
eDistance := GetDistance( G_stTarget, G_stArm );
(* Calculate the number of command pulses to the motor for adjusting the length of the arm from the movement amount. *)
PulseNumberCalculation_1(i_eDistance := eDistance, o_ePulses => G_ePulses ); (* Function block call *)
(* Rotate the arm by specifying the angle. [The program is omitted in this program example.] *)
(* Stretch the head of the arm by specifying the number of command pulses of the motor. [The program is omitted in this program example.] *)
(* Update the coordinate of the arm. *)
G_stArm := G_stTarget;
```

Point

When creating a function and function block, whether or not to use an EN and ENO can be selected.
If FALSE is specified to EN, the processing is not executed. ENO becomes FALSE.
Specify the output valuable using '=>' for a function block instance.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_bOneScanOnly	Bit	VAR_GLOBAL	Assign (Device/Label): SM402	Turn ON for only 1 scan after RUN
G_stTarget	stPosition	VAR_GLOBAL	—	Target (X, Y-coordinate)
G_stArm	stPosition	VAR_GLOBAL	—	Head of arm (X, Y-coordinate)
G_eAngle	FLOAT [Single Precision]	VAR_GLOBAL	—	Rotation angle (degree)
G_ePulses	FLOAT [Single Precision]	VAR_GLOBAL	—	Number of command pulses of motor (pulse)

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
eTargetAngle	FLOAT [Single Precision]	VAR	—	Angle to the target (degree): 0 to 180
eArmAngle	FLOAT [Single Precision]	VAR	—	Angle of arm (degree): 0 to 180
bResult1	Bit	VAR	—	Operation result 1
bResult2	Bit	VAR	—	Operation result 2
stOrigin	stPosition	VAR	—	Origin (X, Y-coordinate = 0. 0)
eDistance	FLOAT [Single Precision]	VAR	—	Distance
PulseNumberCalculation_1	PulseNumberCalculation	VAR	Initial Value • e1round: 0.2 • eResolution: 3000.0	Calculation of number of command pulses

Point

The setting for initial values of labels does not exist for the MELSEC iQ-F series; therefore, set the initial values by using a program or a watch window of an engineering tool.

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stPosition	eXcoordinate	FLOAT [Single Precision]	0.0	X-coordinate
	eYcoordinate	FLOAT [Single Precision]	0.0	Y-coordinate

Point

In GX Works3, initial values can be set for each internal variable of function block instances.
To initialize structure type variables, set the initial values to the structure definition or create an initialization program. For a global label, initial values can be set to the devices to be assigned.

POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
GetAngle	Function	Calculates angle for the target X, Y-coordinate.	Page 73 Rotation Angle Calculation (FUN): GetAngle
GetXY	Function	Calculates X, Y-coordinate from radius and angle.	Page 75 X, Y-Coordinate Calculation (FUN): GetXY
GetDistance	Function	Calculates distance between two points from X, Y-coordinate.	Page 74 Distance Calculation (FUN): GetDistance
PulseNumberCalculation	Function block	Calculates number of command pulses to a motor from the movement amount.	Page 76 Command Pulse Calculation (FB): PulseNumberCalculation

10 SORTING OF DEFECTIVE PRODUCTS (ARRAY AND ITERATION PROCESSING)

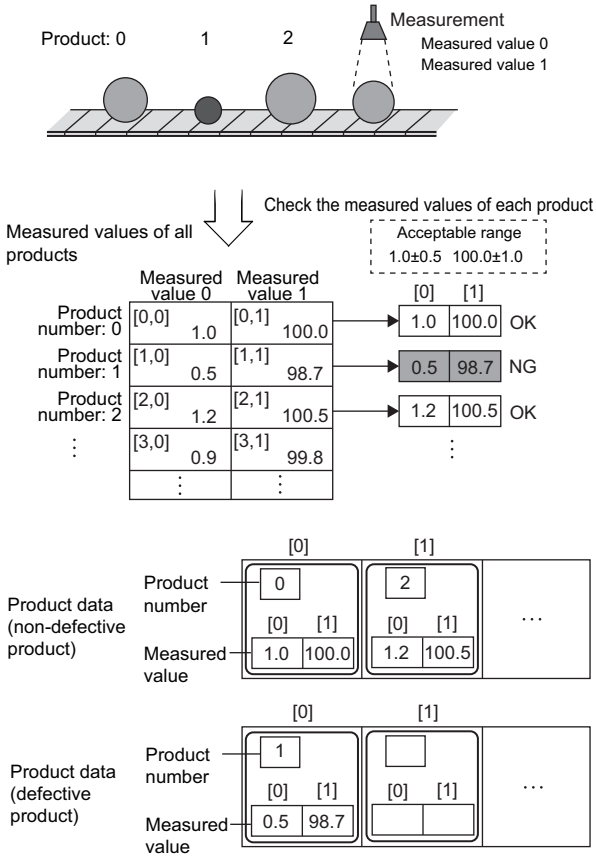
This chapter shows the examples for an iteration statement using an array in the program which organizes multiple product data.

Create the following POUs in the program example.

Data name	Data type	Description	Reference
ProductCheck	Function block	Judges the products as non-defective or defective from the product data.	Page 80 Product Check (FB): ProductCheck
Assortment	Function block	Sorts product data into non-defectives and defectives.	Page 82 Sorting Product Data (FB): Assortment
DataManagement	Program block	Organizes product data.	Page 83 Product Data Management: DataManagement

Overview of function

This function sorts product data into non-defectives and defectives by checking the data of the multiple products of which sizes and weights are measured.



1. Measure the product.

This program example processes the data of two measured values for eight products. Store the arbitrary value for the measured value for the G_eValueArray (Measured values of all products).

2. Check the measured values for each product.

- Non-defective product: A product of which all measured values are within the allowable range.
- Defective product: A product of which measured value is out of the allowable range.

3. Sort the product data into non-defectives and defectives.

4. After checking all the products, acquire the data of non-defective products and defective products.

Global labels to be used

The following shows the global labels and structure definitions used in the program example.

Set the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_eValueArray	FLOAT [Single Precision](0..7,0..1)	VAR_GLOBAL	—	Measured values for all products (Values are stored in order of product number.)

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
GC_wValueNumber	Word [Signed]	VAR_GLOBAL_CONSTANT	Constant: 2	Number of measured values for one product
GC_wTotalProduct	Word [Signed]	VAR_GLOBAL_CONSTANT	Constant: 8	Total number of products
G_stProductArray	stProduct(0..7)	VAR_GLOBAL	—	Product data (non-defective product)
G_stDefectiveArray	stProduct(0..7)	VAR_GLOBAL	—	Product data (defective product)

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stProduct	wProductNumber	Word [Signed]	—	Product number
	eValueArray	FLOAT [Single Precision](0..1)	—	Measured value

10.1 Product Check (FB): ProductCheck

This function block checks if the value is within the allowable range.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	EN/ENO	Title
Function Block	ProductCheck	ST	No	Product check

Program example

```
(* Check if the values of each element in the array are within the allowable range. *)
FOR wIndex := 0 TO (i_wValueNumber - 1) BY 1 DO
    (* Calculate the upper and lower limit from the standard value and tolerance. *)
    eMaxValue := i_eAcceptableArray[wIndex, c_wBasicSize] + i_eAcceptableArray[wIndex, c_wTolerance];
    eMinValue := i_eAcceptableArray[wIndex, c_wBasicSize] - i_eAcceptableArray[wIndex, c_wTolerance];
    (* Check the upper and lower limit. *)
    IF (eMaxValue >= i_eValueArray[wIndex]) AND (eMinValue <= i_eValueArray[wIndex]) THEN
        o_bResult := TRUE;
        ELSE
        o_bResult := FALSE;
        EXIT; (* End processing when the value out of the range exists. *)
    END_IF;
END_FOR;
```

Point

The same data processing can be performed for the multiple elements of array by combining array type data and iteration statement.
Specify the elements of each array using an operational expression as a variable of the defined data type.
Selection statement (IF statement, CASE statement) and iteration statement (FOR statement, WHILE statement, REPEAT statement) can be hierarchized.

Variable

Define the labels by setting the following items in GX Works3.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_eValueArray	FLOAT [Single Precision](0..1)	VAR_INPUT	—	Judgment value
i_eAcceptableArray	FLOAT [Single Precision](0..1,0..1)	VAR_INPUT	—	Allowable range
i_wValueNumber	Word [Signed]	VAR_INPUT	—	Number of measured values for one product
o_bResult	Bit	VAR_OUTPUT	—	Check result

Label Name	Data Type	Class	Initial Value/Constant	Comment
c_wBasicSize	Word [Signed]	VAR_CONSTANT	Constant: 0	Element number to which the standard value is to be stored.
c_wTolerance	Word [Signed]	VAR_CONSTANT	Constant: 1	Element number to which the tolerance is to be stored.
eMaxValue	FLOAT [Single Precision]	VAR	—	Upper limit of judgment
eMinValue	FLOAT [Single Precision]	VAR	—	Lower limit of judgment
wIndex	Word [Signed]	VAR	—	Element number

Point

Global data can be used in a function block.
Array type data can be specified to an input variable.

POU

Do not use.

10.2 Sorting Product Data (FB): Assortment

This function block stores product data after sorting them into non-defective products and defective products.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	EN/ENO	Title
Function Block	Assortment	ST	No	Sorting product data

Program example

```
(* Sort the product data into non-defectives and defectives depending on the check result. *)
IF i_bCheck THEN
    (* Non-defective product *)
    (* Store the measured value to the Product data (non-defective product). *)
    o_stProductArray[wTotal] := i_stProduct;
    (* Increment the number of non-defective products *)
    wTotal := wTotal + 1;
    ELSE
    (* Defective product *)
    (* Store the measured value to the Product data (defective product). *)
    o_stDefectiveArray[wDefectiveTotal] := i_stProduct;
    (* Increment the number of defective products *)
    wDefectiveTotal := wDefectiveTotal + 1;
END_IF;
(* Update yield rate *)
o_eYieldRatio := INT_TO_REAL(wTotal) / INT_TO_REAL(wTotal + wDefectiveTotal);
```

Point

The structure which includes array in a member and array of structure type can be used in ST programs.

Variable

Define the labels by setting the following items in GX Works3.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
i_bCheck	Bit	VAR_INPUT	—	Check result
i_stProduct	stProduct	VAR_INPUT	—	Checked product data
o_stProductArray	stProduct(0..7)	VAR_OUTPUT	—	Product data (non-defective product)
o_stDefectiveArray	stProduct(0..7)	VAR_OUTPUT	—	Product data (defective product)
o_eYieldRatio	FLOAT [Single Precision]	VAR_OUTPUT	—	Yield rate
wTotal	Word [Signed]	VAR	—	Number of non-defective products
wDefectiveTotal	Word [Signed]	VAR	—	Number of defective products

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stProduct	wProductNumber	Word [Signed]	—	Product number
	eValueArray	FLOAT [Single Precision](0..1)	—	Measured value

POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
INT_TO_REAL	Standard function	Convert of INT type → REAL Converts a value from INT type data to REAL type data.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/ Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

10.3 Product Data Management: DataManagement

This program stores the measured values for each product with sorting non-defective products and defective products.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	DataManagement	ST	Product data management

Program example

```
(* Check the product data and sort them into non-defectives and defectives. *)
(* Set the allowable range to be checked. *)
Check_1.i_eAcceptableArray[0,0] := 1.0; (* Standard value of Measured value 0 *)
Check_1.i_eAcceptableArray[0,1] := 0.5; (* Tolerance of Measured value 0 *)
Check_1.i_eAcceptableArray[1,0] := 100.0; (* Standard value of Measured value 1 *)
Check_1.i_eAcceptableArray[1,1] := 1.0; (* Tolerance of Measured value 1 *)
Check_1.i_wValueNumber := GC_wValueNumber; (* Number of measured values for one product *)
(* Check the measured values for all products by iteration processing. *)
REPEAT
    (* Processing each three product *)
    wDataEnd := wProductNumber + 3;
    (* If the remaining number of products are three or less, the processing is continued to the end. *)
    IF wDataEnd > GC_wTotalProduct THEN
        wDataEnd := GC_wTotalProduct;
    END_IF;
    (* Repeat judging and sorting for three products. *)
    WHILE wProductNumber < wDataEnd DO
        (* Acquire the measured values of the processing target. *)
        FOR wIndex := 0 TO (GC_wValueNumber - 1) BY 1 DO
            eValueArray[wIndex] := G_eValueArray[wProductNumber, wIndex];
        END_FOR;
        (* Judge the product as non-defective or defective from the product data. *)
        Check_1(i_eValueArray := eValueArray, o_bResult => bResult);
        (* Sort the product data into non-defectives and defectives depending on the result of the bResult (check result). *)
        stProductData.wProductNumber := wProductNumber;
        stProductData.eValueArray := eValueArray;
        Assortment_1( i_bCheck := bResult, i_stProduct := stProductData);
        (* Go to the next product number. *)
        wProductNumber := wProductNumber + 1;
    END_WHILE;
    (* End processing under the following conditions. *)
    UNTIL ((Assortment_1.o_eYieldRatio < c_eLimit) (* The yield rate is lower than the standard. *)
    OR(wProductNumber >= GC_wTotalProduct)) (* Or, number of total products exceed the product number *)
END_REPEAT;
(* Acquire the product data sorted into non-defective products and defective products. *)
G_stProductArray := Assortment_1.o_stProductArray;
G_stDefectiveArray := Assortment_1.o_stDefectiveArray;
```

Point

Different initial values cannot be set for each element of an array. Set the different values by a program.
For a function block, arguments can be specified before and after the call statement.
Selection statement (IF statement, CASE statement) and iteration statement (FOR statement, WHILE statement, REPEAT statement) can be hierarchized.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Comment
G_eValueArray	FLOAT [Single Precision](0..7,0..1)	VAR_GLOBAL	—	Measured values for all products (Values are stored in order of product number.)
GC_wValueNumber	Word [Signed]	VAR_GLOBAL_CONSTANT	Constant: 2	Number of measured values for one product
GC_wTotalProduct	Word [Signed]	VAR_GLOBAL_CONSTANT	Constant: 8	Total number of products
G_stProductArray	stProduct(0..7)	VAR_GLOBAL	—	Product data (non-defective product)
G_stDefectiveArray	stProduct(0..7)	VAR_GLOBAL	—	Product data (defective product)

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Comment
wProductNumber	Word [Signed]	VAR	—	Product number (for internal iterative processing)
wDataEnd	Word [Signed]	VAR	—	Data termination (for internal iterative processing)
wIndex	Word [Signed]	VAR	—	Element number (for internal iterative processing)
eValueArray	FLOAT [Single Precision](0..1)	VAR	—	Measured value
bResult	Bit	VAR	—	Checking result (for internal iterative processing)
stProductData	stProduct	VAR	—	Checked product data
c_eLimit	FLOAT [Single Precision]	VAR_CONSTANT	Constant: 0.8	Allowable yield rate
Check_1	ProductCheck	VAR	—	Product check processing
Assortment_1	Assortment	VAR	—	Sorting processing for product data

■Structure

Structure name	Label Name	Data Type	Initial Value	Comment
stProduct	wProductNumber	Word [Signed]	—	Product number
	eValueArray	FLOAT [Single Precision](0..1)	—	Measured value

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
ProductCheck	Function block	Judges the products as non-defective or defective from the product data.	Page 80 Product Check (FB): ProductCheck
Assortment	Function block	Sorts product data into non-defectives and defectives.	Page 82 Sorting Product Data (FB): Assortment

11 MEASUREMENT OF OPERATING TIME (TIME AND CHARACTER STRING)

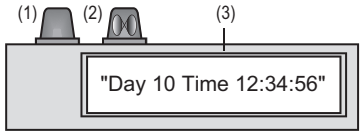
This chapter shows the examples for processing timer or time data using the program for the device which turns ON a lamp depending on the operating time and displays its operating time.
Create the following POU in the program example.

Data name	Data type	Description	Reference
OperatingTime	Program block	Counts operating time of a device in second unit.	Page 86 Operating Time Management: OperatingTime
FlickerTimer	Function block	Flashes the output signals alternately.	Page 87 Flicker Timer (FB): FlickerTimer
LampOnOff	Program block	Turns the lamp ON/OFF depending on the device status.	Page 88 Lamp ON/OFF: LampOnOff
SecondsToTimeArray	Program block	Calculates hour, minute, second from the time in second unit.	Page 89 Conversion from Sec. to Hour/Min/Sec: SecondsToTimeArray
TimeToString	Program block	Converts operating time (time type) to character string for display.	Page 90 Conversion from Time to String: TimeToString

Overview of function

The following processing are performed.

1. Operating time of the device is counted in second unit.
2. The lamp is turned ON/OFF by judging the status of device from the operation status and operating time.
3. Operating time is converted to hour, minute, and second unit.
4. Operating time is converted to the character string for screen display.

Device	Number	Name	Description	
	(1)	Operation lamp	Turns ON during the device is in operation. OFF: Stopped, ON: Operating	If the operating time is three weeks or more, the lamp flashes alternately.
	(2)	Warning lamp	If the operating time is one week or more, the lamp turns ON. OFF: Normal, ON: Warning	
	(3)	Screen display	Displays operating time (day, hour, minute, second).	

Global labels to be used

The following shows the global labels and structure definitions used in the program example.
Set the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/Constant	Description
G_bOperatingStatus	Bit	VAR_GLOBAL	—	Operation status (TRUE: Operating, FALSE: Stopped)
G_tmTime	Time	VAR_GLOBAL	—	Total operating time (Up to T#24d20h31m23s647ms)
G_bOperationLamp	Bit	VAR_GLOBAL	—	Operation lamp (TRUE: ON, FALSE: OFF)
G_bWarningLamp	Bit	VAR_GLOBAL	—	Warning lamp (TRUE: ON, FALSE: OFF)
G_dSeconds	Double Word [Signed]	VAR_GLOBAL	—	Operating time (0 to 86399 seconds)
G_wTimeArray	Word [Signed](0..2)	VAR_GLOBAL	—	Operating time ([0]: Hour, [1]: Minute, [2]: Second)
G_sDisplayedCharacters	String(32)	VAR_GLOBAL	—	Characters to be displayed on the operating time display screen
G_bOneScanOnly	Bit	VAR_GLOBAL	Assign (Device/Label): SM402	Turn ON for only 1 scan after RUN

■Structure

Do not use.

11.1 Operating Time Management: OperatingTime

This program counts operating time of a device in second unit.

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	OperatingTime	ST	Operating time management

Program example

```
(* Count the operating time for each one second. *)
bResult := OUT_T(G_bOperatingStatus, td1sTimer, 10); (* Timer setting value: 1000 ms *)
IF td1sTimer.S THEN (* After one second *)
    (* Operating time (Timer type) count up *)
    G_tmTime := G_tmTime + T#1000ms;
    IF G_tmTime < T#0ms THEN (* If an overflow occurred *)
        G_tmTime := T#0ms; (* Clear to 0 *)
    END_IF;
    (* Operating time (second unit) count up *)
    G_dSeconds := G_dSeconds +1;
    IF G_dSeconds >= 86400 THEN
        G_dSeconds := 0; (* Clear to 0 if the time is reached at 24 hours *)
    END_IF;
    (* Reset the timer (Current value: 0, Contact: OFF) *)
    RST(TRUE, td1sTimer.N);
    RST(TRUE, td1sTimer.S);
END_IF;
```

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Description
G_bOperatingStatus	Bit	VAR_GLOBAL	—	Operation status (TRUE: Operating, FALSE: Stopped)
G_tmTime	Time	VAR_GLOBAL	—	Total operating time (Up to T#24d20h31m23s647ms)
G_dSeconds	Double Word [Signed]	VAR_GLOBAL	—	Operating time (0 to 86399 seconds)

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Description
bResult	Bit	VAR	—	ENO for executing timer
td1sTimer	Timer	VAR	—	Timer for measuring one second

POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
OUT_T	Instruction	Low-speed timer instruction Starts time measurement when the operation result up to the OUT instruction is ON and the coil is turned ON.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

11.2 Flicker Timer (FB): FlickerTimer

This function block flashes Output signal 0 and 1 alternately when the input signal is ON.
Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	EN/ENO	Title
Function Block	FlickerTimer	ST	Yes	Flicker timer

Program example

```
(* Flash the Output signal 0 and 1 alternately. (Flip-flop circuit) *)
(* Calculate the setting value for the low-speed timer. *)
wInterval := TIME_TO_INT(i_tmInterval) / 100;
(* If the Timer 1 is OFF, turn the Timer 0 ON after the set time (ms). *)
bResult := OUT_T(NOT tdTimer1.S, tdTimer0, wInterval);
(* If the Timer 0 is turned ON from OFF, turn the Timer 1 ON after the set time (ms). *)
bResult := OUT_T(tdTimer0.S, tdTimer1, wInterval);
(* If the Timer 0 is ON, turn ON the Output signal 0. *)
o_bOutputSignal0 := tdTimer0.S;
(* If the Timer 0 is OFF, turn ON the Output signal 1. *)
o_bOutputSignal1 := NOT tdTimer0.S;
```

Point

The setting value for a timer needs to be equal to or more than the total value of scan time and timer limit setting. The timer limit is set as follows:

- MELSEC iQ-R: Set it in the parameter of an engineering tool. (Default: 100 ms)
- MELSEC iQ-F: The OUT_T instruction operates as a timer of 100 ms, the OUTH instruction as a timer of 10 ms, and the OUTHS instruction as a timer of 1 ms.

Variable

Define the labels by setting the following items in GX Works3.

■Local label

Label Name	Data Type	Class	Initial Value/Constant	Description
i_tmInterval	Time	VAR_INPUT	—	Switching interval
o_bOutputSignal0	Bit	VAR_OUTPUT	Initial Value: TRUE	Output signal 0
o_bOutputSignal1	Bit	VAR_OUTPUT	Initial Value: TRUE	Output signal 1
bResult	Bit	VAR	—	ENO for executing timer
wInterval	Word [Signed]	VAR	—	Low-speed timer setting value
tdTimer0	Timer	VAR	—	Timer 0
tdTimer1	Timer	VAR	—	Timer 1

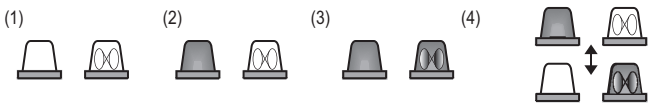
POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
OUT_T	Instruction	Low-speed timer instruction Starts time measurement when the operation result up to the OUT instruction is ON and the coil is turned ON.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
TIME_TO_INT	Standard function	Conversion of TIME type → INT type Converts a value from TIME type data to INT type data.	

11.3 Lamp ON/OFF: LampOnOff

This program turns the lamp ON/OFF depending on the device status.



Number	Operation lamp	Warning lamp	Device status		Remarks
(1)	OFF	OFF	Normal	Stopped	Before operation
(2)	ON	OFF		Operating	Operating time: 0 to 7 days
(3)	ON	ON	Warning		Operating time: One week or more
(4)	Flashing alternately		Error		Operating time: Three weeks or more

Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	LampOnOff	ST	Lamp ON/OFF

Program example

```
(* Turn the operation lamp ON/OFF according to the operation state. *)
G_bOperationLamp := G_bOperatingStatus;
(* Turn the warning lamp ON if the total operating time is one week (seven days) or more. *)
G_bWarningLamp := G_tmTime >= T#7d;
(* If the operating time is three weeks (21 days) or more *)
(* Turn the operation lamp and warning lamp alternately. *)
FlickerTimer_1(EN := G_tmTime >= T#21d, i_tmInterval := T#1000ms);
IF FlickerTimer_1.ENO THEN
    G_bOperationLamp := FlickerTimer_1.o_bOutputSignal0;
    G_bWarningLamp := FlickerTimer_1.o_bOutputSignal1;
END_IF;
```

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Description
G_bOperatingStatus	Bit	VAR_GLOBAL	—	Operation status (TRUE: Operating, FALSE: Stopped)
G_bOperationLamp	Bit	VAR_GLOBAL	—	Operation lamp (TRUE: ON, FALSE: OFF)
G_bWarningLamp	Bit	VAR_GLOBAL	—	Warning lamp (TRUE: ON, FALSE: OFF)
G_tmTime	Time	VAR_GLOBAL	—	Total operating time (up to T#24d20h31m23s647ms)

■Local label

Label Name	Data Type	Class	Initial Value/ Constant	Description
FlickerTimer_1	FlickerTimer	VAR	—	Flicker timer

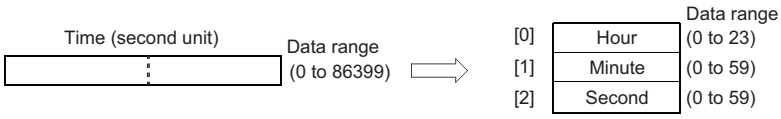
POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
FlickerTimer	Function block	Flashes the output signals alternately.	Page 87 Flicker Timer (FB): FlickerTimer

11.4 Conversion from Sec. to Hour/Min/Sec: SecondsToTimeArray

This program calculates hour, minute, second from the time in second unit. The values of hour, minute, and second are stored to the array type data (same format data as the argument of SEC2TIME (clock instruction)).



Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	SecondsToTimeArray	ST	Conversion from sec. to h/m/s

Program example

```
(* Calculate hour, minute, second from the time in second unit. *)
G_wTimeArray[0] := GET_INT_ADDR( G_dSeconds / 3600); (* Hour *)
G_wTimeArray[1] := GET_INT_ADDR((G_dSeconds MOD 3600) / 60); (* Minute *)
G_wTimeArray[2] := GET_INT_ADDR((G_dSeconds MOD 3600) MOD 60); (* Second *)
```

Point

MOD is a modulus operator.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Description
G_dSeconds	Double Word [Signed]	VAR_GLOBAL	—	Operating time (0 to 86399 seconds)
G_wTimeArray	Word [Signed](0..2)	VAR_GLOBAL	—	Operating time ([0]: Hour, [1]: Minute, [2]: Second)

■Local label

Do not use.

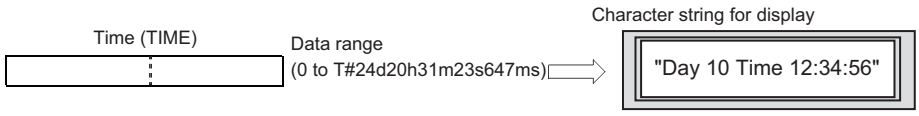
POU

For details on the POU used, refer to the following table.

Data name	Data type	Description	Reference
GET_INT_ADDR	Standard function	Eliminate the need for type conversion Outputs the input variable as INT type.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

11.5 Conversion from Time to String: TimeToString

This program converts operating time (time type) to character string for display.



Create a POU by setting the following items in GX Works3.

Data Type	Data Name	Program Language	Title
Program Block	TimeToString	ST	Conversion from Time to String

Program example

```
(* Convert the operating time to character string for display. *)
(* Initialize the internal variable. *)
IF G_bOneScanOnly THEN (* Execute only once after RUN *)
(* Initialize time unit (day, hour, minute, second in milliseconds). *)
    dUnitArray[0] := TIME_TO_DINT(T#1d); (* Day *)
    dUnitArray[1] := TIME_TO_DINT(T#1h); (* Hour *)
    dUnitArray[2] := TIME_TO_DINT(T#1m); (* Minute *)
    dUnitArray[3] := TIME_TO_DINT(T#1s); (* Second *)
(* Specify the digit of numeric value to be converted to a character string. *)
    wDigitArray[0] := 3; (* Convert to 2-digit numeric value. (Specify the total number of digits including a sign to 3.) *)
    wDigitArray[1] := 0; (* No decimal point *)
END_IF;
sTimeString := ''; (* Initialize character string. *)
(* Convert the time type data to milliseconds. *)
dTime := TIME_TO_DINT(G_tmTime);
(* Processing the day, hour, minute, second in this order. *)
FOR wIndex := 0 TO 3 BY 1 DO
(* Convert the time (in milliseconds) to day/hour/minute/second. *)
    wTime := DINT_TO_INT(dTime /dUnitArray[wIndex]);
    dTime := dTime MOD dUnitArray[wIndex];
(* Convert the value of day/hour/minute/second to character string with 2-digit numeric value. *)
    STR(TRUE, wDigitArray, wTime, sItemString); (* Convert to 3 characters including a sign. *)
    sItemString := DELETE(sItemString, 1, 1); (* Delete a sign in the first character. *)
    (* Set the character string to be added. *)
    CASE wIndex OF
        0 : sAdd := '$"Day ';
        1 : sAdd := ' Time ';
        2,3:sAdd := ':';
    END_CASE;
    (* Add a character string. *)
    sTimeString := CONCAT(sTimeString, sAdd, sItemString);
END_FOR;
(* Add a character string at the end. *)
sTimeString := CONCAT(sTimeString, '$"');
(* Set the created character string to the display character string. *)
G_sDisplayedCharacters := sTimeString;
```

Point

Use TIME_TO_STRING of the standard function to convert the values (in milliseconds) to character string. The program above converts the values to the character string after calculating the value for each day, time, minute, and second.

Variable

Define the labels by setting the following items in GX Works3.

■Global label

Label Name	Data Type	Class	Assign/Initial Value/ Constant	Description
G_tmTime	Time	VAR_GLOBAL	—	Total operating time (Up to T#24d20h31m23s647ms)
G_bOneScanOnly	Bit	VAR_GLOBAL	Assign (Device/Label): SM402	Turn ON for only 1 scan after RUN
G_sDisplayedCharacters	String(32)	VAR_GLOBAL	—	Characters to be displayed on the operating time display screen

■Local label

Label Name	Data Type	Class	Initial Value/ Constant	Description
dUnitArray	Double Word [Signed](0..3)	VAR	—	Unit (day, hour, minute, second in milliseconds)
wDigitArray	Word [Signed](0..1)	VAR	—	Digit specification ([0]: Number of all digits, [1]: Number of decimal place)
sTimeString	String(32)	VAR	—	Character string to be created (Day, hour, minute, second)
dTime	Double Word [Signed]	VAR	—	Time (in milliseconds)
wIndex	Word [Signed]	VAR	—	Element number
wTime	Word [Signed]	VAR	—	Time (Day/hour/minute/second)
sItemString	String(32)	VAR	—	Character string of 2-digit numeric value for day/time/minute/second
sAdd	String(32)	VAR	—	Character string to be added

POU

For details on the POUs used, refer to the following table.

Data name	Data type	Description	Reference
TIME_TO_DINT	Standard function	Conversion of TIME type → DINT type Converts a value from TIME type data to DINT type data.	MELSEC iQ-R Programming Manual (CPU Module Instructions, Standard Functions/ Function Blocks) MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
DINT_TO_INT	Standard function	Conversion of DINT type → INT type Converts a value from DINT type data to INT type data.	
STR	Instruction	Converts 16-bit binary data to character string by adding a decimal point to the specified place of the data.	
CONCAT	Standard function	Concatenation of string data Concatenates character strings and outputs the result.	

APPENDIX

Appendix 1 Specifications of Structured Text language

This section explains the types of basic components and their descriptions used in the Structured Text language.

Statement

The following shows the types of statement and their description method.

Type			Description method	Description
Assignment statement			<Variable> := <Expression>; Assign the result	An evaluation result of the right side is assigned to the variable of the left side.
Subprogram control statement	Call statement		<Identifier>(argument 1, argument 2, ...)	A function or function block is called.
	RETURN statement		RETURN;	A program is ended.

Control statement	Selection statement	IF	IF <Condition 1> THEN
			<Statement 1>;
			ELSIF <Condition 2> THEN
			<Statement 2>;
			Miltiple ELSIFs (in the line frame) are allowable.
			ELSE
			<Statement 3>;
			ELSIF, ELSE (in the dashed-line frame) are omittable.
			END_IF;
				An execution statement is selected depending on the condition of the boolean value.
				If the Condition 1 is TRUE, the Statement 1 is executed.
				If the Condition 1 is FALSE, the condition of ELSIF is judged. If the Condition 2 is TRUE, the Statement 2 is executed.
				If the conditions both IF and ELSIF are FALSE, the Statement 3 after ELSE is executed.

		CASE	CASE <Condition> OF <Value 1> : <Statement 1>; <Value 2>..<Value 3> : <Statement 2>; Multiple statements are allowable. ELSE <Statement 3>; ELSE (in the dashed-line frame) is omittable. END_CASE;	An execution statement is selected depending on the condition of the integer value. If the result of the conditional expression is equals to the Value 1, the Statement 1 is executed. When specifying the range of an integer value to be judged, use '..' to describe the statement. If the value of the result of the conditional expression is within the range from the Value 2 to the Value 3, the Statement 2 is executed. If the result of the conditional expression is not equal to all the integer values or their ranges, the Statement 3 after ELSE is executed.
	Iteration statement	FOR	FOR <Variable> := <Initial (expression)> TO <End (expression)> BY <Increment (expression)> BY (in the dashed-line (thin) frame) is omittable. DO <Statement>; END_FOR;	The execution statement is executed for multiple times depending on the end condition of the integer value. Set the initial value to the Variable of an integer type first. After that, the Statement is executed until the Variable reaches the value of the End. The Increment value is added to the variable each time when the Statement is executed.
		WHILE	WHILE <Condition> DO <Statement>; Repeat until the result becomes FALSE END_WHILE;	The execution statement is executed for multiple times depending on the end condition of boolean value. If the conditional expression is TRUE, the execution statement is executed. The processing is repeated until the result of the conditional expression becomes FALSE.
		REPEAT	REPEAT <Statement>; Repeat until the result becomes TRUE UNTIL <Condition> END_REPEAT;	The execution statement is executed for multiple times depending on the end condition of boolean value. The condition is judged after the execution of the execution statement. The processing is repeated until the result of the conditional expression becomes TRUE.
	Exit of iteration statement		EXIT;	Exits the iteration statement.
Empty statement			;	Nothing is processed.

Point

Hierarchization of functions and function blocks can be performed up to 32 times in total.
Hierarchization of IF statement, CASE statement, WHILE statement, and REPEAT statement can be performed up to 128 times in total.

Operator

The following shows the types of operators and their description method.

Type		Operator	Priority (High to low)	Example	
				Generic mathematical expression	ST
Inversion of sign		-	1	B = - A	eValueB := - eValueA;
Logical operation	NOT operation	NOT		$B = \overline{A}$	bFlagB := NOT bFlagA;
Exponentiation		**	2	$B = C^A$	eValueB := eValueC ** eValueA;
Basic arithmetic expression	Multiplication	*	3	$A \times B = C$	eValueC := eValueA * eValueB;
	Division	/		$A \div B = C \ldots D$	eValueC := eValueA / eValueB;
	Modulus operation	MOD			eValueC := eValueA MOD eValueB;
	Addition	+	4	A + B = C	eValueC := eValueA + eValueB;
	Subtraction	-		A - B = C	eValueC := eValueA - eValueB;
Comparison operation	Greater that, less than	>, <	5	A > B	bFlag := eValueA > eValueB;
	Greater than or equal to, less than or equal to	>=, <=		$A \leq B$	bFlag := eValueA <= eValueB;
	Equality	=	6	A = B	bFlag := eValueA = eValueB;
	Inequality	<>		$A \neq B$	bFlag := eValueA <> eValueB;
Logical operation	AND operation	AND, &	7	$A \wedge B$	bFlag := eValueA AND eValueB;
	XOR operation	XOR	8	$A \veebar B$	bFlag := eValueA XOR eValueB;
	OR operation	OR	9	$A \vee B$	bFlag := eValueA OR eValueB;

Point

Up to 1024 operators can be described in one expression.

Priority

When multiple operational expressions are described in one statement, the operations are processed in order from a high priority operator.
When some operators of which priority is the same are used in one statement, the operators are operated in order from the left.
The operational expression in the brackets '()' is operated first.

Comment

The following shows the types of comments and their description method.

Type	Symbol	Description	Example
Multiple line comment	(* *)	The range from start symbol to end symbol is regarded as a comment.	(* Comment *)
	/* */		/* Comment */
Single line comment	//	The range from start symbol to end of the line is regarded as a comment.	// Comment

Point

A comment symbol for the Structured Text language defined in IEC 61131-3 is (**) only, however, in GX Works3, the same symbols as C language (/**/, //) can be described.
In GX Works2, the comment used (**) can only be used.

Device

Devices can be specified in the same manner as a ladder program. Add # when specifying local devices.
Device specification (digit specification, bit specification, and indirect specification) and index modification can be used.

Restriction

A pointer cannot be used in ST program.

Contact/coil/current value of timer and counter

The devices such as timer and counter can be used by specifying their purpose to be used as a contact/coil/current value in the Structured Text language.
When do not specify them, the devices are distinguished to be used as a contact/coil/current value automatically depending on the instruction to be used.
Contact and coil are treated as a bit type. Current value is treated as the following data type.

Device	Notation				Data type of current value
	No specification	Contact	Coil	Current value	
Timer	T	TS	TC	TN	Word [Unsigned]/Bit String [16-bit]
Retentive timer	ST	STS	STC	STN	
Counter	C	CS	CC	CN	
Long timer	LT	LTS	LTC	LTN	Double Word [Unsigned]/Bit String [32-bit]
Long retentive timer	LST	LSTS	LSTC	LSTN	
Long counter	LC	LCS	LCC	LCN	

Type specification of word device

Word device can be used by specifying its data type.

Data type	Device type specifier	Example	Description
Word [Unsigned]/Bit String [16-bit]	:U	D0:U	The value of D0 is treated as the value of 16-bit WORD type.
Double Word [Unsigned]/Bit String [32-bit]	:UD	D0:UD	The values of D10 and D1 are treated as the value of 32-bit DWORD type.
Word [Signed]	(None)	D0	The value of D0 is treated as the value of 16-bit INT type.
Double Word [Signed]	:D	D0:D	The values of D0 and D1 are treated as the value of 32-bit DINT type.
FLOAT [Single Precision]	:E	D0:E	The values of D0 and D1 are treated as the value of 32-bit REAL type.
FLOAT [Double Precision]	:ED	D0:ED	The values from D0 to D3 are treated as the value of 64-bit LREAL type.

A device type specifier cannot be added to digit-specified devices or indirectly specified devices.

■Type conversion of word device

When the word device with no type specification is used for an argument of function or function block, the word device is treated according to the data type of the argument definition. (Same as standard function, standard function block, and instruction.)
The result may differ depending on the specification method for the argument since the type conversion is performed automatically at the specification of the input argument.

Ex.

32-bit binary data transfer instruction (DMOV) (Instruction of which input argument and output argument are both ANY32 type)

- D0 = 16#ABCD
- D1 = 16#1234

• Word [Signed] type label, G_wLabel to which D0 is assigned

ST	Description	Value to be transferred
bResult := DMOV(TRUE, D0, D10);	The 32-bit data stored to D0 and D1 are transferred to D10 and D11.	16#1234ABCD
bResult := DMOV(TRUE, D0:UD, D10:UD);		
bResult := DMOV(TRUE, D0:U, D10);	The integer value of Word [Unsigned]/Bit String [16-bit] type stored to D0 is converted to Double Word [Signed] type automatically, and the zero-extended value is transferred to D10 and D11.	16#0000ABCD
bResult := DMOV(TRUE, G_wLebel, D10);	The integer value of Word [Signed] type stored to D0 is converted to Double Word [Signed] type automatically, and the sing-extended value is transferred to D10 and D11.	16#FFFFABCD
bResult := DMOV(TRUE, D0:U, D10:U);	A conversion error occurs in D10:U since the output variable cannot be converted automatically.	—

When using a word device with no type specification in an operational expression, the data type is converted from Word [Signed] automatically.

☞ Page 30 Data type that can be converted automatically

Label

Specify labels in the same manner as a ladder program.

Bit specification (example: Lbl.3) and digit specification (example: K4Lbl) of labels can be used.

Restriction

Pointer type labels cannot be used in an ST program.

Constant

The following shows the description method of constant.

Type		Notation	Example	Example with '_' added *1
Boolean value		Describe a boolean value with TRUE or FALSE.	TRUE, FALSE	
		Specify the value with 1 or 0. Each notation for integer can be applied.	2#0, 8#1, 0, H1	
Integer	Binary	Add '2#' in front of a binary number.	2#0010, 2#01101010	2#111_1111_1111_1111
	Octal	Add '8#' in front of an octal number.	8#2, 8#152, 8#377	8#7_7777
	Decimal	Enter a decimal number directly.	2, 106, -1	32_767
		Add 'K' in front of a decimal number.	K2, K106, K-1	—*2
	Hexadecimal	Add '16#' in front of a hexadecimal number.	16#2, 16#6A, 16#FF	16#7F_FF
		Add 'H' in front of a hexadecimal number.	H2, H6A, HFF	—*2
Real number	Decimal notation	Enter a real number directly.	1200.0, 0.012, -0.1	3.14_159
		Add 'E' in front of a real number.	E1200, E0.012, E-0.1	—*2
	Exponential notation	Add 'E' between the mantissa part and the exponent. 'mEn' indicates that multiplying mantissa part 'm' by the nth power of 10.	1.2E3, 1.2E-2, -1.0E-1	2.99_792_458E8
		Add 'E' in front of the mantissa part, and add '+' or '-' between the mantissa part and the exponent. 'Em+n' indicates that multiplying mantissa part 'm' by the nth power of 10.*3	E1.2+3, E1.2-2, E-1.0-1	—*2
Character string	ASCII Shift-JIS	Enclose a character string in single quotes (').	'ABC'	
	Unicode	Enclose a character string in double quotes (").	"ABC"	
Time		Add 'T#' or 'TIME#' in front of a value.	T#1h, T#1d2h3m4s5ms, TIME#1h	

*1 In the notation of integer and real numbers, the numbers can be delimited using an underscore '_' to make programs easy to see. And, the delimiters of values (underscore '_') are ignored in the program processing.

*2 For the notation with K, H, or E added in front, underscores (_) cannot be used.

*3 By adding '+' or '-' right after the value for the real number with 'E' added, it is regarded as an index.
When describing it as an arithmetic operation, insert a space or tabulator between the value and operator.
(Example: 'E1.2+3' indicates 1200.0, and 'E1.2 +3' indicates 4.2.)

Specifying a data type

The data types of the following values can explicitly be specified. When not specified, the data types are determined automatically.

Data type names are not case-sensitive and cannot be used together with the constant notation using K, H, or E.

Constant	Available data type		Example	Example with '_' added
Boolean value	Bit	BOOL	BOOL#TRUE, BOOL#FALSE, BOOL#0, BOOL#1	
Integer	Word [Unsigned]/Bit String [16-bit]	UINT	UINT#2#01101010, UINT#8#152, UINT#106, UINT#16#6A	UINT#2#0110_1010
		WORD*1	WORD#2#01101010, WORD#8#152, WORD#106, WORD#16#6A	WORD#2#0110_1010
	Double Word [Unsigned]/ Bit String [32-bit]	UDINT	UDINT#2#11111111111111111, UDINT#8#77777, UDINT#32767, UDINT#16#7FFF	UDINT#16#7F_FF
		DWORD*1	DWORD#2#11111111111111111, DWORD#8#77777, DWORD#32767, DWORD#16#7FFF	DWORD#16#7F_FF
	Word [Signed]	INT	INT#2#01101010, INT#8#152, INT#106, INT#16#6A	INT#2#0110_1010
	Double Word [Signed]	DINT	DINT#2#11111111111111111, DINT#8#77777, DINT#32767, DINT#16#7FFF	DINT#16#7F_FF
Real number	FLOAT [Single Precision]	REAL	REAL#2.34, REAL#1.0E6	REAL#3.14_159
	FLOAT [Double Precision]	LREAL	LREAL#-2.34, LREAL#1.001E16	LREAL#1.00_1E16

*1 'WORD#' and 'DWORD#' cannot be used for the operands of basic arithmetic operations (target values of operations), the call statements of functions and function blocks, and the ANY_NUM type arguments in function call expressions, . Use 'UINT#' or 'UDINT#' instead.

Using '$' in a character string type constant

When specifying linefeed using a character string type constant, add '$'.

Type	Notation
Dollar sign ($)	$$, $24
Single quote (')	$', $27
Double quote (")	$", $22
Line feed	$L, $l, $0A
Newline	$N, $n, $0D $0A
Form feed (page)	$P, $p, $0C
Carriage Return	$R, $r, $0D
Tabulator	$T, $t, $09
Character corresponding to ASCII code	$[ASCII code (2-digits of hexadecimal value)]

Description method of the time length

Describe the time type of constants as follows:

- T#23d23h59m59s999ms or TIME#23d23h59m59s999ms

Time unit	d (day)	h (hour)	m (minute)	s (second)	ms (millisecond)
Range	0 to 24	0 to 23	0 to 59	0 to 59	0 to 999

Time units should be consecutive in order of d, h, m, s, and ms.

By adding a sign (-) after '#', values can be described as signed ones.

(Example: T#-24d20h31m23s648ms)

■Omitting a time unit

Time units that are placed before and after ones to be described can be omitted.

(Example: T#1m2s)

Time units that are placed between ones to be described cannot be omitted.

(Example: 'T#1m2ms' is not allowed. Describe 'T#1m0s2ms'.)

The last time unit can be described in a notation using a decimal point (decimal representation of unsigned real numbers).

(Example: 'T#1.234s' is described as 'T#1s234ms'.)

The ms (millisecond) unit cannot be described in a decimal point notation. Values after a decimal point are rounded down.

(Example: 'T#1.234ms' is described as 'T#1ms'.)

■Setting range

When using a sign (-) or omitting time units that are placed before ones to be described, the range of values that can be described is as follows:

Time unit	d (day)	h (hour)	m (minute)	s (second)	ms (millisecond)
No omission	-24 to 24	0 to 23	0 to 59	0 to 59	0 to 999
'd' omitted	—	-596 to 596	0 to 59	0 to 59	0 to 999
'd' and 'h' omitted	—	—	-35791 to 35791	0 to 59	0 to 999
'd', 'h', and 'm' omitted	—	—	—	-2147483 to 2147483	0 to 999
'd', 'h', 'm', and 's' omitted	—	—	—	—	-2147483648 to 2147483647

The time length can be set to the time type variable (☞ Page 33 Time type variable) in the following range.

- T#-24d20h31m23s648ms to T#24d20h31m23s647ms
- T#-596h31m23s648ms to T#596h31m23s647ms
- T#-35791m23s648ms to T#35791m23s647ms
- T#-2147483s648ms to T#2147483s647ms
- T#-2147483648ms to T#2147483647ms

Function and function block

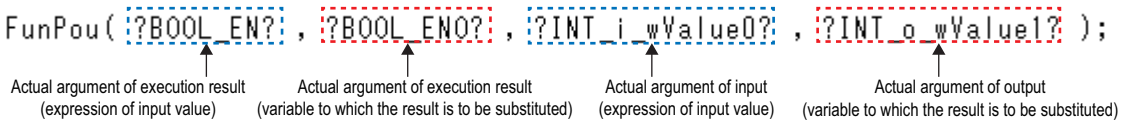
Function and function block are the POUs in which a subroutine to be called from a program is defined.
The defined function can be used in a program block, function block, and other functions.
The defined function block can be used in the program block and other function blocks by creating an instance.

Argument

The following shows the description method of arguments in a call statement.

■Argument specification with actual argument

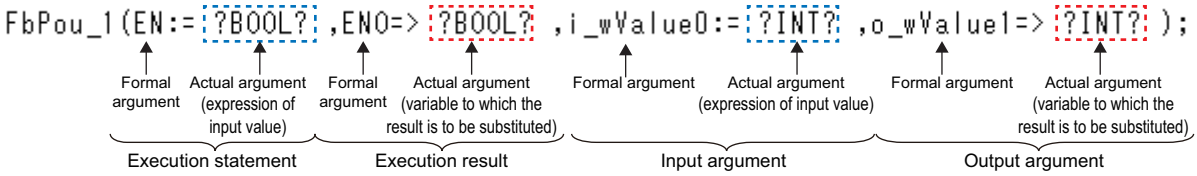
List the arguments by delimiting them with commas (,) in the brackets.
The template of a function is displayed in the following format. (☞ Page 52 Entering arguments)



The order of the arguments will be the order defined in the local label setting of function definition or function block definition.
When function or function block is created with the setting to use EN/ENO, specify EN for the first argument and ENO for the second one.

■Argument specification for assigning actual argument to formal argument

List the formal arguments and actual arguments by delimiting with commas (,) in the brackets.
The template of a function block is displayed in the following format. (☞ Page 53 Entering arguments)



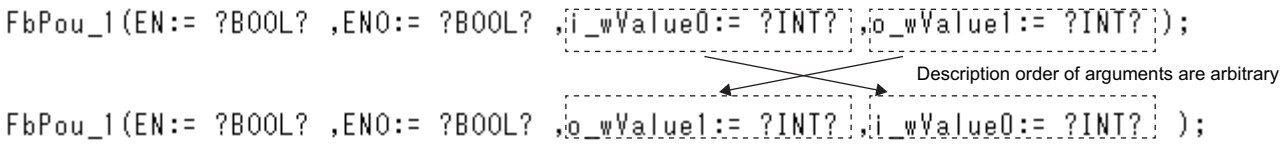
The names of formal arguments will be the variable names specified in the local label setting of function definition or function block definition.
Specify 'EN' and 'ENO' as formal arguments for EN and ENO.

Point

When assigning the actual argument to the formal argument, use the following format.

- Input argument, input/output argument and EN: '<Formal argument name>:=<Expression>'
- Output argument and ENO: '<Formal argument name>=><Variable>'

When assigning the actual argument to the formal argument, the order of the arguments can be changed arbitrary.

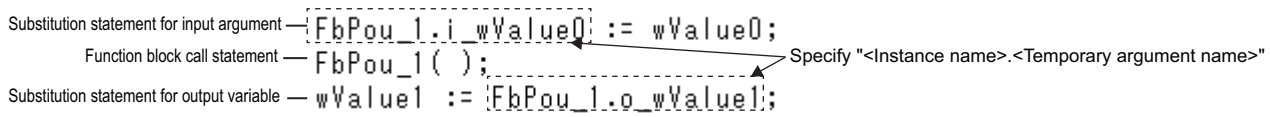


Point

When specifying the argument by assigning the actual argument to the formal argument, the description of the argument can be omitted.

■Argument specification before and after the call statement

For a function block, arguments can be specified before and after the call statement. Describe the assignment statement for an input variable before the call statement, and describe an assignment statement for the output variable after the call statement.



A

Return value

A function which has a return value returns the value to the call source after the completion of the execution.

■Data type of return value

Set the data type of a return value on the "New Data" screen or the "Properties" screen.

A function with no data type set is treated as a function with no return value.

A function with no return value must be described as a call statement. (Function call statement)

■Setting value to return value

Set the value to be returned as a return value to the variable of which function name is its identifier in the function.

Program example

The sum from i_wValue0 to i_wValue2 (input argument) are returned as a return value of FunAdd (function).

ST (Program of function, FunAdd)

```
FunAdd := wValue0 + wValue1 + wValue2;
```

■Using return value in call source program

A function with return value can be handled as an expression. (Function call expression)

In the call source program of a function, operational expressions and conditional statements can be described using a function call expression as a variable of return value data type.

Program example

The average from wValue0 to wValue2 are assigned to Average3.

FunAdd (function) returns the sum of wValue0 to wValue2 (input argument) as a return value.

ST (Call source program)

```
Average3 := FunAdd( wValue0 , wValue1, wValue2) / 3;
```

EN and ENO

When creating a function and function block, whether or not to use an EN and ENO can be selected.

By using an EN (enable input) and ENO (enable output), the execution processing can be controlled.

- EN: Set the execution condition.
- ENO: Execution result is output.

EN/ENO is boolean type.

A function or function block with an EN is executed only when the execution condition of the EN is TRUE.

The following shows the values of output variable and return value depending on the state of EN and ENO.

EN	ENO	Output variable, returned value	Remarks
TRUE	TRUE	Operation output value	Normal completion
	FALSE	Undefined value	Undefined value is returned when FALSE is assigned to the ENO during processing. The output value depends on the actual system.
FALSE	FALSE	• Function: Value at call • Function block: Previous result	The processing is ended without execution. Assignment of value to the input variable and output variable are not executed.

■Setting value to ENO

To set the value of ENO, assign a boolean value to the variable, 'ENO' in a function or function block.

Program example

If an operation error occurs in the BCD instruction, the processing of function/function block is terminated and resulted in error end.

ST (Program of function/function block)

```
ENO := BCD(EN, wValue0, D0);
IF ENO = FALSE THEN
    RETURN;
END_IF;
```

Appendix 2 Instructions That Cannot be Used in ST Programs

The following instructions, which are used in ladder programs, are described by using an operator or a control statement in ST programs.

Instructions that can be described in assignment statement

Type	Instruction symbol	Corresponding type conversion function
Converting single-precision real number to 16-bit signed binary data	FLT2INT(P)	REAL_TO_INT(_E)
Converting single-precision real number to 16-bit unsigned binary data	FLT2UINT(P)	—(REAL_TO_DINT(_E), DINT_TO_WORD(_E))
Converting single-precision real number to 32-bit signed binary data	FLT2DINT(P)	REAL_TO_DINT(_E)
Converting single-precision real number to 32-bit unsigned binary data	FLT2UDINT(P)	—(REAL_TO_DINT(_E), DINT_TO_DWORD(_E))
Converting double-precision real number to 16-bit signed binary data	DBL2INT(P)	LREAL_TO_INT(_E)
Converting double-precision real number to 16-bit unsigned binary data	DBL2UINT(P)	—(LREAL_TO_DINT(_E), DINT_TO_WORD(_E))
Converting double-precision real number to 32-bit signed binary data	DBL2DINT(P)	LREAL_TO_DINT(_E)
Converting double-precision real number to 32-bit unsigned binary data	DBL2UDINT(P)	—(LREAL_TO_DINT(_E), DINT_TO_DWORD(_E))
Converting 16-bit signed binary data to 16-bit unsigned binary data	INT2UINT(P)	INT_TO_WORD(_E)
Converting 16-bit signed binary data to 32-bit signed binary data	INT2DINT(P)	INT_TO_DINT(_E)*1
Converting 16-bit signed binary data to 32-bit unsigned binary data	INT2UDINT(P)	INT_TO_DWORD(_E)
Converting 16-bit unsigned binary data to 16-bit signed binary data	UINT2INT(P)	WORD_TO_INT(_E)
Converting 16-bit unsigned binary data to 32-bit signed binary data	UINT2DINT(P)	WORD_TO_DINT(_E)*1
Converting 16-bit unsigned binary data to 32-bit unsigned binary data	UINT2UDINT(P)	WORD_TO_DWORD(_E)*1
Converting 32-bit signed binary data to 16-bit signed binary data	DINT2INT(P)	DINT_TO_INT(_E)
Converting 32-bit signed binary data to 16-bit unsigned binary data	DINT2UINT(P)	DINT_TO_WORD(_E)
Converting 32-bit signed binary data to 32-bit unsigned binary data	DINT2UDINT(P)	DINT_TO_DWORD(_E)
Converting 32-bit unsigned binary data to 16-bit signed binary data	UDINT2INT(P)	DWORD_TO_INT(_E)
Converting 32-bit unsigned binary data to 16-bit unsigned binary data	UDINT2UINT(P)	DWORD_TO_WORD(_E)
Converting 32-bit unsigned binary data to 32-bit signed binary data	UDINT2DINT(P)	DWORD_TO_DINT(_E)
Transferring string data	$MOV(P)	—*1
Transferring Unicode string data	$MOV(P)_WS	—*1
Converting 16-bit signed binary data to single-precision real number	INT2FLT(P)	INT_TO_REAL(_E)*1
Converting 16-bit unsigned binary data to single-precision real number	UINT2FLT(P)	—(WORD_TO_INT(_E), INT_TO_REAL(_E))*1
Converting 32-bit signed binary data to single-precision real number	DINT2FLT(P)	DINT_TO_REAL(_E)
Converting 32-bit unsigned binary data to single-precision real number	UDINT2FLT(P)	—(DWORD_TO_DINT(_E), DINT_TO_REAL(_E))
Converting double-precision real number to single-precision real number	DBL2FLT(P)	LREAL_TO_REAL(_E)
Converting 16-bit signed binary data to double-precision real number	INT2DBL(P)	INT_TO_LREAL(_E)*1
Converting 16-bit unsigned binary data to double-precision real number	UINT2DBL(P)	—(WORD_TO_DINT(_E), DINT_TO_LREAL(_E))*1
Converting 32-bit signed binary data to double-precision real number	DINT2DBL(P)	DINT_TO_LREAL(_E)*1
Converting 32-bit unsigned binary data to double-precision real number	UDINT2DBL(P)	—(DWORD_TO_DINT(_E), DINT_TO_LREAL(_E))*1
Converting single-precision real number to double-precision real number	FLT2DBL(P)	REAL_TO_LREAL(_E)*1

*1 Only assignment statement can be used for transferring character string data, or for a data type that can be converted automatically (☞ Page 30 Type conversion which is performed automatically).

Instructions that can be described with operator

Instructions that can be described with arithmetic operator

Type	Instruction symbol
Adding 16-bit binary data	+(P)(_U) [Using two operands]
Subtracting 16-bit binary data	-(P)(_U) [Using two operands]
Adding 32-bit binary data	D+(P)(_U) [Using two operands]
Subtracting 32-bit binary data	D-(P)(_U) [Using two operands]
Multiplying 16-bit binary data	*(P)(_U)

Type	Instruction symbol
Dividing 16-bit binary data	/(P)(_U)
Multiplying 32-bit binary data	D*(P)(_U)
Dividing 32-bit binary data	D/(P)(_U)
Adding BCD 4-digit data	B+(P) [Using two operands]
Subtracting BCD 4-digit data	B-(P) [Using two operands]
Adding BCD 8-digit data	DB+(P) [Using two operands]
Subtracting BCD 8-digit data	DB-(P) [Using two operands]
Multiplying BCD 4-digit data	B*(P)
Dividing BCD 4-digit data	B/(P)
Multiplying BCD 8-digit data	DB*(P)
Dividing BCD 8-digit data	DB/(P)
Adding 16-bit binary block data	BK+(P)(_U)
Subtracting 16-bit binary block data	BK-(P)(_U)
Adding 32-bit binary block data	DBK+(P)(_U)
Subtracting 32-bit binary block data	DBK-(P)(_U)
Concatenating string data	$+(P) [Using two operands]
Adding single-precision real numbers	E+(P) [Using two operands]
Subtracting single-precision real numbers	E-(P) [Using two operands]
Adding double-precision real numbers	ED+(P) [Using two operands]
Subtracting double-precision real numbers	ED-(P) [Using two operands]
Multiplying single-precision real numbers	E*(P)
Dividing single-precision real numbers	E/(P)
Multiplying double-precision real numbers	ED*(P)
Dividing double-precision real numbers	ED/(P)
Adding clock data	DATE+(P)
Subtracting clock data	DATE-(P)
Adding expansion clock data	S(P).DATE+
Subtracting expansion clock data	S(P).DATE-

Instructions that can be described with logical operator and comparison operator

Type	Instruction symbol
Operation start, series connection, parallel connection	LD, LDI, AND, ANI, OR, ORI
Ladder block series/parallel connection	ANB, ORB
Comparing 16-bit binary data	LD□(_U), AND□(_U), OR□(_U)
Comparing 32-bit binary data	LDD□(_U), ANDD□(_U), ORD□(_U)
Comparing 16-bit binary block data	BKCMP□(P)(_U)
Comparing 32-bit binary block data	DBKCMP□(P)(_U)
Performing an AND operation on 16-bit data	WAND(P) [Using two operands]
Performing an AND operation on 32-bit data	DAND(P) [Using two operands]
Performing an OR operation on 16-bit data	WOR(P) [Using two operands]
Performing an OR operation on 32-bit data	DOR(P) [Using two operands]
Performing an XOR operation on 16-bit data	WXOR(P) [Using two operands]
Performing an XOR operation on 32-bit data	DXOR(P) [Using two operands]
Performing an XNOR operation on 16-bit data	WXNR(P) [Using two operands]
Performing an XNOR operation on 32-bit data	DXNR(P) [Using two operands]
Comparing string data	LD$□, AND$□, OR$□
Comparing single-precision real numbers	LDE□, ANDE□, ORE□
Comparing double-precision real numbers	LDED□, ANDED□, ORED□
Comparing date data	LDDT□, ANDDT□, ORDT□
Comparing time data	LDTM□, ANDTM□, ORTM□

Instructions that can be described with control statement or function

Type	Instruction symbol
Performing the FOR to NEXT instruction loop	FOR, NEXT
Pointer branch	CJ, SCJ, JMP
Jumping to END	GOEND
Returning from the interrupt program	IRET
Forcibly terminating the FOR to NEXT instruction loop	BREAK(P)
Calling a subroutine program	CALL(P)
Returning from the subroutine program called	RET
Calling a subroutine program and turning the output OFF	FCALL(P)
Calling a subroutine program in the specified program file	ECALL(P)
Calling a subroutine program in the specified program file and turning the output OFF	EFCALL(P)
Calling a subroutine program	XCALL

Unnecessary instructions for ST program

Type	Instruction symbol
Ending the sequence program	END
No operation (NOP)	NOP, NOPLF

Appendix 3 Considerations for Using the MELSEC iQ-F Series

Description in this manual is based on the use of MELSEC iQ-R series.
This section shows the considerations for creating an ST program for the MELSEC iQ-F series by referring to this manual.

Differences between the MELSEC iQ-F series and MELSEC iQ-R series

The MELSEC iQ-F series has specifications different from the MELSEC iQ-R series for the items in the table below.
Note that some devices and data types that are described in this manual are not available for creating an ST program for the MELSEC iQ-F series.

Item		Difference	Reference
Device		Available device types are different. Check the specifications of a CPU module to be used.	MELSEC iQ-F FX5 User's Manual (Application)
Data type	FLOAT [Double Precision] (LREAL)	The MELSEC iQ-F series does not support it.	MELSEC iQ-F FX5 Programming Manual (Program Design)
Constant	String [Unicode] type	The MELSEC iQ-F series does not support it.	

Restrictions on Structured Text language are the same between the MELSEC iQ-R series and MELSEC iQ-F series.
However, besides above, there may be differences in some other specifications such as for instructions, etc. between the series.
Check the specifications of modules to be used when creating an ST program.

INDEX

I

MEMO

REVISIONS

*The manual number is given on the bottom left of the back cover.

Revision date	*Manual number	Description
February 2015	SH(NA)-081483ENG-A	First edition
April 2015	SH(NA)-081483ENG-B	■Added or modified parts TERMS, Section 6.3, Appendix 1
May 2016	SH(NA)-081483ENG-C	■Added or modified parts Section 1.2, Section 10.1, Section 10.3
December 2017	SH(NA)-081483ENG-D	■Added or modified parts Section 4.4, Appendix 1
November 2020	SH(NA)-081483ENG-E	■Added or modified parts SAFETY PRECAUTIONS, CONDITIONS OF USE FOR THE PRODUCT, INTRODUCTION, Section 2.2, Section 3.2, Section 3.3, Section 8.4, Section 8.6, Section 9.2, Section 9.3, Section 10.2, Section 11.1, Section 11.2, Section 11.4, Section 11.5, Appendix 1, Appendix 3
May 2022	SH(NA)-081483ENG-F	■Added or modified part Section 11.5

Japanese manual number: SH-081445-G

TRADEMARKS

Unicode is either a registered trademark or a trademark of Unicode, Inc. in the United States and other countries.
The company names, system names and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.
In some cases, trademark symbols such as '™' or '®' are not specified in this manual.

SH(NA)-081483ENG-F(2205)KWIX
MODEL: R-ST-GUIDE-E
MODEL CODE: 13JX28

MITSUBISHI ELECTRIC CORPORATION

HEAD OFFICE : TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS : 1-14 , YADA-MINAMI 5-CHOME , HIGASHI-KU, NAGOYA , JAPAN