Code Review for C - MISRA 2004 rules

| **Code review reference** | **MISRA-C: 2004 reference** | **Code review message** | **Description** |
| --- | --- | --- | --- |
| Code compliance |  |  |  |
| M1.1 | Rule 1.1 | ANSI C error: <error> | All code shall conform to ISO 9899:1990  Required |
| M1.1w | Rule 1.1 | ANSI C warning: <warning> | Required |
| Language extensions |  |  | Required |
| M2.2 | Rule 2.2 | Source code shall only use /\* ... \*/ style comments | Source code shall only use /\* ... \*/ style comments  Required |
| M2.3 | Rule 2.3 | The character sequence /\* shall not be used within a comment | The character sequence /\* shall not be used within a comment  Required |
| E2.3.1 |  |  | The character sequence // should not be used within a 'C-style' comment  Advisory |
| E2.3.2 |  |  | Line-splicing shall not be used in // comments  Advisory |
| E2.6 |  |  | A function should not contain unused label declarations  Advisory |
| E2.7 |  | There should be no unused parameters in functions | Advisory |
| E2.8 |  | Macro <name> is never used | Advisory |
| E2.9 |  | Type <name> is never used | Advisory |
| E2.10 |  | Tag <name> is never used | Advisory |
| E2.50 |  | Functions should have less than '100' lines. Note The number of lines can be specified. | Advisory |
| E2.51 |  | Functions should have less than '15' V(g) complexity. Note: The complexity limit of lines can be specified. | Advisory |
| Documentation |  |  |  |
| M3.4 | Rule 3.4 | All uses of the #pragma directive shall be documented and explained | Required |
| Character sets |  |  |  |
| M4.1.1 | Rule 4.1 | Only escape sequences that are defined in the ISO C standard shall be used | Only escape sequences that are defined in the ISO C standard shall be used  Required |
| M4.1.2 | Rule 4.1 | Only ISO C escape sequences are allowed(\v) | Only ISO C escape sequences are allowed(\v)  Required |
| M4.2 | Rule 4.2 | Trigraphs shall not be used | Trigraphs shall not be used  Required |
| Identifiers |  |  |  |
| M5.1 | Rule 5.1 | Identifiers <name> and <name> are identical in the first <value> characters. Note The number of characters can be specified. | Identifiers (internal and external) shall not rely on the significance of more than 31 characters  Required |
| E5.1.1 |  | Identifiers <name> and <name> are ambiguous because of possible character confusion. Note that ambiguous characters can be parameterized. | Advisory |
| E5.1.2 |  | Possible typing mistakes between the variables <name> or <name> because of repeating character. | Advisory |
| E5.1.3 |  | Identifiers <name> and <name> are identical in the first <param> characters ignoring case | Advisory |
| E5.1.4 |  | Macros <name> and <name> are identical in the first <param> characters | Advisory |
| E5.1.5 |  | Macro <name> and identifier <name> are identical in the first <param> characters | Advisory |
| E5.1.6 |  | Macros <name> and <name> are identical in the first <param> characters ignoring case | Advisory |
| E5.1.7 |  | Macro <name> and identifier <name> are identical in the first <param> characters ignoring case | Advisory |
| M5.2 | Rule 5.2 | Identifier <name> in an inner scope hides the same identifier in an outer scope : <location> | Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier  Required |
| E5.3 |  | The tag name <name> should not be reused. Name already found in <location> | Advisory |
| M5.3.1 | Rule 5.3 |  | The typedef name '%name%' should not be reused except for its tag. Name already found in %location%  Required |
| M5.3.2 | Rule 5.3 |  | The typedef name '%name%' should not be reused even for its tag. Name already found in %location%  Required |
| M5.4 | Rule 5.4 | A struct and union cannot use the same tag name | A tag name shall be a unique identifier  Required |
| M5.5 | Rule 5.5 | The static object or function <name> should not be reused. Static object or function already found in <location>. | No object or function identifier with static storage duration should be reused  Advisory |
| M5.6 | Rule 5.6 | Avoid using the same identifier <name> in two different name spaces. Identifier already found in <location> | No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure and union member names  Advisory |
| M5.7 | Rule 5.7 | The identifier <name> should not be reused. Identifier already found in <location>. | Advisory |
| Types |  |  |  |
| M6.1.1 | Rule 6.1 | The C language plain char type should only be used for character values. | The C language plain char type should only be used for character values.  Required |
| M6.1.2 | Rule 6.1 | Case char value is applicable only if the switch statement value is plain character variable | Required |
| M6.1.3 | Rule 6.1 | Avoid using comparison operators on plain char. | Required |
| M6.2 | Rule 6.2 | The C language signed char or unsigned char types should only be used for numeric values. | The C language signed char or unsigned char types should only be used for numeric values.  Required |
| M6.3 | Rule 6.3 | The C language numeric type <name> should not be used directly but instead used to define typedef. | typedefs that indicate size and signedness should be used in place of the basic types  Advisory |
| E6.3 |  | The implicit 'int' type should not be used. | Required |
| M6.4.1 | Rule 6.4 | Bit fields should only be of type 'unsigned int' or 'signed int'. | Required |
| M6.4.2 | Rule 6.4 | Bit fields should not be of type 'enum' | Required |
| M6.4.3 | Rule 6.4 | Bit fields should only be of explicitly signed or unsigned type | Required |
| M6.4.4 | Rule 6.4 | Bit fields should not be of type 'bool' under c99 | Required |
| M6.4.5 | Rule 6.4 | Bit fields should not be of type 'boolean' outside c99 | Required |
| M6.5 | Rule 6.5 | Bit fields of type 'signed int' must be at least 2 bits long. | Required |
| Constants |  |  |  |
| M7.1 | Rule 7.1 | Octal constants and escape sequences should not be used. | Octal constants (other than zero) and octal escape sequences shall not be used  Required |
| E7.1 |  | Octal and hexadecimal escape sequences shall be terminated | Required |
| E7.2 |  | The lowercase character 'l' shall not be used in a literal suffix | Required |
| E7.3 |  | A string literal shall not be assigned to an object unless the object's type is pointer to a const-qualified char | Required |
| Declarations and definitions |  |  |  |
| M8.1.1 | Rule 8.1 | A prototype for the function <name> should be declared before defining the function. | Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call  Required |
| E8.1.1 |  | A prototype for the global object <name> should be declared before defining the object | Required |
| M8.1.2 | Rule 8.1 | A prototype for the function <name> should be declared before calling the function. | Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call  Required |
| M8.1.3 | Rule 8.1 | A prototype for the function <name> should be declared before calling the function | Required |
| M8.2.1 | Rule 8.2 | The type of <name> should be explicitly stated. | Whenever an object or function is declared or defined, its type shall be explicitly stated  Required |
| M8.2.2 | Rule 8.2 | The type of parameter <name> should be explicitly stated | Required |
| M8.3 | Rule 8.3 | Parameters and return types should use the same type names in the declaration and in the definition, even if basic types are the same. | For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical  Required |
| E8.3 |  | Parameters and return types should use compatible type in the declaration and in the definition | Required |
| M8.4 | Rule 8.4 | If objects or functions are declared multiple times their types should be compatible. | Required |
| M8.5.1 | Rule 8.5 | The body of function <name> should not be located in a header file. | Required |
| E.8.50 |  | Use the const qualification for variable <name> which is pointer and which is not used to change the pointed object | Required |
| **E.8.51** |  | The object <name> is never referenced | Required |
| M8.5.2 | Rule 8.5 | The memory storage (definition) for the variable <name> should not be in a header file. | Objects shall be defined at block scope if they are only accessed from within a single function.  Required |
| M8.6 | Rule 8.6 | Functions should not be declared at block scope. | Required |
| **M8.7** | Rule 8.7 | Global objects should not be declared if they are only used from within a single function. | Objects shall be defined at block scope if they are only accessed from within a single function  Required |
| M8.8.2 | Rule 8.8 | Static function <name> should only be declared in a single file. Redundant declaration found at: <location> | Required |
| M8.8.3 | Rule 8.8 | Static object <name> should only be declared in a single file. Redundant declaration found at: <location> | Required |
| M8.8.4 | Rule 8.8 | Identifiers <name>that declare objects or functions with external linkage shall be declared once in one and only one file | Required |
| M8.8.5 | Rule 8.8 | Identifiers <name> that declare objects or functions with external linkage shall be unique | Required |
| M8.9.1 | Rule 8.9 | The global object or function <name> should have exactly one external definition. Redundant definition found in <location>. | An identifier with external linkage shall have exactly one external definition |
| **M8.9.2** | Rule 8.9 | The global object or function <name> should have exactly one external definition. No definition found. | Required |
| **M8.10.1** | Rule 8.10 | Global object <name> that are only used within the same file should be declared using the static storage-class specifier. | All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.  Required |
| **M8.10.2** | Rule 8.10 | Global function <name> that are only used within the same file should be declared using the static storage-class specifier. | All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required  Required |
| M8.11 | Rule 8.11 | Global objects or functions that are only used within the same file should be declared with using the static storage-class specifier. | The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage  Required |
| M8.12 | Rule 8.12 | When a global array variable can be used from multiple files, its size should be defined at initialization time. | Required |
| E.8.14 |  | Inline function <name> should be static | Required |
| Initialization |  | The restrict type qualifier shall not be used | Required |
| M9.1 | Rule 9.1 | Variables with automatic storage duration should be initialized before being used. | Required |
| M9.2 | Rule 9.2 | Nested braces should be used to initialize nested multi-dimension arrays and nested structures. | Required |
| E9.2 |  | Arrays shall not be partially initialized | Required |
| M9.3 | Rule 9.3 | Either all members or only the first member of an enumerator list should be initialized. | In an enumerator list, the “=” construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized  Required |
| M9.3 | Rule 9.3 | Either all members or only the first member of an enumerator list should be initialized | Required |
| E9.3 | Rule E9.3 | Enumeration member <name> have a not unique implicitly specified value | Required |
| E9.4 |  | The global variable <name> is not initialized | Required |
| Arithmetic type conversions |  |  |  |
| E10.1 |  | Constraint violation : can't use floating type as operand of '[], %, <<, >>, ~, &, |, ^' | Required |
| M10.1.1 | Rule 10.1 | Implicit conversion of a complex integer expression to a smaller sized integer is not allowed. | The value of an expression of integer type shall not be implicitly converted to a different underlying type if:   * a)  it is not a conversion to a wider integer type of the same signedness, or * b)  the expression is complex, or * c)  the expression is not constant and is a function argument, or * d)  the expression is not constant and is a return expression.   Required |
| M10.1.2 | Rule 10.1 | Implicit conversion of an integer expression to a different signedness is not allowed. | Required |
| M10.2 | Rule 10.2 | Conversion of a complex floating expression is not allowed. Only constant expressions can be implicitly converted and only to a wider floating type of the same signedness. | The value of an expression of floating type shall not be implicitly converted to a different type if:   * a) it is not a conversion to a wider floating type, or * b) the expression is complex, or * c) the expression is a function argument, or * d) the expression is a return expression.   Required |
| E10.2 |  | Operand should be boolean. | Required |
| M10.3 | Rule 10.3 | Type cast of complex integer expressions is only allowed into a narrower type of the same signedness. | The value of a complex expression of integer type may only be cast to a type that is narrower and of the same signedness as the underlying type of the expression  Required |
| E10.3 |  | Can't use a boolean as a numeric value | Required |
| M10.4 | Rule 10.4 | Type cast of complex floating expressions is only allowed into a narrower type of the same signedness. | The value of a complex expression of floating type may only be cast to a narrower floating type  Required |
| E10.4 |  | Can't use a char as a numeric value | Required |
| M10.5 | Rule 10.5 | When using operator '~' or '<<' on 'unsigned char' or 'unsigned int', you should always cast returned value | Required |
| E10.5 | Rule E10.5 | Can't use a not **anonymous enum** as a numeric value | Required |
| M10.6 | Rule 10.6 | Definitions of unsigned type constants should use the 'U' suffix. | A “U” suffix shall be applied to all constants of unsigned type  Required |
| E10.6 |  | Shift and bitwise operations should be performed on unsigned value | Required |
| E10.7 |  | Right hand operand of shift operation should be an unsigned value | Required |
| E10.8 |  | Unary minus operation should not be performed on unsigned value | Required |
| E10.9 |  | Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations | Required |
| E10.10 |  | The value of an expression shall not be assigned to an object with a narrower essential type | Required |
| E10.11 |  | The value of an expression shall not be assigned to an object with a different essential type category | Required |
| E10.12 |  | Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category | Required |
| E10.13 |  | The value of an expression should not be cast to an inappropriate essential type | Required |
| E10.14 |  | The value of a composite expression shall not be assigned to an object with wider essential type | Required |
| E10.15 |  | If a composite expression is used as one operand of an operation in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type | Required |
| E10.16 |  | The value of a composite expression shall not be cast to a different essential type category or a wider essential type | Required |
| Pointer type conversions |  |  |  |
| M11.1 | Rule 11.1 | A function pointer should not be converted to another type of pointer. | Conversions shall not be performed between a pointer to a function and any type other than an integral type  Required |
| E11.1 |  | Conversions shall not be performed between a pointer to an incomplete type and any other type | Required |
| M11.2 | Rule 11.2 | An object pointer should not be converted to another type of pointer. | Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void  Required |
| E11.2 |  | A conversion should not be performed from pointer to void into pointer to object | Required |
| M11.3 | Rule 11.3 | Casting a pointer type to an integer type should not occur. | A cast should not be performed between a pointer type and an integral type  Advisory |
| E11.3 | E11.3 | A cast shall not be performed between pointer **to void**and an arithmetic type | Required |
| E11.4 |  | A cast shall not be performed between pointer to object and a non-integer arithmetic type | Required |
| M11.4.1 | Rule 11.4 | Casting an object pointer type to a different object pointer type should not occur. | A cast should not be performed between a pointer to object type and a different pointer to object type  Advisory |
| M11.4.2 | Rule 11.4 | Casting an object pointer type to a different object pointer type should not occur, especially when object sizes are not the same. | Advisory |
| M11.5 | Rule 11.5 | Casting of pointers to a type that removes any const or volatile qualification on the pointed object should not occur. | A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.  Required |
| Expressions |  |  |  |
| M12.1 | Rule 12.1 | Implicit operator precedence may cause ambiguity. Use parenthesis to clarify this expression. | Limited dependence should be placed on C’s operator precedence rules in expressions  Advisory |
| E12.11 |  | Implicit bitwise operator precedence may cause ambiguity. Use parenthesis to clarify this expression. | Advisory |
| M12.3 | Rule 12.3 | The sizeof operator should not be used on expressions that contain side effects. | Required |
| M12.4.1 | Rule 12.4 | An expression that contains a side effect should not be used in the right-hand operand of a logical && or || operator. | The right-hand operand of a logical && or || operator shall not contain side effects  Required |
| M12.4.2 | Rule 12.4 | The function in the right-hand operand of a logical && or || operator might cause side effects. |
| M12.5 | Rule 12.5 | Parenthesis should be used around expressions that are operands of a logical && or ||. | Required |
| E12.51 |  | Ternary expression ?: should not be used. | Advisory |
| E12.54 |  | Expressions should not cause a side effect assignment. | Advisory |
| M12.6 | Rule 12.6 | Only Boolean operands should be used with logical operators ( &&, || and !). | The operands of logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, || and !)  Advisory |
| E12.61 |  | The operator on a Boolean expression should be a logical operator ( &amp;&amp;, || or !). | Advisory |
| M12.7 | Rule 12.7 | Bitwise operators should only use unsigned operands. | Bitwise operators shall not be applied to operands whose underlying type is signed  Required |
| M12.8 | Rule 12.8 | The right-hand operand of a shift operator should not be too big or negative. | The right-hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left-hand operand  Required |
| M12.9 | Rule 12.9 | Only use unary minus operators with signed expressions. | The unary minus operator shall not be applied to an expression whose underlying type is unsigned  Required |
| M12.10 | Rule 12.10 | Do not use the comma operator | Required |
| M12.13 | Rule 12.13 | The increment (++) or the decrement (--) operators should not be used with other operators in an expression. | Advisory |
| Control statement expressions |  |  |  |
| E13.1 |  | The result of an assignment operator should not be used in an expression | Required |
| M13.1.1 | Rule 13.1 | Boolean expressions should not contain assignment operators. | Assignment operators shall not be used in expressions that yield a Boolean value  Required |
| M13.1.2 | Rule 13.1 | Boolean expressions should not contain side effect operators. | Required |
| M 13.2 | Rule 13.2 | Non-Boolean values that are tested against zero should have an explicit test | Tests of a value against zero should be made explicit, unless the operand is effectively Boolean  Advisory |
| M13.3 | Rule 13.3 | The equal or not equal operator should not be used in floating-point expressions. | Floating-point expressions shall not be tested for equality or inequality  Required |
| M13.4 | Rule 13.4 | Floating-point variables should not be used to control a for statement. | Required |
| M13.5.1 | Rule 13.5 | Only loop counter should be initialized in a loop initialization part. | The three expressions of a statement shall be concerned with loop control only.  Required |
| M13.5.2 | Rule 13.5 | In the 'update part' of a 'for statement', only 'loop counter' should be updated | Required |
| M13.5.3 | Rule 13.5 | There should be one and only one loop counter for loop statement. | Required |
| M13.6 | Rule 13.6 | Loop counter of a 'for statement' should not be modified within the body of the loop. | Required |
| M13.7 | Rule 13.7 | Invariant Boolean expressions should not be used. | Boolean operations whose results are invariant shall not be permitted  Required |
| Control flow |  |  |  |
| M14.1 | Rule 14.1 | Unreachable code. | Required |
| M14.2 | Rule 14.2 | A non-null statement should either have a side effect or change the control flow. | Required |
| M14.3 | Rule 14.3 | A null statement in original source code should be on a separate line and the semicolon should be followed by at least one white space and then a comment. | Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character  Required |
| M14.4 | Rule 14.4 | Do not use the goto statement. | Required |
| E14.4.1 |  | The goto statement shall jump to a label declared later in the same function | Required |
| E14.4.2 |  | Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement | Required |
| E14.4.3 |  | There should be no more than one break or goto statement used to terminate any iteration statement | Required |
| M14.5 | Rule 14.5 | Do not use the continue statement. | Required |
| M14.6 | Rule 14.6 | Only one break statement should be used within a loop. | For any iteration statement there shall be at most one break statement used for loop termination  Required |
| M14.7.1 | Rule 14.7 | Only one exit point should be defined in a function. | A function shall have a single point of exit at the end of the function  Required |
| M14.7.2 | Rule 14.7 | The return keyword should not be used in a conditional block. | Required |
| M14.8.1 | Rule 14.8 | The switch statement should be followed by a compound statement {}. | The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement  Required |
| M14.8.2 | Rule 14.8 | The while statement should be followed by a compound statement {}. |
| M14.8.3 | Rule 14.8 | The do..while statement should contain a compound statement {}. |
| M14.8.4 | Rule 14.8 | The for statement should be followed by a compound statement {}. |
| M14.9.1 | Rule 14.9 | The if (expression) construct should be followed by a compound statement {}. | An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement  Required |
| M14.9.2 | Rule 14.9 | The else keyword should be followed by either a compound statement or another if statement. |
| M14.9.3 | Rule 14.9 | The else keyword should be followed by a compound statement |
| M14.10 | Rule 14.10 | All if ... else if sequences should have an else block. | All if ... else if constructs shall be terminated with an else clause  Required |
| Switch statements |  |  |  |
| M15.0 | Rule 15.0 | A switch block should start with a case. | The MISRA C switch syntax shall be used  Required |
| M15.1 | Rule 15.1 | A case or default statements should only be used directly within the compound block of a switch statement. | A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement  Required |
| E15.10 |  | The switch expression should not have side effects. | Required |
| M15.2 | Rule 15.2 | The break statement should only be used to terminate every non-empty switch block. | An unconditional break statement shall terminate every non-empty switch clause  Required |
| M15.3.1 | Rule 15.3 | The switch statement should have a default clause. | Required |
| M15.3.2 | Rule 15.3 | The default clause should be the last clause of the switch statement. |  |
| M15.4.1 | Rule 15.4 | A Boolean should not be used as a switch expression. | A switch expression shall not represent a value that is effectively Boolean  Required |
| M15.4.2 | Rule 15.4 | A constant should not be used as a switch expression. | Required |
| M15.5 | Rule 15.5 | At least one case should be defined in the switch. | Every switch statement shall have at least one case clause  Required |
| Functions |  |  |  |
| M16.1 | Rule 16.1 | The function <name> should not have a variable number of arguments. | Functions shall not be defined with a variable number of arguments  Required |
| Rule M16.1.2 | Rule 16.1 | The library functions 'va\_list, va\_arg, va\_start, va\_end, va\_copy' should not be used | Required |
| M16.2.1 | Rule 16.2 | Recursive functions are not allowed. The function <name> is directly recursive. | Functions shall not call themselves, either directly or indirectly Functions shall not call themselves, either directly or indirectly  Required |
| M16.2.2 | Rule 16.2 | Recursive functions are not allowed. The function <name> is recursive when calling <name>. |
| M16.3 | Rule 16.3 | The function prototype should name all its parameters. | Identifiers shall be given for all of the parameters in a function prototype declaration  Required |
| M16.4 | Rule 16.4 | The identifiers used in the prototype and definition should be the same. | Required |
| M16.5 | Rule 16.5 | Functions with no parameters should use the void type. | Required |
| **E16.50** |  | The function <name> is never referenced. | Required |
| M16.6 | Rule 16.6 | The number of arguments used in the call does not match the number declared in the prototype. | Required |
| M16.7 | Rule 16.7 | Use the const qualification for parameter <name> which is pointer and which is not used to change the pointed object. | A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object  Required |
| M16.8 | Rule 16.8 | The return should always be defined with an expression for non-void functions. | All exit paths from a function with non-void return type shall have an explicit return statement with an expression  Required |
| M16.9 | Rule 16.9 | Function identifiers should always use a parenthesis or a preceding &. | A function identifier shall only be used with either a preceding &, or with a parenthesized parameter list, which may be empty  Required |
| M16.10 | Rule 16.10 | When a function returns a value, this value should be used. | If a function returns error information, then that error information shall be tested  Required |
| Pointers and arrays |  |  |  |
| M17.4 | Rule 17.4 | Pointer arithmetic except array indexing should not be used. | Array indexing shall be the only allowed form of pointer arithmetic  Required |
| M17.5 | Rule 17.5 | A declaration should not use more than two levels of pointer indirection. | Advisory |
| Structures and unions |  |  |  |
| M18.1 | Rule 18.1 | Structure or union types should be finalized before the end of the compilation units. | Required |
| E18.1 |  | Flexible arrays members shall not be declared | Required |
| 18.2 |  | Variable-length array types shall not be used | Required |
| E18.3 |  | The declaration of an array parameter shall not contain the static keyword between the [] | Required |
| M18.4 | Rule 18.4 | Do not use unions. | Required |
| Preprocessing directives |  |  |  |
| M19.1 | Rule 19.1 | Only preprocessor directives or comments may occur before the #include statements. | #include statements in a file should only be preceded by other preprocessor directives or comments  Advisory |
| M19.2 | Rule 19.2 | Do not use non-standard characters in included file names. | Advisory |
| M19.3 | Rule 19.3 | Filenames with the #include directive should always use the <filename> or "filename" syntax. | Required |
| M19.4 | Rule 19.4 | A C macro should only be expanded to a constant, a braced initializer, a parenthesised expression, a storage class keyword, a type qualifier, or a do-while-zero block. | Required |
| M19.5 | Rule 19.5 | Macro definitions or #undef should not be located within a block. | Required |
| M19.6 | Rule 19.6 | Do not use the #undef directive. | Required |
| M19.7 | Rule 19.7 | Function should be used instead of macros when possible. | Advisory |
| M19.8 | Rule 19.8 | Missing argument when calling the macro. | A function-like macro shall not be invoked without all of its arguments.  Required |
| M19.9 | Rule 19.9 | The preprocessing directive <name> should not be used as argument to the macro. | Arguments to a function-like macro shall not contain tokens that look like preprocessing directives  Required |
| M19.10 | Rule 19.10 | The parameter <name> in the macro should be enclosed in parentheses except when it is used as the operand of # or ##. | In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##  Required |
| M19.11 | Rule 19.11 | Undefined macro identifier in the preprocessor directive. | All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator  Required |
| M19.12 | Rule 19.12 | The # or ## preprocessor operator should not be used more than once. | There shall be at most one occurrence of the # or # preprocessor operators in a single macro definition  Required |
| M19.13 | Rule 19.13 | The # and ## preprocessor operator should be avoided. | Advisory |
| M19.14 | Rule 19.14 | Only use the 'defined' preprocessor operator with a single identifier. | The defined preprocessor operator shall only be used in one of the two standard forms  Required |
| M19.15 | Rule 19.15 | Header file contents should be protected against multiple inclusions | Precautions shall be taken in order to prevent the contents of a header file being included twice  Required |
| M19.16 | Rule 19.16 | Possible bad syntax in preprocessing directive. | Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor  Required |
| M19.17 | Rule 19.17 | A #if, #ifdef, #else, #elif or #endif preprocessor directive has been found without its matching directive in the same file. | All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related  Required |
| E19.18 |  | The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1 | Required |
| E19.19 |  | A macro parameter immediately following a # operator shall not immediately be followed by a ## operator | Required |
| E19.20 |  | Macro parameter <name> used as an operand to the # and ## operators shall not be used elsewhere in this macro | Required |
| Standard libraries |  |  |  |
| M20.1 | Rule 20.1 | <name> should not be defined, redefined or undefined. | Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined  Required |
| E20.1 |  | A macro shall not be defined with the same name as a keyword | Required |
| M20.2.1 |  | #define and #undef shall not be used on a reserved identifier or reserved macro name: Identifier <name> already found in <name> | Required |
| M20.2.2 | Rule 20.2 | #define and #undef shall not be used on identifier beginning with an underscore or on 'defined' keyword" | Required |
| M20.2.3 | Rule 20.2 | Declared identifier should not be a reserved identifier or reserved macro name: Identifier <name> already found in <name> | Required |
| M20.2.4 | Rule 20.2 | Declared identifier should not begin with an underscore or be 'defined' keyword | Required |
| M20.4 | Rule 20.4 | Dynamic heap memory allocation shall not be used. | This precludes the use of the functions calloc, malloc, realloc free and strdup. There is a whole range of unspecified, undefined and implementation-defined behaviour associated with dynamic memory allocation, as well as a number of other potential pitfalls. Dynamic heap memory allocation may lead to memory leaks, data inconsistency, memory exhaustion, non-deterministic.  Note that some implementations may use dynamic heap memory allocation to implement other functions (for example functions in the library string.h). If this is the case then these functions shall also be avoided.  Required |
| M20.5 | Rule 20.5 | The error indicator errno shall not be used. | errno is a facility of C, which in theory should be useful, but which in practice is poorly defined by the standard. A non zero value may or may not indicate that a problem has occurred; as a result it shall not be used. Even for those functions for which the behaviour of errno is well defined, it is preferable to check the values of inputs before calling the function rather than rely on using errno to trap errors (see Rule 16.10).  Required |
| M20.6 | Rule 20.6 | The macro offsetof, in library <stddef.h>, shall not be used. | Use of this macro can lead to undefined behaviour when the types of the operands are incompatible or when bit fields are used.  Required |
| M20.7 | Rule 20.7 | The setjmp macro and the longjmp function shall not be used. | etjmp and longjmp allow the normal function call mechanisms to be bypassed, and shall not be used.  Remark : sigsetjmp and siglongjmp (Gnu Library) are also detected  Required |
| E20.7 |  | The standard header file <setjmp.h> shall not be used | Required |
| M20.8 | Rule 20.8 | The signal handling facilities of <signal.h> shall not be used. | Signal handling contains implementation-defined and undefined behavior.  Required |
| M20.9 | Rule 20.9 | The input/output library <stdio.h> shall not be used in production code. | This includes file and I/O functions fgetpos, fopen, ftell, gets, perror, remove, rename, and ungetc.  Streams and file I/O have a large number of unspecified, undefined and implementation-defined behaviours associated with them. It is assumed within this document that they will not normally be needed in production code in embedded systems.  If any of the features of stdio.h need to be used in production code, then the issues associated with the feature need to be understood.  Required |
| M20.10 | Rule 20.10 | The library functions atof, atoi and atol from library <stdlib.h> shall not be used. | These functions have undefined behavior associated with them when the string cannot be converted.  Required |
| M20.11 | Rule 20.11 | The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used. | These functions will not normally be required in an embedded system, which does not normally need to communicate with an environment  Then, it is essential to check on the implementation-defined behavior of the function in the environment.  Required |
| E20.11 |  | The library macro or function 'bsearch, qsort' should not be used | Required |
| M20.12 | Rule 20.12 | The time handling functions of library <time.h> shall not be used. | Includes time, strftime. This library is associated with clock times. Various aspects are implementation dependent or unspecified, such as the format of times. If any of the facilities of time.h are used, then the exact implementation for the compiler being used must be determined, and a deviation being raised.  Required |
| E20.12 |  | The input/output library <wchar.h> shall not be used in production code | Required |
| E20.13 |  | The standard header file <tgmath.h> shall not be used | Required |
| E20.14 |  | The library macro or function 'feclearexcept, fegetexceptflag, feraiseexcept, fesetexceptflag, fetestexcept, FE\_INEXACT, FE\_DIVBYZERO, FE\_UNDERFLOW, FE\_OVERFLOW, FE\_INVALID, FE\_ALL\_EXCEPT' should not be used | Required |
| U99.1 |  | User defined rule | Required |
| Table 1. MISRA 2004 rules | | | |