MISRA C:2004 Rule Coverage

	Supported	All	% Coverage
All	130	142	91.5
Required	112	124	90.3
Advisory	18	18	100.0

MISRA C:2004 Supported Rules

Rule	Rule Name	Category
Rule 1.1	All code shall conform to ISO/IEC 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.	Required
Rule 2.1	Assembly language shall be encapsulated and isolated.	Required
Rule 2.2	Source code shall only use /* ... */ style comments.	Required
Rule 2.3	The character sequence /* shall not be used within a comment.	Required
Rule 2.4	Sections of code should not be "commented out".	Advisory
Rule 4.1	Only those escape sequences that are defined in the ISO C standard shall be used.	Required
Rule 4.2	Trigraphs shall not be used.	Required
Rule 5.1	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.	Required
Rule 5.2	Identifiers in an inner scope shall not use the same name as an identfier in an outer scope, and therefore hide that identfier.	Required
Rule 5.3	A typedef name shall be a unique identifier.	Required
Rule 5.4	A tag name shall be a unique identifier.	Required
Rule 5.5	No object or function identifier with static storage duration should be reused.	Advisory
Rule 5.6	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.	Advisory
Rule 5.7	No identifier name should be reused.	Advisory
Rule 6.1	The plain char type shall be used only for the storage and use of character values.	Required
Rule 6.2	Signed and unsigned char type shall be used only for the storage and use of numeric values.	Required
Rule 6.3	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	Advisory
Rule 6.4	Bit fields shall only be defined to be of type unsigned int or signed int.	Required
Rule 6.5	Bit fields of signed type shall be at least 2 bits long.	Required
Rule 7.1	Octal constants (other than zero) and octal escape sequences shall not be used.	Required

Rule 8.1	Functions shall have prototype declarations and prototype shall be visible at both the function definition and call.	Required
Rule 8.2	Whenever an object or function is declared or defined, its type shall be explicitly stated.	Required
Rule 8.3	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.	Required
Rule 8.4	If objects or functions are declared more than once their types shall be compatible.	Required
Rule 8.5	There shall be no definitions of objects or functions in a header file.	Required
Rule 8.6	Functions shall be declared at file scope.	Required
Rule 8.7	Objects shall be defined at block scope if they are only accessed from within a single function.	Required
Rule 8.8	An external object or function shall be declared in one and only one file.	Required
Rule 8.9	An identifier with external linkage shall have exactly one external definition.	Required
Rule 8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.	Required
Rule 8.11	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.	Required
Rule 8.12	When an array is declared with external linkage, its size shall be stated explicitly or defned implicitly by initialisation.	Required
Rule 9.1	All automatic variables shall have been assigned a value before being used.	Required
Rule 9.2	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.	Required
Rule 9.3	In an enumerator list, the "=" construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.	Required
Rule 10.1	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
Rule 10.2	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
Rule 10.3	The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.	Required
Rule 10.4	The value of a complex expression of floating type shall only be cast to a floating type that is narrower or of the same size.	Required
Rule 10.5	If the bitwise operators ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	Required
Rule 10.6	A "U" suffix shall be applied to all constants of unsigned type.	Required
Rule 11.1	Conversions shall not be performed between a pointer to a function and any type other than an integral type.	Required
Rule 11.2	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
Rule 11.3	A cast should not be performed between a pointer type and an integral type.	Advisory
Rule 11.4	A cast should not be performed between a pointer to object type and a different pointer to object type.	Advisory
Rule 11.5	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.	Required
Rule 12.1	Limited dependence should be placed on C's operator precedence rules in expressions.	Advisory
Rule 12.2	The value of an expression shall be the same under any order of evaluation that the standard permits.	Required
Rule 12.3	The sizeof operator shall not be used on expressions that contain side effects.	Required
Rule 12.4	The right hand operand of a logical && or || operator shall not contain side effects.	Required
Rule 12.5	The operands of a logical && or || shall be primary expressions.	Required
Rule 12.6	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

Rule 12.7	Bitwise operators shall not be applied to operands whose underlying type is signed.	Required
Rule 12.8	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
Rule 12.9	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	Required
Rule 12.10	The comma operator shall not be used.	Required
Rule 12.11	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	Advisory
Rule 12.12	The underlying bit representations of floating-point values shall not be used.	Required
Rule 12.13	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	Advisory
Rule 13.1	Assignment operators shall not be used in expressions that yield a Boolean value.	Required
Rule 13.2	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	Advisory
Rule 13.3	Floating-point expressions shall not be tested for equality or inequality.	Required
Rule 13.4	The controlling expression of a for statement shall not contain any objects of floating type.	Required
Rule 13.5	The three expressions of a for statement shall be concerned only with loop control.	Required
Rule 13.6	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	Required
Rule 13.7	Boolean operations whose results are invariant shall not be permitted.	Required
Rule 14.1	There shall be no unreachable code.	Required
Rule 14.2	All non-null statements shall either (a) have at least one side-effect however executed, or (b) cause control flow to change.	Required
Rule 14.3	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
Rule 14.4	The goto statement shall not be used.	Required
Rule 14.5	The continue statement shall not be used.	Required
Rule 14.6	For any iteration statement there shall be at most one break statement used for loop termination.	Required
Rule 14.7	A function shall have a single point of exit at the end of the function.	Required
Rule 14.8	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Required
Rule 14.9	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
Rule 14.10	All if ... else if constructs shall be terminated with an else statement.	Required
Rule 15.0	The MISRA C switch syntax shall be used.	Required
Rule 15.1	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	Required
Rule 15.2	An unconditional break statement shall terminate every non-empty switch clause.	Required
Rule 15.3	The final clause of a switch statement shall be the default clause.	Required
Rule 15.4	A switch expression shall not represent a value that is effectively Boolean.	Required
Rule 15.5	Every switch statement shall have at least one case clause.	Required
Rule 16.1	Functions shall not be defined with variable numbers of arguments.	Required
Rule 16.2	Functions shall not call themselves, either directly or indirectly.	Required
Rule 16.3	Identifiers shall be given for all of the parameters in a function prototype declaration.	Required
Rule 16.4	The identifiers used in the declaration and definition of a function shall be identical.	Required
Rule 16.5	Functions with no parameters shall be declared and defined with the parameter list void.	Required
Rule 16.6	The number of arguments passed to a function shall match the number of parameters.	Required
Rule 16.7	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.	Advisory
Rule 16.8	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	Required

Rule 16.9	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.	Required
Rule 16.10	If a function returns error information, then that error information shall be tested.	Required
Rule 17.1	Pointer arithmetic shall only be applied to pointers that address an array or array element.	Required
Rule 17.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Required
Rule 17.3	The relational operators >, >=, <, <= shall not be applied to pointer types except where they point to the same array.	Required
Rule 17.4	Array indexing shall be the only allowed form of pointer arithmetic.	Required
Rule 17.5	The declaration of objects should contain no more than 2 levels of pointer indirection.	Advisory
Rule 17.6	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Required
Rule 18.1	All structure or union types shall be complete at the end of a translation unit.	Required
Rule 18.2	An object shall not be assigned to an overlapping object.	Required
Rule 18.4	Unions shall not be used.	Required
Rule 19.1	#include statements in a file should only be preceded by other preprocessor directives or comments.	Advisory
Rule 19.2	Non-standard characters should not occur in header file names in #include directives.	Advisory
Rule 19.3	The #include directive shall be followed by either a or "filename" sequence.	Required
Rule 19.4	C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.	Required
Rule 19.5	Macros shall not be #defined or #undefd within a block.	Required
Rule 19.6	#undef shall not be used.	Required
Rule 19.7	A function should be used in preference to a function-like macro.	Advisory
Rule 19.8	A function-like macro shall not be invoked without all of its arguments.	Required
Rule 19.9	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	Required
Rule 19.10	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
Rule 19.11	All macro Identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.	Required
Rule 19.12	There shall be at most one occurrence of the # or ## operators in a single macro definition.	Required
Rule 19.13	The # and ## preprocessor operators should not be used.	Advisory
Rule 19.14	The defined preprocessor operator shall only be used in one of the two standard forms.	Required
Rule 19.15	Precautions shall be taken in order to prevent the contents of a header file being included twice.	Required
Rule 19.16	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.	Required
Rule 19.17	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
Rule 20.1	Reserved identifers, macros and functions in the standard library, shall not be defined, redefined or undefined.	Required
Rule 20.2	The names of standard library macros, objects and functions shall not be reused.	Required
Rule 20.3	The validity of values passed to library functions shall be checked.	Required
Rule 20.4	Dynamic heap memory allocation shall not be used.	Required
Rule 20.5	The error indicator errno shall not be used.	Required
Rule 20.6	The macro offsetof, in library , shall not be used.	Required
Rule 20.7	The setjmp macro and the longjmp function shall not be used.	Required
Rule 20.8	The signal handling facilities of shall not be used.	Required
Rule 20.9	The input/output library shall not be used in production code.	Required
Rule 20.10	The library functions atof, atoi and atol from library shall not be used.	Required

Rule 20.11	The library functions abort, exit, getenv and system from library shall not be used.	Required
Rule 20.12	The time handling functions of library shall not be used.	Required

MISRA C++:2008 Rule Coverage

	Supported	All	% Coverage
All	214	228	93.9
Required	196	198	99.0
Advisory	18	18	100.0
Document	0	12	0.0

MISRA C++:2008 Supported Rules

Rule	Rule Name	Category
Rule 0-1-1	A project shall not contain unreachable code.	Required
Rule 0-1-2	A project shall not contain infeasible paths.	Required
Rule 0-1-3	A project shall not contain unused variables.	Required
Rule 0-1-4	A project shall not contain non-volatile POD variables having only one use.	Required
Rule 0-1-5	A project shall not contain unused type declarations.	Required
Rule 0-1-6	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.	Required
Rule 0-1-7	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.	Required
Rule 0-1-8	All functions with void return type shall have external side effect(s).	Required
Rule 0-1-9	There shall be no dead code.	Required
Rule 0-1-10	Defined functions shall be called at least once.	Required
Rule 0-1-11	There shall be no unused parameters (named or unnamed) in non-virtual functions.	Required
Rule 0-1-12	There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it.	Required
Rule 0-2-1	An object shall not be assigned to an overlapping object.	Required
Rule 0-3-2	If a function returns error information, then that error information shall be tested.	Required
Rule 2-3-1	Trigraphs shall not be used.	Required
Rule 2-5-1	Digraphs should not be used.	Advisory
Rule 2-7-1	The character sequence /* shall not be used within a C-style comment.	Required
Rule 2-7-2	Sections of code should not be "commented out" using C-style comments.	Required
Rule 2-7-3	Sections of code should not be "commented out" using C++ comments.	Advisory
Rule 2-10-1	Different identifiers shall be typographically unambiguous.	Required
Rule 2-10-2	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.	Required
Rule 2-10-3	A typedef name (including qualification, if any) shall be a unique identifier.	Required
Rule 2-10-4	A class, union or enum name (including qualification, if any) shall be a unique identifier.	Required
Rule 2-10-5	The identifier name of a non-member object or function with static storage duration should not be reused.	Advisory
Rule 2-10-6	If an identifier refers to a type, it shall not also refer to an object or a function in the same scope.	Required
Rule 2-13-1	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used.	Required
Rule 2-13-2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Required
Rule 2-13-3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Required
Rule 2-13-4	Literal suffixes shall be upper case.	Required

Rule 2-13-5	Narrow and wide string literals shall not be concatenated.	Required
Rule 3-1-1	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.	Required
Rule 3-1-2	Functions shall not be declared at block scope.	Required
Rule 3-1-3	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.	Required
Rule 3-2-1	All declarations of an object or function shall have compatible types.	Required
Rule 3-2-2	The One Definition Rule shall not be violated.	Required
Rule 3-2-3	A type, object or function that is used in multiple translation units shall be declared in one and only one file.	Required
Rule 3-2-4	An identifier with external linkage shall have exactly one external definition.	Required
Rule 3-3-1	Objects or functions with external linkage shall be declared in a header file.	Required
Rule 3-3-2	If a function has internal linkage then all re-declarations shall include the static storage class specifier.	Required
Rule 3-4-1	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.	Required
Rule 3-9-1	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.	Required
Rule 3-9-2	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	Advisory
Rule 3-9-3	The underlying bit representations of floating-point values shall not be used.	Required
Rule 4-5-1	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.	Required
Rule 4-5-2	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [ ], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=.	Required
Rule 4-5-3	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.	Required
Rule 4-10-1	NULL shall not be used as an integer value.	Required
Rule 4-10-2	Literal zero (0) shall not be used as the null-pointer constant.	Required
Rule 5-0-1	The value of an expression shall be the same under any order of evaluation that the standard permits.	Required
Rule 5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions.	Advisory
Rule 5-0-3	A cvalue expression shall not be implicitly converted to a different underlying type.	Required
Rule 5-0-4	An implicit integral conversion shall not change the signedness of the underlying type.	Required
Rule 5-0-5	There shall be no implicit floating-integral conversions.	Required
Rule 5-0-6	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.	Required
Rule 5-0-7	There shall be no explicit floating-integral conversions of a cvalue expression.	Required
Rule 5-0-8	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.	Required
Rule 5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.	Required
Rule 5-0-10	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	Required
Rule 5-0-11	The plain char type shall only be used for the storage and use of character values.	Required
Rule 5-0-12	Signed char and unsigned char type shall only be used for the storage and use of numeric values.	Required
Rule 5-0-13	The condition of an if-statement and the condition of an iteration-statement shall have type bool.	Required
Rule 5-0-14	The first operand of a conditional-operator shall have type bool.	Required
Rule 5-0-15	Array indexing shall be the only allowed form of pointer arithmetic.	Required

Rule 5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.	Required
Rule 5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Required
Rule 5-0-18	The relational operators >, >=, < and <= shall not be applied to objects of pointer type, except where they point to the same array.	Required
Rule 5-0-19	The declaration of objects shall contain no more than two levels of pointer indirection.	Required
Rule 5-0-20	Non-constant operands to a binary bitwise operator shall have the same underlying type.	Required
Rule 5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type.	Required
Rule 5-2-1	Each operand of a logical && or || shall be a postfix-expression.	Required
Rule 5-2-2	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.	Required
Rule 5-2-3	Casts from a base class to a derived class should not be performed on polymorphic types.	Advisory
Rule 5-2-4	C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used.	Required
Rule 5-2-5	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.	Required
Rule 5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.	Required
Rule 5-2-7	An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.	Required
Rule 5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Required
Rule 5-2-9	A cast should not convert a pointer type to an integral type.	Advisory
Rule 5-2-10	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	Advisory
Rule 5-2-11	The comma operator, && operator and the || operator shall not be overloaded.	Required
Rule 5-2-12	An identifier with array type passed as a function argument shall not decay to a pointer.	Required
Rule 5-3-1	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.	Required
Rule 5-3-2	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	Required
Rule 5-3-3	The unary & operator shall not be overloaded.	Required
Rule 5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects.	Required
Rule 5-8-1	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
Rule 5-14-1	The right hand operand of a logical && or || operator shall not contain side effects.	Required
Rule 5-18-1	The comma operator shall not be used.	Required
Rule 5-19-1	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	Advisory
Rule 6-2-1	Assignment operators shall not be used in sub-expressions.	Required
Rule 6-2-2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.	Required
Rule 6-2-3	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
Rule 6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Required
Rule 6-4-1	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	Required
Rule 6-4-2	All if ... else if constructs shall be terminated with an else statement.	Required
Rule 6-4-3	A switch statement shall be a well-formed switch statement.	Required
Rule 6-4-4	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	Required

Rule 6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause.	Required
Rule 6-4-6	The final clause of a switch statement shall be the default-clause.	Required
Rule 6-4-7	The condition of a switch statement shall not have bool type.	Required
Rule 6-4-8	Every switch statement shall have at least one case clause.	Required
Rule 6-5-1	A for loop shall contain a single loop-counter which shall not have floating type.	Required
Rule 6-5-2	If loop-counter is not modified by -- or ++, then within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.	Required
Rule 6-5-3	The loop-counter shall not be modified within condition or statement.	Required
Rule 6-5-4	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.	Required
Rule 6-5-5	A loop-control-variable other than loop-counter shall not be modified within condition or expression.	Required
Rule 6-5-6	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.	Required
Rule 6-6-1	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.	Required
Rule 6-6-2	The goto statement shall jump to a label declared later in the same function.	Required
Rule 6-6-3	The continue statement shall only be used within a well-formed for loop.	Required
Rule 6-6-4	For any iteration statement there shall be no more than one break or goto statement used for loop termination.	Required
Rule 6-6-5	A function shall have a single point of exit at the end of the function.	Required
Rule 7-1-1	A variable that is not modified shall be const qualified.	Required
Rule 7-1-2	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.	Required
Rule 7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	Required
Rule 7-3-1	Global namespace shall only contain main, namespace declarations and extern declarations.	Required
Rule 7-3-2	The identifier main shall not be used for a function other than the global function main.	Required
Rule 7-3-3	There shall be no unnamed namespaces in header files.	Required
Rule 7-3-4	using-directives shall not be used.	Required
Rule 7-3-5	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.	Required
Rule 7-3-6	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Required
Rule 7-4-2	Assembler instructions shall only be introduced using the asm declaration.	Required
Rule 7-4-3	Assembly language shall be encapsulated and isolated.	Required
Rule 7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Required
Rule 7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Required
Rule 7-5-3	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.	Required
Rule 7-5-4	Functions should not call themselves, either directly or indirectly.	Advisory
Rule 8-0-1	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.	Required
Rule 8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Required
Rule 8-4-1	Functions shall not be defined using the ellipsis notation.	Required
Rule 8-4-2	The identifiers used for the parameters in a redeclaration of a function shall be identical to those in the declaration.	Required
Rule 8-4-3	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	Required
Rule 8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Required

Rule 8-5-1	All variables shall have a defined value before they are used.	Required
Rule 8-5-2	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.	Required
Rule 8-5-3	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
Rule 9-3-1	const member functions shall not return non-const pointers or references to class-data.	Required
Rule 9-3-2	Member functions shall not return non-const handles to class-data.	Required
Rule 9-3-3	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.	Required
Rule 9-5-1	Unions shall not be used.	Required
Rule 9-6-2	Bit-fields shall be either bool type or an explicitly unsigned or signed integral type.	Required
Rule 9-6-3	Bit-fields shall not have enum type.	Required
Rule 9-6-4	Named bit-fields with signed integer type shall have a length of more than one bit.	Required
Rule 10-1-1	Classes should not be derived from virtual bases.	Advisory
Rule 10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy.	Required
Rule 10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.	Required
Rule 10-2-1	All accessible entity names within a multiple inheritance hierarchy should be unique.	Advisory
Rule 10-3-1	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy.	Required
Rule 10-3-2	Each overriding virtual function shall be declared with the virtual keyword.	Required
Rule 10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.	Required
Rule 11-0-1	Member data in non-POD class types shall be private.	Required
Rule 12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor.	Required
Rule 12-1-2	All constructors of a class should explicitly call a constructor for all of its immediate base classes and all virtual base classes.	Advisory
Rule 12-1-3	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Required
Rule 12-8-1	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.	Required
Rule 12-8-2	The copy assignment operator shall be declared protected or private in an abstract class.	Required
Rule 14-5-1	A non-member generic function shall only be declared in a namespace that is not an associated namespace.	Required
Rule 14-5-2	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter.	Required
Rule 14-5-3	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.	Required
Rule 14-6-1	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.	Required
Rule 14-6-2	The function chosen by overload resolution shall resolve to a function declared previously in the translation unit.	Required
Rule 14-7-1	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once.	Required
Rule 14-7-2	For any given template specialization, an explicit instantiation of the template with the template-arguments used in the specialization shall not render the program ill-formed.	Required
Rule 14-7-3	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.	Required
Rule 14-8-1	Overloaded function templates shall not be explicitly specialized.	Required
Rule 14-8-2	The viable function set for a function call should either contain no function specializations, or only contain function specializations.	Advisory
Rule 15-0-2	An exception object should not have pointer type.	Advisory
Rule 15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement.	Required

Rule 15-1-1	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.	Required
Rule 15-1-2	NULL shall not be thrown explicitly.	Required
Rule 15-1-3	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.	Required
Rule 15-3-1	Exceptions shall be raised only after start-up and before termination of the program.	Required
Rule 15-3-2	There should be at least one exception handler to catch all otherwise unhandled exceptions.	Advisory
Rule 15-3-3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference nonstatic members from this class or its bases.	Required
Rule 15-3-4	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.	Required
Rule 15-3-5	A class type exception shall always be caught by reference.	Required
Rule 15-3-6	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.	Required
Rule 15-3-7	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.	Required
Rule 15-4-1	If a function is declared with an exception-specification, then all declarations of the same function (in other translation units) shall be declared with the same set of type-ids.	Required
Rule 15-5-1	A class destructor shall not exit with an exception.	Required
Rule 15-5-2	Where a function's declaration includes an exception specification, the function shall only be capable of throwing exceptions of the indicated type(s).	Required
Rule 15-5-3	The terminate() function shall not be called implicitly.	Required
Rule 16-0-1	#include directives in a file shall only be preceded by other preprocessor directives or comments.	Required
Rule 16-0-2	Macros shall only be #defined or #undefd in the global namespace.	Required
Rule 16-0-3	#undef shall not be used.	Required
Rule 16-0-4	Function-like macros shall not be defined.	Required
Rule 16-0-5	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	Required
Rule 16-0-6	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
Rule 16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.	Required
Rule 16-0-8	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.	Required
Rule 16-1-1	The defined preprocessor operator shall only be used in one of the two standard forms.	Required
Rule 16-1-2	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
Rule 16-2-1	The pre-processor shall only be used for file inclusion and include guards.	Required
Rule 16-2-2	C++ macros shall only be used for include guards, type qualifiers, or storage class specifiers.	Required
Rule 16-2-3	Include guards shall be provided.	Required
Rule 16-2-4	The ', ", /* or // characters shall not occur in a header file name.	Required
Rule 16-2-5	The \ character should not occur in a header file name.	Advisory
Rule 16-2-6	The #include directive shall be followed by either a or "filename" sequence.	Required
Rule 16-3-1	There shall be at most one occurrence of the # or ## operators in a single macro definition.	Required
Rule 16-3-2	The # and ## operators should not be used.	Advisory
Rule 17-0-1	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.	Required
Rule 17-0-2	The names of standard library macros and objects shall not be reused.	Required
Rule 17-0-3	The names of standard library functions shall not be overridden.	Required

Rule 17-0-5	The setjmp macro and the longjmp function shall not be used.	Required
Rule 18-0-1	C++ libraries with corresponding C compatible libraries must use the C++ version.	Required
Rule 18-0-2	The library functions atof, atoi and atol from library shall not be used.	Required
Rule 18-0-3	The library functions abort, exit, getenv and system from library shall not be used.	Required
Rule 18-0-4	The time handling functions of library shall not be used.	Required
Rule 18-0-5	The unbounded functions of library shall not be used.	Required
Rule 18-2-1	The macro offsetof shall not be used.	Required
Rule 18-4-1	Dynamic heap memory allocation shall not be used.	Required
Rule 18-7-1	The signal handling facilities of shall not be used.	Required
Rule 19-3-1	The error indicator errno shall not be used.	Required
Rule 27-0-1	The stream input/output library shall not be used.	Required

MISRA C:2012 Rule Coverage

	Supported	All	% Coverage
All	182	173	105.2
Mandatory	16	16	100.0
Required	126	118	106.8
Advisory	40	39	102.6

MISRA C:2012 Supported Rules

Rule	Rule Name	Category
Directive 4.3	Assembly language shall be encapsulated and isolated.	Required
Directive 4.4	Sections of code should not be "commented out".	Advisory
Directive 4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous.	Advisory
Directive 4.6	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	Advisory
Directive 4.7	If a function returns error information, then that error information shall be tested.	Required
Directive 4.8	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
Directive 4.9	A function should be used in preference to a function-like macro where they are interchangeable.	Advisory
Directive 4.10	Precautions shall be taken in order to prevent the contents of a header file being included more than once.	Required
Directive 4.11	The validity of values passed to library functions shall be checked.	Required
Directive 4.12	Dynamic memory allocation shall not be used.	Required
Directive 4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.	Advisory
Directive 4.14	The validity of values received from external sources shall be checked.	Required
Directive 5.1	There shall be no data races between threads.	Required
Directive 5.2	There shall be no deadlocks between threads.	Required
Rule 1.1	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits.	Required
Rule 1.2	Language extensions should not be used.	Advisory
Rule 1.4	Emergent language features shall not be used.	Required
Rule 2.1	A project shall not contain unreachable code.	Required
Rule 2.2	There shall be no dead code.	Required
Rule 2.3	A project should not contain unused type declarations.	Advisory

Rule 2.4	A project should not contain unused tag declarations.	Advisory
Rule 2.5	A project should not contain unused macro declarations.	Advisory
Rule 2.6	A function should not contain unused label declarations.	Advisory
Rule 2.7	There should be no unused parameters in functions.	Advisory
Rule 3.1	The character sequences /* and // shall not be used within a comment.	Required
Rule 3.2	Line-splicing shall not be used in // comments.	Required
Rule 4.1	Octal and hexadecimal escape sequences shall be terminated.	Required
Rule 4.2	Trigraphs should not be used.	Advisory
Rule 5.1	External identifiers shall be distinct.	Required
Rule 5.2	Identifiers declared in the same scope and name space shall be distinct.	Required
Rule 5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	Required
Rule 5.4	Macro identifiers shall be distinct.	Required
Rule 5.5	Identifiers shall be distinct from macro names.	Required
Rule 5.6	A typedef name shall be a unique identifier.	Required
Rule 5.7	A tag name shall be a unique identifier.	Required
Rule 5.8	Identifiers that define objects or functions with external linkage shall be unique.	Required
Rule 5.9	Identifiers that define objects or functions with internal linkage should be unique.	Advisory
Rule 6.1	Bit-fields shall only be declared with an appropriate type.	Required
Rule 6.2	Single-bit named bit fields shall not be of a signed type.	Required
Rule 6.3	A bit field shall not be declared as a member of a union.	Required
Rule 7.1	Octal constants shall not be used.	Required
Rule 7.2	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.	Required
Rule 7.3	The lowercase character "l" shall not be used in a literal suffix.	Required
Rule 7.4	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char".	Required
Rule 8.1	Types shall be explicitly specified.	Required
Rule 8.2	Function types shall be in prototype form with named parameters.	Required
Rule 8.3	All declarations of an object or function shall use the same names and type qualifiers.	Required
Rule 8.4	A compatible declaration shall be visible when an object or function with external linkage is defined.	Required
Rule 8.5	An external object or function shall be declared once in one and only one file.	Required
Rule 8.6	An identifier with external linkage shall have exactly one external definition.	Required
Rule 8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.	Advisory
Rule 8.8	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.	Required
Rule 8.9	An object should be defined at block scope if its identifier only appears in a single function.	Advisory
Rule 8.10	An inline function shall be declared with the static storage class.	Required
Rule 8.11	When an array with external linkage is declared, its size should be explicitly specified.	Advisory
Rule 8.12	Within a n enumerator list, the value of an implicitly-specified enumeration constant shall be unique.	Required
Rule 8.13	A pointer should point to a const-qualified type whenever possible.	Advisory
Rule 8.14	The restrict type qualifier shall not be used.	Required
Rule 9.1	The value of an object with automatic storage duration shall not be read before it has been set.	Mandatory
Rule 9.2	The initializer for an aggregate or union shall be enclosed in braces.	Required
Rule 9.3	Arrays shall not be partially initialized.	Required
Rule 9.4	An element of an object shall not be initialized more than once.	Required
Rule 9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.	Required

Rule 10.1	Operands shall not be of an inappropriate essential type.	Required
Rule 10.2	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operation.	Required
Rule 10.3	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
Rule 10.4	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.	Required
Rule 10.5	The value of an expression should not be cast to an inappropriate essential type.	Advisory
Rule 10.6	The value of a composite expression shall not be assigned to an object with wider essential type.	Required
Rule 10.7	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
Rule 10.8	The value of a composite expression shall not be cast to a different essential type category or a wider essential type.	Required
Rule 11.1	Conversions shall not be performed between a pointer to a function and any other type.	Required
Rule 11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type.	Required
Rule 11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type.	Required
Rule 11.4	A conversion should not be performed between a pointer to object and an integer type.	Advisory
Rule 11.5	A conversion should not be performed from pointer to void into pointer to object.	Advisory
Rule 11.6	A cast shall not be performed between pointer to void and an arithmetic type.	Required
Rule 11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type.	Required
Rule 11.8	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.	Required
Rule 11.9	The macro NULL shall be the only permitted form of integer null pointer constant.	Required
Rule 12.1	The precedence of operators within expressions should be made explicit.	Advisory
Rule 12.2	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
Rule 12.3	The comma operator should not be used.	Advisory
Rule 12.4	Evaluation of constant expressions should not lead to unsigned integer wrap-around.	Advisory
Rule 12.5	The sizeof operator shall not have an operand which is a function parameter declared as "array of type".	Mandatory
Rule 12.6	Structure and union members of atomic objects shall not be directly accessed.	Required
Rule 13.1	Initializer lists shall not contain persistent side effects.	Required
Rule 13.2	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.	Required
Rule 13.3	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
Rule 13.4	The result of an assignment operator should not be used.	Advisory
Rule 13.5	The right hand operand of a logical && or || operator shall not contain persistent side effects.	Required
Rule 13.6	The operand of the sizeof operator shall not contain any expression which has potential side effects.	Mandatory
Rule 14.1	A loop counter shall not have essentially floating type.	Required
Rule 14.2	A for loop shall be well-formed.	Required
Rule 14.3	Controlling expressions shall not be invariant.	Required
Rule 14.4	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type.	Required
Rule 15.1	The goto statement should not be used.	Advisory
Rule 15.2	The goto statement shall jump to a label declared later in the same function.	Required

Rule 15.3	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.	Required
Rule 15.4	There should be no more than one break or goto statement used to terminate any iteration statement.	Advisory
Rule 15.5	A function should have a single point of exit at the end.	Advisory
Rule 15.6	The body of an iteration-statement or a selection-statement shall be a compound statement.	Required
Rule 15.7	All if ... else if constructs shall be terminated with an else statement.	Required
Rule 16.1	All switch statements shall be well formed.	Required
Rule 16.2	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	Required
Rule 16.3	An unconditional break statement shall terminate every switch-clause.	Required
Rule 16.4	Every switch statement shall have a default label.	Required
Rule 16.5	A default label shall appear as either the first or the last switch label of a switch statement.	Required
Rule 16.6	Every switch statement shall have at least two switch clauses.	Required
Rule 16.7	A switch expression shall not have an essentially Boolean type.	Required
Rule 17.1	The features of shall not be used.	Required
Rule 17.2	Functions shall not call themselves, either directly or indirectly.	Required
Rule 17.3	A function shall not be declared implicitly.	Mandatory
Rule 17.4	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	Mandatory
Rule 17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.	Advisory
Rule 17.6	The declaration of an array parameter shall not contain the static keyword between the [].	Mandatory
Rule 17.7	The value returned by a function having non-void return type shall be used.	Required
Rule 17.8	A function parameter should not be modified.	Advisory
Rule 17.10	A function declared with a _Noreturn function specifier shall have void return type.	Required
Rule 17.12	A function identifier should only be used with either a preceding &, or with a parenthesized parameter list.	Advisory
Rule 18.1	A pointer resulting from arithmetic on a pointer operand shall address an elements of the same array as that pointer operand.	Required
Rule 18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Required
Rule 18.3	The relational operators >, >=, < and <= shall only be applied to pointers that point into the same object.	Required
Rule 18.4	The +, -, += and -= operators should not be applied to an expression of pointer type.	Advisory
Rule 18.5	Declarations should contain no more than two levels of pointer nesting.	Advisory
Rule 18.6	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
Rule 18.7	Flexible array members shall not be declared.	Required
Rule 18.8	Variable-length array types shall not be used.	Required
Rule 19.1	An object shall not be assigned or copied to an overlapping object.	Mandatory
Rule 19.2	The union keyword should not be used.	Advisory
Rule 20.1	#include directives should only be preceded by preprocessor directives or comments.	Advisory
Rule 20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.	Required
Rule 20.3	The #include directive shall be followed by either a or "filename" sequence.	Required
Rule 20.4	A macro shall not be defined with the same name as a keyword.	Required
Rule 20.5	#undef should not be used.	Advisory
Rule 20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument.	Required
Rule 20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.	Required

Rule 20.8	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.	Required
Rule 20.9	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.	Required
Rule 20.10	The # and ## preprocessor operators should not be used.	Advisory
Rule 20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.	Required
Rule 20.12	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
Rule 20.13	A line whose first token is # shall be a valid preprocessing directive.	Required
Rule 20.14	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
Rule 21.1	#define and #undef shall not be used on a reserved identifier or reserved macro name.	Required
Rule 21.2	A reserved identifier or macro name shall not be declared.	Required
Rule 21.3	The memory allocation and deallocation functions of shall not be used.	Required
Rule 21.4	The standard header file shall not be used.	Required
Rule 21.5	The standard header file shall not be used.	Required
Rule 21.6	The Standard Library input/output functions shall not be used.	Required
Rule 21.7	The Standard Library functions atof, atoi, atol and atoll of shall not be used.	Required
Rule 21.8	The Standard Library termination functions of shall not be used.	Required
Rule 21.9	The Standard Library functions bsearch and qsort of shall not be used.	Required
Rule 21.10	The Standard Library time and date functions shall not be used.	Required
Rule 21.11	The standard header file shall not be used.	Required
Rule 21.12	The exception handling features of should not be used.	Advisory
Rule 21.13	Any value passed to a function in shall be representable as an unsigned char or be the value EOF.	Mandatory
Rule 21.14	The Standard Library function memcmp shall not be used to compare null terminated strings.	Required
Rule 21.15	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types.	Required
Rule 21.16	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
Rule 21.17	Use of the string handling functions from shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters.	Mandatory
Rule 21.18	The size_t argument passed to any function in shall have an appropriate value.	Mandatory
Rule 21.19	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
Rule 21.20	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
Rule 21.21	The Standard Library function system of shall not be used.	Required
Rule 21.24	The random number generator functions of shall not be used.	Required
Rule 21.25	All memory synchronization operations shall be executed in sequentially consistent order.	Required
Rule 22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.	Required
Rule 22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function.	Mandatory
Rule 22.3	The same file shall not be open for read and write access at the same time on different streams.	Required
Rule 22.4	There shall be no attempt to write to a stream which has been opened as read-only.	Mandatory
Rule 22.5	A pointer to a FILE object shall not be dereferenced.	Mandatory
Rule 22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	Mandatory

Rule 22.7	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF.	Required
Rule 22.8	The value of errno shall be set to zero prior to a call to an errno-setting-function.	Required
Rule 22.9	The value of errno shall be tested against zero after calling an errno-setting-function.	Required
Rule 22.10	The value of errno shall only be tested when the last function to be called was an errno-setting-function.	Required
Rule 22.11	A thread that was previously either joined or detached shall not be subsequently joined nor detached.	Required
Rule 22.16	All mutex objects locked by a thread shall be explicitly unlocked by the same thread.	Required
Rule 23.2	A generic selection that is not expanded from a macro shall not contain potential side effects in the controlling expression.	Required
Rule 23.3	A generic selection should contain at least one non-default association.	Advisory
Rule 23.8	A default association shall appear as either the first or the last association of a generic selection.	Required