

[Notes]

R20TS0390EJ0100

Rev.1.00

C/C++ Compiler Package for RX Family

Jan. 16, 2019

Outline

When using the C/C++ compiler package for RX family CC-RX, note the following points.

1. Point for caution when the -misra2012 option is specified (CCRX#050)
2. Point for caution regarding constant expressions that include type conversion from the floating-point type to the 64-bit integer type (CCRX#051)

Note: The number following the note is an identifying number for the precautionary note.

1. Point for caution when the -misra2012 option is specified (CCRX#050)

1.1 Applicable Products

CC-RX V2.05.00 to V2.08.00, V3.00.00 [Professional edition] (Rule 16.1 and 16.4)

CC-RX V2.06.00 to V2.08.00, V3.00.00 [Professional edition] (Rule 15.6, 15.7, and 16.2)

1.2 Details

When checking source code against MISRA-C:2012 rules by specifying -misra2012 option, the compiler may output a message for a code which does not violate the rules and may not output a message for a code which violates the rules.

MISRA-C is a set of software development guidelines whose purpose is to maintain the safety, portability and reliability of embedded systems programmed in the C language.

1.3 Conditions

An error occurs when following rule numbers are specified to be checked.

- Rule 15.6
No message is output for a code that violates the rule if the -lang=c99 option is specified.
- Rule 15.7
No message is output for a code that violates the rule if the -lang=c99 option is specified.
- Rule 16.1
No message is output for a code that violates the rule if all of the following conditions are met:
 - (1) “{” is written immediately after a switch statement (controlling expression).
 - (2) Both a case clause and a default clause are written in the switch statement (1).
 - (3) Each case clause and default clause in (2) ends with a break statement or a compound statement ^(Note1) (block) which includes a break statement at the end.
 - (4) At least one of the case clauses or default clauses in (3) meets all the conditions below.
 - (4-1) A compound statement (block) which is neither a selection statement (if or switch) nor a repeat statement (while, do-while, or for) is written at the end.
 - (4-2) A statement is written before the compound statement (block) in (4-1).

Note 1: A compound statement is a statement enclosed with “{ }”. An if statement enclosed with “{ }” is also a compound statement.

- Rule 16.2
No message is output for a code that violates the rule if all of the following conditions are met.
 - (1) The -lang=c99 option is specified.
 - (2) A case or default label is written immediately after switch (controlling expression) without “{”.
- Rule 16.4
A message may be output for a code that does not violate the rule if either of the following conditions is met:
 - (1) -lang=c is specified and a compound statement (block) is written in the function definition.
 - (2) -lang=c99 is specified, and a compound statement (block), selection statement (if or switch), or repeat statement (while, do-while, or for) is written in the function definition.
(This includes a case where a selection statement or repeat statement is written without “{ }”.)

1.4 Example

The example of an error is shown below. Characters in red are the parts corresponding to the conditions.

[C source code] (rule 16.1)

1:	int x;	
2:	void func(void) {	
3:	switch(x) {	// Condition (1)
4:	case 1:	// Condition (2)
5:	++x;	// Condition (4-2)
6:	{	// Condition (4-1)
7:	--x;	
8:	break;	// Condition (3)
9:	}	// Condition (4-1)
10:	default:	// Condition (2)
11:	break;	// Condition (3)
12:	}	// Condition (1)
13:	}	

Although the C source code above violates rule 16.1 of MISRA C: 2012, no message is output.

Lines 3 and 12: Condition (1) is met because “{” is written immediately after switch (controlling expression).

Lines 4 and 10: Condition (2) is met because both a case clause and a default clause are written.

Lines 8 and 11: Condition (3) is met because the case clause and default clause end with a break statement.

Lines 6 and 9: Condition (4-1) is met because the case clause ends with a compound statement (block).

Line 5: Condition (4-2) is met because a statement is written before a compound statement (block).

[C source code] (rule 16.2)

1:	int x;	
2:	void func(void) {	
3:	switch(x)	// Condition (2)
4:	case 1:	// Condition (2)
5:	break;	
6:	}	

The C source code above violates rule 16.2 of MISRA C:2012. Although a message is output when -lang=c is specified, no message is output when -lang=c99 is specified.

Lines 3 and 4: Condition (2) is met because a case label is written without "{" immediately after switch (controlling expression).

1.5 Workaround

There is no workaround for this problem.

1.6 Schedule for Fixing the Problem

This problem is fixed in CC-RX V3.01.00. (Scheduled to be released on January 21.)

2. Point for caution regarding constant expressions that include type conversion from the floating-point type to the 64-bit integer type (CCRX#051)

2.1 Applicable Products

CC-RX V1.00.00 to V1.02.01, V2.00.00 to V2.08.00, and V3.00.00

2.2 Details

The result of a constant expression that includes type conversion from the floating-point type to the 64-bit integer type may be incorrect.

2.3 Conditions

An error occurs when all of the following conditions (1) through (5) are met.

- (1) The -round=zero option is specified.
- (2) A constant expression is written.
- (3) A subexpression in (2) ^(Note1) includes a constant expression of the float, double, or long double floating-point type.
Note 1: Cases where (2) itself is a constant expression of the float, double, or long double type are included.
- (4) There is a type conversion from the subexpression in (3) to the signed long long or unsigned long long 64-bit integer type, including implicit type conversion.
- (5) The value of the subexpression in (3) falls within the following range:
 - (5-1) When converting a float-type constant expression or a double- or long double-type constant expression with the -dbl_size=4 option specified to the following type:
 - (5-1-a) When converting to the signed long long type:
 - 2.147483e+09 to 3.602880e+16
 - or
 - -2.147483e+09 to -3.602880e+16

(5-1-b) When converting to the unsigned long long type:

- 4.294967e+09 to 3.602880e+16

(5-2) When converting a double-type constant expression with the `-dbl_size=8` option specified or a long double-type constant expression to the following type:

(5-2-a) When converting to the signed long long type:

- 2.147483e+09 or more
or
-2.147483e+09 or less

(5-2-b) When converting to the unsigned long long type:

- 4.294967e+09 or more

2.4 Example

The example of an error is shown below.

```
long long ll = (long long)(123456789123.0+123+456); // Condition (2)(3)(4)(5)
```

-Condition (2) is met because constant expression “(long long)(123456789123.0+123+456)” is written.

-Condition (3) is met because a subexpression of the constant expression includes the double type (123456789123.0).

-Condition (4) is met because conversion to the long long type is included.

-Condition (5-2-a) is met because the calculation result (123456789702.0) of the constant expression falls within the range.

[Assembler source code (wrong compilation result)]

```
1:  _ll:
2:      .lword    0BE992000H,0FFFFFFFH
```

-Line 2: Compilation result applicable to this note is produced when condition (1) and condition (2) through (5) in the example above are met.

[Assembler source code (correct compilation result)]

```
1:  _ll:
2:      .lword    0BE992000H,000000000H
```

-Line2: The correct result of compilation is as shown above.

2.5 Workaround

To avoid this problem, take either of the following steps.

- (1) Describe the constant expression in (2) as a converted integer-type constant. Refer to [Workaround example of C source code (1)] below.
- (2) Describe conversion as code-based runtime processing using either (2-1) or (2-2) below instead of a constant expression.
 - (2-1) Assign the value to the 64-bit integer type variable via double-type variable. Refer to [Workaround example of C source code (2-1)] below.
 - (2-2) Replace the variable with an inline function call that returns a double type value. Refer to [Workaround example of C source code (2-2)] below.

[Workaround example of C source code (1)]

```
long long ll = 123456789702ll;
```

[Workaround example of C source (2-1)]

```
1: void func(){
2:     double la = 123456789123.0+123+456; //Define la of a double type variable
3:     long long ll = la;                  //Assign the value of la to the 64-bit
4:                                         //integer type
5: }
```

[Workaround example of C source (2-2)]

```
1: #pragma inline dvalue
2: static double dvalue(){
3:     return 123456789123.0+123+456; //Define the inline function returning double
4:                                     //type value
5: }
6:
7: void func(){
8:     long long ll = dvalue(); //Call the inline function and assign it to the
9:                             //variable ll
10: }
```

2.6 Schedule for Fixing the Problem

This problem is fixed in CC-RX V3.01.00. (Scheduled to be released on January 21.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan. 16, 2019	-	First edition issued

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061 Japan
 Renesas Electronics Corporation

■Inquiry

<https://www.renesas.com/contact/>

Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

The past news contents have been based on information at the time of publication. Now changed or invalid information may be included.

The URLs in the Tool News also may be subject to change or become invalid without prior notice.

All trademarks and registered trademarks are the property of their respective owners.