MISRA C:2012 Technical Corrigendum 2

Technical clarification of MISRA C:2012

March 2022

First published March 2022 by The MISRA Consortium Limited
1 St James Court
Whitefriars
Norwich
Norfolk
NR3 1RU
UK

www.misra.org.uk

ISBN 978-1-911700-00-5 PDF

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

MISRA C:2012
Technical Corrigendum 2

Technical clarification of MISRA C:2012

March 2022

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

Subsequent to the publication of MISRA C:2012 [1], its enhancement by Amendment 1 [3] and Amendment 2 [4], 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 discussion forum [6].

This document provides clarification on these issues, and should be read in conjunction with:

- MISRA C:2012 (Third Edition, First Revision) *Guidelines for the use of the C language in critical systems* [2], as revised by:
  - MISRA C:2012 Amendment 2, *Updates for ISO/IEC 9899:2011 Core functionality* [4]

or

- MISRA C:2012 (Third Edition) *Guidelines for the use of the C language in critical systems* [1], as revised by:
  - MISRA C:2012 Amendment 1, *Additional security guidelines for MISRA C:2012* [3]
  - MISRA C:2012 Amendment 2, *Updates for ISO/IEC 9899:2011 Core functionality* [4]
  - MISRA C:2012 Technical Corrigendum 1 [5]

Andrew Banks FBCS CITP
Chairman, MISRA C Working Group

Contents

1 Clarification of introduction to guidelines

1.1 Section 6.9

Issue

The meaning of the phrase “Applies to” was unclear.

Correction

TC 2.1 : Insert new *Note 1* before the existing notes, which should be renumbered accordingly

1. Where a guideline does not apply to the chosen version of the C Standard, it is treated as “not applicable” for the purposes of MISRA Compliance [42].

2 Clarification of directives

2.1 Dir 4.10

Issue

It was unclear whether the examples given in the document were the only permitted methods of protecting the inclusion of header files.

Correction

TC 2.2 : In the “Example” section, replace:

In order to facilitate checking, the contents of the header should be protected from being included more than once using one of the following two forms:

with:

The following examples show two ways by which the contents of a header file could be protected from being included more than once in a translation unit, but this is not an exclusive list.

3 Clarification of rules

3.1 Rule 2.2

Issue

It is unclear whether a cast operator whose result is *used* is ever *dead code*.

Correction

TC 2.3 : In the "Exception" section, number existing exception as 1.

TC 2.4 : In the "Exception" section, add second exception:

2. A cast operator whose result is *used* is not *dead code*.

3.2 Rule 2.5

Issue

Incorrect terminology was used to describe macro definitions.

Correction

TC 2.5 : Replace the "Headline"

A project should not contain unused macro declarations

with:

A project should not contain unused macro definitions

TC 2.6 : In the "Rationale" section, replace:

If a macro is declared ...

with:

If a macro is defined ...

3.3 Rule 7.4

Issue

It is unclear whether Rule 7.4 applies to variadic functions.

Correction

TC 2.7 : Add a new "Exception" section:

Exception

This rule does not apply to a string literal passed as an argument to the variable argument list of a variadic function.

TC 2.8 : Add to the "Example" section:

This example shows the permitted exemption for variadic functions.

```
extern void f3( uint16_t x, ... );  /* Note: non-compliant with Rule 17.1        */
extern void f4( char *text, ... );  /* Note: non-compliant with Rule 17.1        */

void variadic( void )
{
  f3( 42u, "MISRA" );               /* Compliant by exception                    */
  f4( "MISRA", 42u );               /* Non-compliant - exception only applies to
                                                  variable argument lists   */
}
```

TC 2.9 : Add to the "See also" list:

Rule 17.1

3.4 Rule 8.2

Issue

Add new cross-reference to Rule 8.3.

Correction

TC 2.10 : Add to the "See also" list:

Rule 8.3

3.5 Rule 8.3

Issue

It is unclear whether Rule 8.3 applies if a parameter is not named in a declaration, but is named in the definition.

Correction

TC 2.11 : In the "Exception" section, number existing exception as 1.

TC 2.12 : In the "Exception" section, add second exception:

2. The naming requirements of this rule do not apply to unnamed function parameters. This is covered by Rule 8.2.

TC 2.13 : In the "Example" section, replace the first example with:

```
extern void f ( signed int a );
       void f (        int a );    /* Compliant     - Exception 1                */

extern void g ( signed int b );
extern void g ( signed int   );    /* Compliant     - Exception 2                */

extern void h ( int * const c );
extern void h ( int *       c );   /* Non-compliant - mis-matched type qualifiers */

extern void j ( int d );
extern void j ( int e );           /* Non-compliant - mis-matched parameter names */
```

Note: all the above are not compliant with Dir 4.6; example `g()` is also not compliant with Rule 8.2.

TC 2.14 : Add to the "See also" list:

Rule 8.2

3.6 Rule 8.7

Issue

An example is required in the documentation.

Correction

TC 2.15 : Add a new “Example” section:

Example

```
/* file.h */
extern void ext_fn1 ( void );  /* Compliant     */
extern void ext_fn2 ( void );  /* Non-compliant */

/* file1.c */
#include "file.h"
void ext_fn1 ( void )   /* Compliant     - defined in this translation unit,
                                           but used externally              */
{
  /* Function definition */
}

void ext_fn2 ( void )   /* Non-compliant - defined and used only
                                           in this translation unit         */
{
  /* Function definition */
}

void fn_file1 ( void )
{
  ext_fn2( );
}

/* file2.c */
#include "file.h"
void fn_file2 ( void )
{
  ext_fn1( );
}
```

3.7 Section 8.10.1

Issue

It is unclear whether the rules in Section 8.10 apply to expressions with a pointer type.

Correction

TC 2.16 : Add a paragraph to the end of Section 8.10.1:

The rules in this section do not apply to expressions with a pointer type, unless otherwise specified.

3.8 Rule 10.1

Issue

Rationale 6 should make reference to undefined behaviour.

Correction

TC 2.17 : In the “Rationale” section, replace the second sentence of rationale 6:

The numeric value resulting from their use on *essentially signed types* is implementation-defined.

with:

The numeric value resulting from their use on *essentially signed types* may be undefined or implementation-defined.

TC 2.18 : In the “Source ref” list, replace:

C99 [Undefined 13, 49; Implementation J.3.4(2, 5), J.3.5(5), J.3.9(6)]

with:

C99 [Undefined 13, 48, 49; Implementation J.3.4(2, 5), J.3.5(5), J.3.9(6)]

3.9 Rule 10.2

Issue

It is unclear whether the amplification applies to all *essentially signed* and *unsigned types*. The intention was that it should only apply to operands whose type had a rank equal or lower than the rank of *int*.

Correction

TC 2.19 : Replace the “Amplification” section with:

The appropriate uses are:

1. For the + operator, one operand shall have *essentially character type* and the other shall have *essentially signed type* or *essentially unsigned type* having a rank lower than or equal to that of *int*. The result of the operation has *essentially character type*.

2. For the - operator, the first operand shall have *essentially character type* and the second shall have:

- *essentially signed type* or *essentially unsigned type* or *essentially character type*; and
- a rank lower than or equal to that of *int*

If both operands have *essentially character type* then the result has the *standard type* (usually *int* in this case) else the result has *essentially character type*.

3.10 Rule 10.3

Issue

It is unclear whether Amplification 2 refers to the *essential type* or the C promoted type of the controlling expression.

Correction

TC 2.20 : In the "Amplification" section, replace amplification 2:

2. The conversion of the *constant expression* in a *switch* statement's *case* label to the promoted type of the controlling expression.

with:

2. The conversion of the *constant expression* in a *switch* statement's *case* label to the *essential type* of the controlling expression.

3.11 Rule 11.3

Issue

The Rule is missing a cross-reference to Rule 18.1.

Correction

TC 2.21 : Add to the "See also" section:

Rule 18.1

3.12 Rule 11.6

Issue

The conversion of an integer into a pointer to *void* was incorrectly described as resulting in undefined behaviour.

Correction

TC 2.22 : In the "Amplification" section, replace the second paragraph:

Conversion of an integer into a pointer to *void* may result in a pointer that is not correctly aligned, resulting in undefined behaviour.

with:

Conversion of an integer into a pointer to *void* results in behaviour that is implementation-defined.

3.13 Rule 13.2

Issue

It is unclear whether this rule takes into account the value of an object. For example:

```
int a = 0;
if (  ( x = a ) && ( x = b ) )
```

The second assignment to `x` will never take place if `a` is known to be `0`. The intention was that the value of the object should not be considered and the above example would be non-compliant with this rule.

Correction

TC 2.23 : In the "Amplification" section, number the existing two notes as 1 and 3 respectively.

TC 2.24 : In the "Amplification" section, insert a new note 2 between the existing two notes:

2. All parts of the expression are considered when determining whether an object is read or modified, irrespective of any known values.

TC 2.25 : In the "Rationale" section, replace the final paragraph:

Many of the common instances of the unpredictable behaviour associated with expression evaluation can be avoided by following the advice given by Rule 13.3 and Rule 13.4.

with:

Many of the common instances of the unpredictable behaviour, associated with expression evaluation, can be avoided by following the advice given by this rule, by Rule 13.3, and by Rule 13.4. However, in order to simplify this rule, it does restrict some forms which are well-defined.

3.14 Rule 13.6

Issue

There are use cases for violating `sizeof( expressions )`, which the current *mandatory* category prevents.

Correction

TC 2.26 : Revise the "Category" from *Mandatory* to *Required*.

3.15 Rule 14.3

Issue

Exception 2 was intended to permit only a constant expression, but the current wording permits expressions that evaluate to 0 at run-time.

Correction

TC 2.27 : In the "Exception" section, replace exception 2:

A do … while loop with an *essentially Boolean* controlling expression that evaluates to 0 is permitted.

with:

A do … while loop with an *essentially Boolean* controlling expression that evaluates to *false* and satisfies the *constraints* and semantics for an *integer constant expression* is permitted.

TC 2.28 : In the "Example" section, add a new example:

```
do
{
  /* Non-compliant - not covered by exception 2 */
} while ( (s8a < 10) && (s8a > 20) );
```

3.16 Rule 15.7

Issue

The cross-reference to Rule 16.5 should read Rule 16.4.

Correction

TC 2.29 : In the "See also" section, replace:

See also: Rule 16.5

with:

See also: Rule 16.4

3.17 Rule 17.4

Issue

Enforcing this rule for the `main` function may conflict with Rule 2.1.

Correction

TC 2.30 : Add a new “Exception” section:

Exception

For C99 and later, The Standard specifies that if control reaches the end of `main` without encountering a `return` statement, the effect is that of executing `return 0`. Therefore, for C99 and later, the `return` statement may be omitted for function `main`.

3.18 Rule 17.5

Issue

There is an inconsistency between the headline and the category, in that the headline uses *shall* (denoting *Required*) but the guideline is incorrectly categorized as *Advisory*.

Correction

TC 2.31 : Revise the “Category” from *Advisory* to *Required*.

Correction

TC 2.32 : In the “Amplification” section, replace *should* with *shall*.

3.19 Rule 18.1

Issue

It is unclear whether the following code violates Rule 18.1:

```
uint32_t variable = .... ;
uint8_t *ptr = (uint8_t *) &variable;
uint8_t u8 = ptr[ 3 ];  // Treat ptr as array of 4 8-bit objects
```

Correction

TC 2.33 : In the “Amplification” section, add a note:

Note: A pointer to an object of type `T` which has been converted to a pointer to an object of type `char`, `signed char` or `unsigned char` (see exception to Rule 11.3) is treated as an array of that type with bound equal to `sizeof(T)`.

TC 2.34 : Add to the “See also” list:

Rule 11.3

3.20 Rule 20.14

Issue

Incorrect filename in example.

Correction

TC 2.35 : In the “Example” section, replace the final line:

```
/* End of file1.h */
```

with:

```
/* End of file2.h */
```

3.21 Rule 21.19

Issue

A cross-reference intended for use only within MISRA C:2012 Amendment 1 [3] has been incorporated into the consolidated document.

Correction

TC 2.36 : In the “See also” section, delete the reference to Rule 21.8.

3.22 Rule 21.20

Issue

The current wording is imprecise with respect to the sets of related library functions.

Correction

TC 2.37 : Replace the “Amplification” section with:

For the purposes of this rule:

- a call to `setlocale` function following a call to `localeconv` function shall be treated as if they are calls to the same function.
- the `asctime` and `ctime` functions shall be treated as if they are the same function.
- the `gmtime` and `localtime` functions shall be treated as if they are the same function.

3.23 Rule 22.9

Issue

An incorrect headline was printed in the hard copy version of the First Revision of the Third Edition of the MISRA C:2012 guidelines. The correct headline appears in the PDF version and in Appendix A of the hard copy.

Correction

TC 2.38 : Replace the "Headline":

The value of `errno` shall be set to zero after calling an *errno-setting-function*

with:

The value of `errno` shall be tested against zero after calling an *errno-setting-function*

4 Clarification of appendices

4.1 Appendix A

Consequential amendments

TC 2.39 : For Rule 2.5 update the “Headline” replacing *declarations* with *definitions*.

TC 2.40 : For Rule 13.6, revise the “Category” from *Mandatory* to *Required*.

TC 2.41 : For Rule 17.5, revise the “Category” from *Advisory* to *Required*.

4.2 Appendix B

Consequential amendments

TC 2.42 : For Rule 13.6, update the “Category” from *Mandatory* to *Required*.

TC 2.43 : For Rule 17.5, update the “Category” from *Advisory* to *Required*.

4.3 Appendix C

Issue

It is unclear if the unqualified term *conversion* means implicit or explicit conversions, or both.

Correction

TC 2.44 : Insert a *Note* immediately before the paragraph starting *An explanation…*.

Note: Conversions may be implicit or explicit (i.e. by means of a cast) – where the term *conversion* is used without qualification it means either or both forms as the situation requires.

4.4 Appendix D

4.4.1 Appendix D.1

Issue

It is unclear what is the *essential type* of `ptrdiff_t`, `size_t`, `intptr_t` and other types defined in the C Standard Library header files.

Correction

TC 2.45 : Replace

The *essential type* of an expression only differs from the standard C type (*standard type*) in expressions where the *standard type* is either *signed int* or *unsigned int*.

with

The *essential type* of an expression only differs from the standard C type (*standard type*) in expressions where the *standard type* is either *signed int* or *unsigned int*.

The *essential type* of an expression with a C type defined in a C Standard Library header is that in which it is implemented. For example, `int_least8_t` may be implemented as *signed int* and will have an essential type of *essentially signed int* in that case.

4.4.2 Appendix D.7

Issue

The operations listed in D.7.9 overlap with those in D.7.10 and D.7.11. For example, these changes clarify that addition between operands with *essentially character* and *essentially long* types will result in an expression with an *essentially long* type.

Correction

TC 2.46 : Replace Sections D.7.9, D.7.10 and D.7.11 with a single revised section D.7.9

D.7.9 Operations subject to the usual arithmetic conversions (* / % + - & | ^)

1. If the operator is + and one operand is *essentially character* and the other is *essentially signed* or *essentially unsigned* having a rank lower than or equal to that of *int* then the *essential type* of the result is *char*;

2. Else if the operator is - and the first operand is *essentially character* type and the second is *essentially signed* or *essentially unsigned* having a rank lower than or equal to that of *int* then the *essential type* of the result is *char*;

3. Else if the operands are both *essentially signed* then:

   3.1 If the expression is an *integer constant expression* then the *essential type* of the result is the *STLR* of the result;

   3.2 Else the *essential type* of the result is the *essential type* of the operand with the highest rank.

4. Else if the operands are both *essentially unsigned* then:

   4.1 If the expression is an *integer constant expression* then the *essential type* of the result is the *UTLR* of the result;

   4.2 Else the *essential type* of the result is the *essential type* of the operand with the highest rank.

5. Else the *essential type* is the *standard type*.

5 References

[1] MISRA C:2012 (Third Edition) *Guidelines for the use of the C Language in critical systems*,
ISBN 978-1-906400-10-1 (paperback) 978-1-906400-11-8 (PDF),
MIRA, March 2013

[2] MISRA C:2012 (Third Edition, First Revision) *Guidelines for the use of the C Language in critical systems*,
ISBN 978-1-906400-21-7 (paperback) 978-1-906400-22-4 (PDF),
HORIBA MIRA Limited, February 2019

[3] MISRA C:2012 Amendment 1, *Additional security guidelines for MISRA C:2012*,
ISBN 978-1-906400-16-3 PDF,
HORIBA MIRA Limited, April 2016

[4] MISRA C:2012 Amendment 2, *Updates for ISO/IEC 9899:2011 core functionality*,
ISBN 978-1-906400-25-5 PDF,
HORIBA MIRA Limited, February 2020

[5] MISRA C:2012 Technical Corrigendum 1,
ISBN 978-1-906400-17-0 PDF,
HORIBA MIRA Limited, June 2017

[6] MISRA discussion forum at https://forum.misra.org.uk/