**MISRA:**

MISRA C is a set of software development guidelines for the C programming language developed by MISRA (Motor Industry Software Reliability Association). Its aims are to facilitate code safety, security, portability and reliability in the context of embedded systems, specifically those systems programmed in ISO C / C90 / C99.

There is also a set of guidelines for MISRA C++ not covered by this article.

**NAMING CONVENTIONS COMPILATION (MISRA BASED):**

**[C1] All code shall conform to ISO 9899:1990 “Programming languages – C”, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996.**

7.1.a. No variable shall have a name that is a keyword of C, C++, or any other well-known extension of the C programming language, including specifically K&R C and C99. Restricted names include interrupt, inline, restrict, class, true, false, public, private,friend, and protected.

7.1.b. No variable shall have a name that overlaps with a variable name from the C Standard Library (e.g., errno).

7.1.c. No variable shall have a name that begins with an underscore.

7.1.d. No variable name shall be longer than 31 characters.

7.1.e. No variable name shall be shorter than 3 characters, including loop counters.

7.1.f. No variable name shall contain any uppercase letters.

7.1.g. No variable name shall contain any numeric value that is called out elsewhere, such as the number of elements in an array or the number of bits in the underlying type.

7.1.h. Underscores shall be used to separate words in variable names.

7.1.i. Each variable’s name shall be descriptive of its purpose.

7.1.j. The names of any global variables shall begin with the letter ‘g’. For example, g\_zero\_offset.

7.1.k. The names of any pointer variables shall begin with the letter ‘p’. For example, p\_led\_reg.

7.1.l. The names of any pointer-to-pointer variables shall begin with the letters ‘pp’. For example, pp\_vector\_table.

7.1.m. The names of all integer variables containing Boolean information (including 0 vs. non-zero) shall begin with the letter ‘b’ and phrased as the question they answer. For example, b\_done\_yet or b\_is\_buffer\_full.

7.1.n. The names of any variables representing non-pointer handles for objects, e.g., file handles, shall begin with the letter ‘h’. For example, h\_input\_file.

7.1.o. In the case of a variable name requiring multiple of the above prefixes, the order of their inclusion before the first underscore shall be [g][p|pp][b|h].

Reasoning: The base rules are adopted to maximize code portability across compilers. Many C compilers recognize differences only in the first 31 characters in a variable’s name and reserve names beginning with an underscore for internal names.

The other rules are meant to highlight risks and ensure consistent proper use of variables. For example, all code relating to the use of global variables and other singleton objects, including peripheral registers, needs to be carefully considered to ensure there can be no race conditions or data corruptions via asynchronous writes.

**MORE C CODING STANDARDS:**

* [C4] Filenames shall be all lower case alphanumeric characters with no white-space characters.
* [C10] All files shall compile without any errors or warnings.
* [C16] Tab characters shall not be used.
* [C17] All unused declarations and definitions shall be removed.
* C23] Header files shall not contain object or normal function definitions.
* [C25] Header files shall not include themselves directly or indirectly.
* [C30] Comments shall be in English.
* [C31] Comments shall be up-to-date.
* [C32] Source code shall only use /\* ... \*/ style comments.
* [C34] Comment characters shall not be nested within other comments.
* [C61] Identifiers (internal and external) shall not rely on the significance of more than 31 characters
* [C64] Identifiers should be given meaningful names.
* [C65] Identifiers shall be named in English.
* [C66.1] Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier
* [C70] Magic numbers shall not be embedded throughout the source code.
* [C76] In an enumerator list, the “=” construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.
* [C80] The equality operators (==, !=) shall not have floating-point operands other than zero.
* [C85] Floating-point computations should use <float.h> and <math.h> for portability.
* [C90] Each object declaration and object definition shall appear on a line by itself.
* [C92] Objects shall be defined at block scope if they are only accessed from within a single function.
* [C102] All objects shall be initialized before being read.
* [C110] Selection statements shall be used in preference to the ternary operator (?:).
* [C232] The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.

**Published documents**

MISRA C:1998

The first edition of MISRA C, "Guidelines for the use of the C language in vehicle based software", which was published in 1998 and is officially known as MISRA-C:1998.

MISRA-C:1998 has 127 rules, of which 93 are required and 34 are advisory; the rules are numbered in sequence from 1 to 127.

MISRA C:2004

In 2004, a second edition "Guidelines for the use of the C language in critical systems", or MISRA-C:2004 was produced, with many substantial changes to the guidelines, including a complete renumbering of the rules.

MISRA-C:2004 contains 142 rules, of which 122 are "required" and 20 are "advisory"; they are divided into 21 topical categories, from "Environment" to "Run-time failures".

MISRA C:2012

In 2013, MISRA C:2012 was announced. MISRA C:2012 extends support to the C99 version of the C language (while maintaining guidelines for C90), in addition to including a number of improvements that can reduce the cost and complexity of compliance, whilst aiding consistent, safe use of C in critical systems.

MISRA-C:2012 contains 143 rules and 16 "directives" (that is, rules whose compliance is more open to interpretation, or relates to process or procedural matters); each of which is classified as mandatory, required, or advisory. They are separately classified as either Single Translation Unit or System. Additionally, the rules are classified as Decidable or Undecidable.

Amendment 1

In April 2016, MISRA published (as free downloads) Amendment 1 to MISRA C:2012 which added fourteen new security guidelines.

Addendum 2 and 3

In January 2018, MISRA published 2 addenda to the MISRA C:2012:

MISRA C:2012 - Addendum 2:  Coverage of MISRA C:2012 against ISO/IEC TS 17961:2013 "C Secure"

MISRA C:2012 - Addendum 3:  Coverage of MISRA C:2012 against CERT C