MISRA C:2012
Technical Corrigendum 1

Technical clarification of MISRA C:2012

June 2017

First published June 2017 by HORIBA MIRA Limited
Watling Street
Nuneaton
Warwickshire
CV10 0TU
UK

www.misra.org.uk

ISBN 978-1-906400-17-0 PDF

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

MISRA C:2012
Technical Corrigendum 1

Technical clarification of
MISRA C:2012

June 2017

MISRA Mission Statement

We provide world-leading best practice guidelines for the safe and secure application of both embedded control systems and standalone software.

MISRA is a collaboration between manufacturers, component suppliers and engineering consultancies which seeks to promote best practice in developing safety- and security-related electronic systems and other software-intensive applications. To this end MISRA publishes documents that provide accessible information for engineers and management, and holds events to permit the exchange of experiences between practitioners.

Foreword

Since the publication of MISRA C:2012 [1] and its adoption by industry and the wider C community, a number of issues have arisen, both from discussions within the MISRA C Working Group and in response to feedback via the MISRA C Forum [2].

This document provides clarification on these issues, and should be read in conjunction with the original MISRA C:2012 document.

Andrew Banks FBCS CITP
Chairman, MISRA C Working Group

Contents

1 Clarification of directives

Dir 4.6

Issue

It is unclear whether a typedef is required to be used in place of “plain” *char* type. Exception 4 is not relevant to the rule as “plain” *char* types are not required to be replaced by a typedef.

Correction

Add sentence at end of first paragraph of Amplification:

The numerical types of *char* are *signed char* and *unsigned char*. These Guidelines do not treat “plain” *char* as a numerical type (see section 8.10.2 on *essentially character* types).

Remove Exception 4:

4. For function “main” a char may be used rather than the typedefs for the input parameter argv.

Dir 4.8

Issue

The interpretation of this directive is unclear when there is more than one pointer to the same structure or union type.

Correction

Add sentence at end of Amplification:

This directive only applies if all the pointers to a particular structure or union in a translation unit are never dereferenced.

Dir 4.11

Issue

A correction is required to the C99 references.

Correction

Replace:

Implementation J.3(8-11)

with:

Implementation J.3.12(8-11)

2 Clarification of rules

Rule 2.2

Issue

It is unclear whether the term *dead code* includes initialization.

Correction

Add extra Note:

Note: Initialization is not the same as an assignment operation and is therefore not a candidate for *dead code*.

Rule 2.5

Issue

It is unclear whether undefining a macro is considered to be a *use* of the macro.

Correction

Add an Amplification:

#undef of a macro is considered to be a *use* of a macro.

Rule 5.9

Issue

It is unclear whether the Amplification only applies to identifiers that define objects or functions with internal linkage.

Correction

Replace:

The identifier should be unique...

with

An identifier name that defines objects or functions with internal linkage should be unique....

Rule 8.4

Issue

An exception is required for function `main`.

Correction

Add Exception.

The function `main` need not have a separate declaration.

Rule 10.1

Issue

The list of prohibited operators does not include objects with pointer type.

Correction

Add following paragraph after paragraph starting “Under this rule”:

In addition, the rule prohibits the use of logical operators (! && ||) on an operand with pointer type.

Rule 10.1

Issue

A correction is required to the C99 references.

Correction

Replace:

Implementation J3.4(2, 5), J3.5(5), J3.9(6)

with:

Implementation J.3.4(2, 5), J.3.5(5), J.3.9(6)

Rule 10.3

Issue

An exception is required to cover switch statements' case labels.

Correction

Add Exception 3.

A *switch* statement's *case* label that is a non-negative *integer constant expression* of *essentially signed type* is permitted when the controlling expression is of *essentially unsigned type* and the value can be represented in that type.

Rule 10.3

Issue

A correction is required to the C99 references.

Correction

Replace:

Implementation 3.5(4)

with:

Implementation J.3.5(4)

Rule 10.3

Issue

A correction is required to Exception 1.

Correction

Replace:

A non-negative *integer constant expression* of *essentially signed type* may be assigned to an object of *essentially unsigned type* if its value can be represented in that type.

with:

An *essentially signed integer constant expression*, with a rank no greater than signed int, may be assigned to an object of *essentially unsigned type* if its value can be represented in that type.

Rule 10.4

Issue

A correction is required to show which rule an example violates.

Correction

Remove example from “non-compliant” section:

```
u8a += cha /* unsigned and char */
```

Add new paragraph after “cha += u8a” example:

The following is compliant by exception 1, but violates Rule 10.3

```
u8a += cha /* unsigned and char */
```

Rule 10.4

Issue

A correction is required to the C99 references.

Correction

Replace:

Implementation 3.6(4)

with:

Implementation J.3.6(4)

Rule 10.5

Issue

It is unclear whether the exception applies to expressions with *essentially enum* type or to just those with an *essentially signed* or *essentially unsigned* type.

Correction

Replace:

An *integer constant expression* with the value 0 or 1 of either signedness ...

with:

An *integer constant expression* with the value 0 or 1 and either *essentially signed* or *essentially unsigned* type ...

Section 8.10.3

Issue

The list of composite operators is not complete.

Correction

Replace:

Bitwise (&, |, ^)

with:

Bitwise (&, |, ^, ~)

Replace:

- The result of a compound assignment operator is not a *composite expression*;

with:

- The results of the following operators are not *composite expressions*:
  - Assignment and compound assignment
  - Postfix and prefix increment and decrement
  - Cast

Add between 2nd and 3rd items in the “Note” list:

- A unary + or unary - expression whose operand is a *composite expression* is also a *composite expression*.

Rule 10.8

Issue

The wording in the rationale is not correct.

Correction

Change the Rationale line:

On a 16-bit machine the addition will be performed in 16-bits with the result wrapping modulo-2 before it is cast to 32-bits.

to:

On a 16-bit machine the addition will be performed in 16-bits with the result wrapping modulo-2^16 before it is cast to 32-bits.

Rule 11.2

Issue

It is unclear whether this rule applies to the unqualified types that are pointed to by the pointers.

Correction

Add a final paragraph to the Amplification:

This rule applies to the unqualified types that are pointed to by the pointers.

Rule 11.4

Issue

It is unclear whether this rule only applies to object pointers. *Note:* Other rules cover the other pointer types.

Correction

Change the Amplification lines:

A pointer should not be converted into an integer. An integer should not be converted into a pointer.

to:

An object pointer should not be converted into an integer. An integer should not be converted into an object pointer.

Rule 11.9

Issue

An exception is required to permit the use of { 0 } to initialize aggregates or unions containing pointers.

Correction

Add Exception:

The initializer { 0 } may be used to initialize an aggregate or union type containing pointers.

Rule 11.9

Issue

The example comment needs improvement.

Correction

Change:

```
/* Could also be stdio.h, stdlib.h and others */
#include <stddef.h>
```

to:

```
#include <stddef.h>  /* To obtain macro NULL */
/* Could also be stdio.h, stdlib.h and others in hosted environments */
```

Rule 12.4

Issue

It is unclear whether this rule applies just to expressions that do not violate the constraints of a *constant expression* or whether it also applies to expressions which exhibit undefined behaviour.

The example with “const uint16_t c” was included to reinforce this point.

Correction

Change the Amplification line:

This rule applies to expressions that satisfy the *constraints* for a *constant expression*, whether or not they appear in a context that requires a *constant expression*.

to:

This rule applies to expressions that satisfy the *constraints* and semantics for a *constant expression*, whether or not they appear in a context that requires a *constant expression*.

Change the Example line:

This rule does not apply to the expression `c + 1` in the following compliant example as it accesses an object and therefore does not satisfy the *constraints* for a *constant expression*.

to:

This rule does not apply to the expression `c + 1` in the following compliant example as it accesses an object and therefore does not satisfy the semantics for a *constant expression*.

Rule 13.2

Issue

There is an incorrect space between the macro name and the "(" in the definition of COPY_ELEMENT.

Correction

Change the Example line:

```
#define COPY_ELEMENT ( index ) ( a[( index )] = b[( index )] )
```

to:

```
#define COPY_ELEMENT( index ) ( a[( index )] = b[( index )] )
```

Rule 14.2

Issue

The meaning of the phrase "assign a value to the loop counter" is unclear. In particular, confirmation is required that the following is compliant.

```
int index;
for ( set_val(&index) ;  index < 10 ; index++) /* set_val assigns to index */
```

Correction

Change the second bullet point from:

- Shall assign a value to the *loop counter*, or

to:

- Shall be an expression whose only *persistent side effect* is to set the value of the *loop counter*, or

Rule 15.6

Issue

The *if .. else if* example is not compliant with rule 15.7.

Correction

Add a comment to the final else statement

```
else
{
  ;    /* no action */
}
```

Rule 15.7

Issue

It is unclear whether all function calls are to be considered as having a *side effect* for the purposes of this rule.

Correction

Add a second paragraph to the Amplification

A function call is considered to be a *side effect* for the purposes of this rule.

Rule 16.1

Issue

The font is incorrect on “opt” in C90 syntax of switch-clause.

Correction

Change:

C90: { *declaration-list*$_{opt}$ *statement-list*opt break; }

to:

C90: { *declaration-list*$_{opt}$ *statement-list*$_{opt}$ break; }

Rule 19.1

Issue

The assignment “a = b” is marked as compliant due to exception 1, but rule 19.1 does not apply since the objects are not overlapping.

Correction

Remove reference to “b” from the example. Replace:

```
} a = { 0 }, b = { 1 };

a.j = a.i;      /* Non-compliant */
a = b;          /* Compliant - exception 1 */
```

with:

```
} a = { 0 };

a.j = a.i;     /* Non-compliant */
```

Rule 21.1

Issue

There is a missing cross-reference to rule 20.5.

Correction

Add a cross-reference to 20.5 to the “See Also” section

Rule 21.2

Issue

The headline of this rule is inconsistent with that of rule 21.1.

Correction

Change:

A reserved identifier or macro name shall not be declared

to:

A reserved identifier or reserved macro name shall not be declared

Rule 21.2

Issue

The example _BUILTIN_sqrt is marked as non-compliant with this rule, but is actually non-compliant with rule 21.1.

Correction

Change:

```
#define _BUILTIN_sqrt( x ) ( x )    /* Non-compliant  */
```

to:

```
static double _BUILTIN_sqrt ( double x )      /* Non-compliant */
{
   return x * x;
}
```

Rule 21.7

Issue

The wording of the headline is inconsistent with other rules.

Correction

Change headline:

The *atof*, *atoi*, *atol* and *atoll* functions of <stdlib.h> shall not be used

to:

The Standard Library functions *atof*, *atoi*, *atol* and *atoll* of <stdlib.h> shall not be used

Rule 21.8

Issue

The wording of the headline is inconsistent with other rules.

Correction

Change headline:

The library functions *abort*, *exit*, *getenv* and *system* of `<stdlib.h>` shall not be used

to:

The Standard Library functions *abort*, *exit*, *getenv* and *system* of `<stdlib.h>` shall not be used

Rule 21.9

Issue

The wording of the headline is inconsistent with other rules.

Correction

Change headline:

The library functions *bsearch* and *qsort* of `<stdlib.h>` shall not be used

to:

The Standard Library functions *bsearch* and *qsort* of `<stdlib.h>` shall not be used

3 Clarification of appendices

Appendix D.3

Issue

It is unclear in some contexts where the *STLR* and *UTLR* should be used.

Correction

Add after last paragraph:

Note: The *STLR* and *UTLR* of an *integer constant expression* is only applied to those operators listed in D.7.

Appendix D.7

Issue

The parenthesis operator is missing from the list of operators in Appendix D.7.

Correction

Add:

Parenthesis (( ))

The *essential type* of the result is the *essential type* of the operand.

Appendix G

Issue

The first entry for J.3.11 does not exist in the original C99 document. It should be retained as an unnumbered item with the remaining items for J.3.11 being renumbered.

Correction

Renumber entries for J.3.11 from:

1, 2, 3, 4, 5, 6, 7, 9, 10, 11

to:

*, 1, 2, 3, 4, 5, 6, 8, 9, 10

Appendix J

Issue

It is unclear whether a function is considered to have a *persistent side effect* if only some paths through the function cause a *persistent side effect*.

Correction

Add to paragraph before example:

The determination of whether a function has *persistent side effects* takes no consideration of the possible values for parameters or other non-local objects.

4 References

[1] MISRA C:2012 *Guidelines for the use of the C language in critical systems*, ISBN 978-1-906400-10-X, MIRA, March 2013

[2] MISRA Web Forum at https://www.misra.org.uk/forum