**MISRA:**

MISRA C:2012 was published on 18 March 2013. MISRA C:2012 extends support to the C99 version of the language whilst maintaining guidelines for C90. Other improvements, many of which have been made as a result of user feedback, include: better rationales for every guideline, identified decidability so users can better interpret the output of checking tools, greater granularity of rules to allow more precise control, a number of expanded examples and integration of MISRA AC AGC. A cross reference for ISO 26262 has also been produced.

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rule | Summary | Guideline Description | Category | Related Coverity Checker |
| Directive 1.1 | Directive | Any implementation-defined behaviour on which the output of the program depends shall be documented and understood. | Required |  |
| Directive 2.1 | Directive | All source files shall compile without any compilation errors. | Required |  |
| Directive 3.1 | Directive | All code shall be traceable to documented requirements. | Required |  |
| Directive 4.1 | Directive | Run-time failures shall be minimized. | Required |  |
| Directive 4.2 | Directive | All usage of assembly language should be documented. | Advisory |  |
| Directive 4.3 | Directive | Assembly language shall be encapsulated and isolated. | Required | MISRA C-2012 Directive 4.3 |
| Directive 4.4 | Directive | Sections of code should not be "commented out". | Advisory | MISRA C-2012 Directive 4.4 |
| Directive 4.5 | Directive | Identifiers in the same name space with overlapping visibility should be typographically unambiguous. | Advisory | MISRA C-2012 Directive 4.5 |
| Directive 4.6 | Directive | Typedefs that indicate size and signedness should be used in place of the basic numerical types. | Advisory | MISRA C-2012 Directive 4.6 |
| Directive 4.7 | Directive | If a function returns error information, then that error information shall be tested. | Required | MISRA C-2012 Directive 4.7 |
| Directive 4.8 | Directive | If a pointer to a structure or union is never dereferenced within a Translation Unit, then the implementation of the object should be hidden. | Advisory | MISRA C-2012 Directive 4.8 |
| Directive 4.9 | Directive | A function should not be used in preference to a function-like macro where they are interchangeable. | Advisory | MISRA C-2012 Directive 4.9 |
| Directive 4.10 | Directive | Precautions shall be taken in order to prevent the contents of a header file being included more than once. | Required | MISRA C-2012 Directive 4.10 |
| Directive 4.11 | Directive | The validity of values passed to library functions shall be checked. | Required | MISRA C-2012 Directive 4.11 |
| Directive 4.12 | Directive | Dynamic memory allocation shall not be used. | Required | MISRA C-2012 Directive 4.12 |
| Directive 4.13 | Directive | Functions which are designed to provide operations on a resource should be called in an appropriate sequence. | Advisory | MISRA C-2012 Directive 4.13 |
| Directive 4.14 | Directive | The validity of values received from external sources shall be checked. | Required | MISRA C-2012 Directive 4.14 |
| Rule 1.1 | Rule | The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits. | Required | MISRA C-2012 Rule 1.1 |
| Rule 1.2 | Rule | Language extensions should not be used. | Advisory | MISRA C-2012 Rule 1.2 |
| Rule 1.3 | Rule | There shall be no occurrence of undefined or critical unspecified behaviour. | Required |  |
| Rule 2.1 | Rule | A project shall not contain unreachable code. | Required | MISRA C-2012 Rule 2.1 |
| Rule 2.2 | Rule | There shall be no dead code. | Required | MISRA C-2012 Rule 2.2 |
| Rule 2.3 | Rule | A project should not contain unused type declarations. | Advisory | MISRA C-2012 Rule 2.3 |
| Rule 2.4 | Rule | A project should not contain unused tag declarations. | Advisory | MISRA C-2012 Rule 2.4 |
| Rule 2.5 | Rule | A project should not contain unused macro declarations. | Advisory | MISRA C-2012 Rule 2.5 |
| Rule 2.6 | Rule | A function should not contain unused label declarations. | Advisory | MISRA C-2012 Rule 2.6 |
| Rule 2.7 | Rule | There should be no unused parameters in functions. | Advisory | MISRA C-2012 Rule 2.7 |
| Rule 3.1 | Rule | The character sequences /\* and // shall not be used within a comment. | Required | MISRA C-2012 Rule 3.1 |
| Rule 3.2 | Rule | Line-splicing shall not be used in // comments. | Required | MISRA C-2012 Rule 3.2 |
| Rule 4.1 | Rule | Octal and hexadecimal escape sequences shall be terminated. | Required | MISRA C-2012 Rule 4.1 |
| Rule 4.2 | Rule | Trigraphs should not be used. | Advisory | MISRA C-2012 Rule 4.2 |
| Rule 5.1 | Rule | External identifiers shall be distinct. | Required | MISRA C-2012 Rule 5.1 |
| Rule 5.2 | Rule | Identifiers declared in the same scope and name space shall be distinct. | Required | MISRA C-2012 Rule 5.2 |
| Rule 5.3 | Rule | An identifier declared in an inner scope shall not hide an identifier declared in an outer scope. | Required | MISRA C-2012 Rule 5.3 |
| Rule 5.4 | Rule | Macro identifiers shall be distinct. | Required | MISRA C-2012 Rule 5.4 |
| Rule 5.5 | Rule | Identifiers shall be distinct from macro names. | Required | MISRA C-2012 Rule 5.5 |
| Rule 5.6 | Rule | A typedef name shall be a unique identifier. | Required | MISRA C-2012 Rule 5.6 |
| Rule 5.7 | Rule | A tag name shall be a unique identifier. | Required | MISRA C-2012 Rule 5.7 |
| Rule 5.8 | Rule | Identifiers that define objects or functions with external linkage shall be unique. | Required | MISRA C-2012 Rule 5.8 |
| Rule 5.9 | Rule | Identifiers that define objects or functions with internal linkage should be unique. | Advisory | MISRA C-2012 Rule 5.9 |
| Rule 6.1 | Rule | Bit-fields shall only be declared with an appropriate type. | Required | MISRA C-2012 Rule 6.1 |
| Rule 6.2 | Rule | Single-bit named bit fields shall not be of a signed type. | Required | MISRA C-2012 Rule 6.2 |
| Rule 7.1 | Rule | Octal constants shall not be used. | Required | MISRA C-2012 Rule 7.1 |
| Rule 7.2 | Rule | A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type. | Required | MISRA C-2012 Rule 7.2 |
| Rule 7.3 | Rule | The lowercase character "l" shall not be used in a literal suffix. | Required | MISRA C-2012 Rule 7.3 |
| Rule 7.4 | Rule | A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char". | Required | MISRA C-2012 Rule 7.4 |
| Rule 8.1 | Rule | Types shall be explicitly specified. | Required | MISRA C-2012 Rule 8.1 |
| Rule 8.2 | Rule | Function types shall be in prototype form with named parameters. | Required | MISRA C-2012 Rule 8.2 |
| Rule 8.3 | Rule | All declarations of an object or function shall use the same names and type qualifiers. | Required | MISRA C-2012 Rule 8.3 |
| Rule 8.4 | Rule | A compatible declaration shall be visible when an object or function with external linkage is defined. | Required | MISRA C-2012 Rule 8.4 |
| Rule 8.5 | Rule | An external object or function shall be declared once in one and only one file. | Required | MISRA C-2012 Rule 8.5 |
| Rule 8.6 | Rule | An identifier with external linkage shall have exactly one external definition. | Required | MISRA C-2012 Rule 8.6 |
| Rule 8.7 | Rule | Functions and objects should not be defined with external linkage if they are referenced in only one translation unit. | Advisory | MISRA C-2012 Rule 8.7 |
| Rule 8.8 | Rule | The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage. | Required | MISRA C-2012 Rule 8.8 |
| Rule 8.9 | Rule | An object should be defined at block scope if its identifier only appears in a single function. | Advisory | MISRA C-2012 Rule 8.9 |
| Rule 8.10 | Rule | An inline function shall be declared with the static storage class. | Required | MISRA C-2012 Rule 8.10 |
| Rule 8.11 | Rule | When an array with external linkage is declared, its size should be explicitly specified. | Advisory | MISRA C-2012 Rule 8.11 |
| Rule 8.12 | Rule | Within a n enumerator list, the value of an implicitly-specified enumeration constant shall be unique. | Required | MISRA C-2012 Rule 8.12 |
| Rule 8.13 | Rule | A pointer should point to a const-qualified type whenever possible. | Advisory | MISRA C-2012 Rule 8.13 |
| Rule 8.14 | Rule | The restrict type qualifier shall not be used. | Required | MISRA C-2012 Rule 8.14 |
| Rule 9.1 | Rule | The value of an object with automatic storage duration shall not be read before it has been set. | Mandatory | MISRA C-2012 Rule 9.1 |
| Rule 9.2 | Rule | The initializer for an aggregate or union shall be enclosed in braces. | Required | MISRA C-2012 Rule 9.2 |
| Rule 9.3 | Rule | Arrays shall not be partially initialized. | Required | MISRA C-2012 Rule 9.3 |
| Rule 9.4 | Rule | An element of an object shall not be initialized more than once. | Required | MISRA C-2012 Rule 9.4 |
| Rule 9.5 | Rule | Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly. | Required | MISRA C-2012 Rule 9.5 |
| Rule 10.1 | Rule | Operands shall not be of an inappropriate essential type. | Required | MISRA C-2012 Rule 10.1 |
| Rule 10.2 | Rule | Expressions of essentially character type shall not be used inappropriately in addition and subtraction operation. | Required | MISRA C-2012 Rule 10.2 |
| Rule 10.3 | Rule | The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category. | Required | MISRA C-2012 Rule 10.3 |
| Rule 10.4 | Rule | Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category. | Required | MISRA C-2012 Rule 10.4 |
| Rule 10.5 | Rule | The value of an expression should not be cast to an inappropriate essential type. | Advisory | MISRA C-2012 Rule 10.5 |
| Rule 10.6 | Rule | The value of a composite expression shall not be assigned to an object with wider essential type. | Required | MISRA C-2012 Rule 10.6 |
| Rule 10.7 | Rule | If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type. | Required | MISRA C-2012 Rule 10.7 |
| Rule 10.8 | Rule | The value of a composite expression shall not be cast to a different essential type category or a wider essential type. | Required | MISRA C-2012 Rule 10.8 |
| Rule 11.1 | Rule | Conversions shall not be performed between a pointer to a function and any other type. | Required | MISRA C-2012 Rule 11.1 |
| Rule 11.2 | Rule | Conversions shall not be performed between a pointer to an incomplete type and any other type. | Required | MISRA C-2012 Rule 11.2 |
| Rule 11.3 | Rule | A cast shall not be performed between a pointer to object type and a pointer to a different object type. | Required | MISRA C-2012 Rule 11.3 |
| Rule 11.4 | Rule | A conversion should not be performed between a pointer to object and an integer type. | Advisory | MISRA C-2012 Rule 11.4 |
| Rule 11.5 | Rule | A conversion should not be performed from pointer to void into pointer to object. | Advisory | MISRA C-2012 Rule 11.5 |
| Rule 11.6 | Rule | A cast shall not be performed between pointer to void and an arithmetic type. | Required | MISRA C-2012 Rule 11.6 |
| Rule 11.7 | Rule | A cast shall not be performed between pointer to object and a non-integer arithmetic type. | Required | MISRA C-2012 Rule 11.7 |
| Rule 11.8 | Rule | A cast shall not remove any const or volatile qualification from the type pointed to by a pointer. | Required | MISRA C-2012 Rule 11.8 |
| Rule 11.9 | Rule | The macro NULL shall be the only permitted form of integer null pointer constant. | Required | MISRA C-2012 Rule 11.9 |
| Rule 12.1 | Rule | The precedence of operators within expressions should be made explicit. | Advisory | MISRA C-2012 Rule 12.1 |
| Rule 12.2 | Rule | The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand. | Required | MISRA C-2012 Rule 12.2 |
| Rule 12.3 | Rule | The comma operator should not be used. | Advisory | MISRA C-2012 Rule 12.3 |
| Rule 12.4 | Rule | Evaluation of constant expressions should not lead to unsigned integer wrap-around. | Advisory | MISRA C-2012 Rule 12.4 |
| Rule 12.5 | Rule | The sizeof operator shall not have an operand which is a function parameter declared as "array of type". | Mandatory | MISRA C-2012 Rule 12.5 |
| Rule 13.1 | Rule | Initializer lists shall not contain persistent side effects. | Required | MISRA C-2012 Rule 13.1 |
| Rule 13.2 | Rule | The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders. | Required | MISRA C-2012 Rule 13.2 |
| Rule 13.3 | Rule | A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator. | Advisory | MISRA C-2012 Rule 13.3 |
| Rule 13.4 | Rule | The result of an assignment operator should not be used. | Advisory | MISRA C-2012 Rule 13.4 |
| Rule 13.5 | Rule | The right hand operand of a logical && or || operator shall not contain persistent side effects. | Required | MISRA C-2012 Rule 13.5 |
| Rule 13.6 | Rule | The operand of the sizeof operator shall not contain any expression which has potential side effects. | Mandatory | MISRA C-2012 Rule 13.6 |
| Rule 14.1 | Rule | A loop counter shall not have essentially floating type. | Required | MISRA C-2012 Rule 14.1 |
| Rule 14.2 | Rule | A for loop shall be well-formed. | Required | MISRA C-2012 Rule 14.2 |
| Rule 14.3 | Rule | Controlling expressions shall not be invariant. | Required | MISRA C-2012 Rule 14.3 |
| Rule 14.4 | Rule | The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type. | Required | MISRA C-2012 Rule 14.4 |
| Rule 15.1 | Rule | The goto statement should not be used. | Advisory | MISRA C-2012 Rule 15.1 |
| Rule 15.2 | Rule | The goto statement shall jump to a label declared later in the same function. | Required | MISRA C-2012 Rule 15.2 |
| Rule 15.3 | Rule | Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement. | Required | MISRA C-2012 Rule 15.3 |
| Rule 15.4 | Rule | There should be no more than one break or goto statement used to terminate any iteration statement. | Advisory | MISRA C-2012 Rule 15.4 |
| Rule 15.5 | Rule | A function should have a single point of exit at the end. | Advisory | MISRA C-2012 Rule 15.5 |
| Rule 15.6 | Rule | The body of an iteration-statement or a selection-statement shall be a compound statement. | Required | MISRA C-2012 Rule 15.6 |
| Rule 15.7 | Rule | All if ... else if constructs shall be terminated with an else statement. | Required | MISRA C-2012 Rule 15.7 |
| Rule 16.1 | Rule | All switch statements shall be well formed. | Required | MISRA C-2012 Rule 16.1 |
| Rule 16.2 | Rule | A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement. | Required | MISRA C-2012 Rule 16.2 |
| Rule 16.3 | Rule | An unconditional break statement shall terminate every switch-clause. | Required | MISRA C-2012 Rule 16.3 |
| Rule 16.4 | Rule | Every switch statement shall have a default label. | Required | MISRA C-2012 Rule 16.4 |
| Rule 16.5 | Rule | A default label shall appear as either the first or the last switch label of a switch statement. | Required | MISRA C-2012 Rule 16.5 |
| Rule 16.6 | Rule | Every switch statement shall have at least two switch clauses. | Required | MISRA C-2012 Rule 16.6 |
| Rule 16.7 | Rule | A switch expression shall not have an essentially Boolean type. | Required | MISRA C-2012 Rule 16.7 |
| Rule 17.1 | Rule | The features of <stdarg.h> shall not be used. | Required | MISRA C-2012 Rule 17.1 |
| Rule 17.2 | Rule | Functions shall not call themselves, either directly or indirectly. | Required | MISRA C-2012 Rule 17.2 |
| Rule 17.3 | Rule | A function shall not be declared implicitly. | Mandatory | MISRA C-2012 Rule 17.3 |
| Rule 17.4 | Rule | All exit paths from a function with non-void return type shall have an explicit return statement with an expression. | Mandatory | MISRA C-2012 Rule 17.4 |
| Rule 17.5 | Rule | The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements. | Advisory | MISRA C-2012 Rule 17.5 |
| Rule 17.6 | Rule | The declaration of an array parameter shall not contain the static keyword between the []. | Mandatory | MISRA C-2012 Rule 17.6 |
| Rule 17.7 | Rule | The value returned by a function having non-void return type shall be used. | Required | MISRA C-2012 Rule 17.7 |
| Rule 17.8 | Rule | A function parameter should not be modified. | Advisory | MISRA C-2012 Rule 17.8 |
| Rule 18.1 | Rule | A pointer resulting from arithmetic on a pointer operand shall address an elements of the same array as that pointer operand. | Required | MISRA C-2012 Rule 18.1 |
| Rule 18.2 | Rule | Subtraction between pointers shall only be applied to pointers that address elements of the same array. | Required | MISRA C-2012 Rule 18.2 |
| Rule 18.3 | Rule | The relational operators >, >=, < and <= shall only be applied to pointers that point into the same object. | Required | MISRA C-2012 Rule 18.3 |
| Rule 18.4 | Rule | The +, -, += and -= operators should not be applied to an expression of pointer type. | Advisory | MISRA C-2012 Rule 18.4 |
| Rule 18.5 | Rule | Declarations should contain no more than two levels of pointer nesting. | Advisory | MISRA C-2012 Rule 18.5 |
| Rule 18.6 | Rule | The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist. | Required | MISRA C-2012 Rule 18.6 |
| Rule 18.7 | Rule | Flexible array members shall not be declared. | Required | MISRA C-2012 Rule 18.7 |
| Rule 18.8 | Rule | Variable-length array types shall not be used. | Required | MISRA C-2012 Rule 18.8 |
| Rule 19.1 | Rule | An object shall not be assigned or copied to an overlapping object. | Mandatory | MISRA C-2012 Rule 19.1 |
| Rule 19.2 | Rule | The union keyword should not be used. | Advisory | MISRA C-2012 Rule 19.2 |
| Rule 20.1 | Rule | #include directives should only be preceded by preprocessor directives or comments. | Advisory | MISRA C-2012 Rule 20.1 |
| Rule 20.2 | Rule | The ', " or \ characters and the /\* or // character sequences shall not occur in a header file name. | Required | MISRA C-2012 Rule 20.2 |
| Rule 20.3 |  | The #include directive shall be followed by either a <filename> or "filename" sequence. | Required | MISRA C-2012 Rule 20.3 |
| Rule 20.4 | Rule | A macro shall not be defined with the same name as a keyword. | Required | MISRA C-2012 Rule 20.4 |
| Rule 20.5 | Rule | #undef should not be used. | Advisory | MISRA C-2012 Rule 20.5 |
| Rule 20.6 | Rule | Tokens that look like a preprocessing directive shall not occur within a macro argument. | Required | MISRA C-2012 Rule 20.6 |
| Rule 20.7 | Rule | Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses. | Required | MISRA C-2012 Rule 20.7 |
| Rule 20.8 | Rule | The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1. | Required | MISRA C-2012 Rule 20.8 |
| Rule 20.9 | Rule | All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation. | Required | MISRA C-2012 Rule 20.9 |
| Rule 20.10 | Rule | The # and ## preprocessor operators should not be used. | Advisory | MISRA C-2012 Rule 20.10 |
| Rule 20.11 | Rule | A macro parameter immediately following a # operator shall not immediately be followed by a ## operator. | Required | MISRA C-2012 Rule 20.11 |
| Rule 20.12 | Rule | A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators. | Required | MISRA C-2012 Rule 20.12 |
| Rule 20.13 | Rule | A line whose first token is # shall be a valid preprocessing directive. | Required | MISRA C-2012 Rule 20.13 |
| Rule 20.14 | Rule | All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related. | Required | MISRA C-2012 Rule 20.14 |
| Rule 21.1 | Rule | #define and #undef shall not be used on a reserved identifier or reserved macro name. | Required | MISRA C-2012 Rule 21.1 |
| Rule 21.2 | Rule | A reserved identifier or macro name shall not be declared. | Required | MISRA C-2012 Rule 21.2 |
| Rule 21.3 | Rule | The memory allocation and deallocation functions of <stdlib.h> shall not be used. | Required | MISRA C-2012 Rule 21.3 |
| Rule 21.4 | Rule | The standard header file <setjmp.h> shall not be used. | Required | MISRA C-2012 Rule 21.4 |
| Rule 21.5 | Rule | The standard header file <signal.h> shall not be used. | Required | MISRA C-2012 Rule 21.5 |
| Rule 21.6 | Rule | The Standard Library input/output functions shall not be used. | Required | MISRA C-2012 Rule 21.6 |
| Rule 21.7 | Rule | The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used. | Required | MISRA C-2012 Rule 21.7 |
| Rule 21.8 | Rule | The library functions abort, exit and system of <stdlib.h> shall not be used. | Required | MISRA C-2012 Rule 21.8 |
| Rule 21.9 | Rule | The library functions bsearch and qsort of <stdlib.h> shall not be used. | Required | MISRA C-2012 Rule 21.9 |
| Rule 21.10 | Rule | The Standard Library time and date functions shall not be used. | Required | MISRA C-2012 Rule 21.10 |
| Rule 21.11 | Rule | The standard header file <tgmath.h> shall not be used. | Required | MISRA C-2012 Rule 21.11 |
| Rule 21.12 | Rule | The exception handling features of <fenv.h> should not be used. | Advisory | MISRA C-2012 Rule 21.12 |
| Rule 21.13 | Rule | Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF. | Mandatory | MISRA C-2012 Rule 21.13 |
| Rule 21.14 | Rule | The Standard Library function memcmp shall not be used to compare null terminated strings. | Required | MISRA C-2012 Rule 21.14 |
| Rule 21.15 | Rule | The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types. | Required | MISRA C-2012 Rule 21.15 |
| Rule 21.16 | Rule | The pointer arguments to the Standard Library function memcmp shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type. | Required | MISRA C-2012 Rule 21.16 |
| Rule 21.17 | Rule | Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters. | Mandatory | MISRA C-2012 Rule 21.17 |
| Rule 21.18 | Rule | The size\_t argument passed to any function in <string.h> shall have an appropriate value. | Mandatory | MISRA C-2012 Rule 21.18 |
| Rule 21.19 | Rule | The pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they have pointer to const-qualified type. | Mandatory | MISRA C-2012 Rule 21.19 |
| Rule 21.20 | Rule | The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function. | Mandatory | MISRA C-2012 Rule 21.20 |
| Rule 22.1 | Rule | All resources obtained dynamically by means of Standard Library functions shall be explicitly released. | Required | MISRA C-2012 Rule 22.1 |
| Rule 22.2 | Rule | A block of memory shall only be freed if it was allocated by means of a Standard Library function. | Mandatory | MISRA C-2012 Rule 22.2 |
| Rule 22.3 | Rule | The same file shall not be open for read and write access at the same time on different streams. | Required | MISRA C-2012 Rule 22.3 |
| Rule 22.4 | Rule | There shall be no attempt to write to a stream which has been opened as read-only. | Mandatory | MISRA C-2012 Rule 22.4 |
| Rule 22.5 | Rule | A pointer to a FILE object shall not be dereferenced. | Mandatory | MISRA C-2012 Rule 22.5 |
| Rule 22.6 | Rule | The value of a pointer to a FILE shall not be used after the associated stream has been closed. | Mandatory | MISRA C-2012 Rule 22.6 |
| Rule 22.7 | Rule | The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF. | Required | MISRA C-2012 Rule 22.7 |
| Rule 22.8 | Rule | The value of errno shall be set to zero prior to a call to an errno-setting-function. | Required | MISRA C-2012 Rule 22.8 |
| Rule 22.9 | Rule | The value of errno shall be tested against zero after calling an errno-setting-function. | Required | MISRA C-2012 Rule 22.9 |
| Rule 22.10 | Rule | The value of errno shall only be tested when the last function to be called was an errno-setting-function. | Required | MISRA C-2012 Rule 22.10 |

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

**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