MISRA C++:2008

Perforce QAC for C++ 2026.1

(MISRA C++:2008 First Edition)

Rule Enforcement Summary

		Total
a	Total Number of Rules	228
b	Total Number of 'Not Statically Enforceable' Rules (Assisted/Unassisted)	12
c	Total Number of Enforceable Rules (a-b)	216
d	Total Number of Enforced Rules	212
e	Total Number of Unenforced Rules (c-d)	4
f	Enforced Rules Percentage (d/c)	98%
g	Unenforced Rules Percentage (e/c)	2%

Id	Description	Category	Enforced
Rule-0_1_1	A project shall not contain unreachable code.	Required	Yes
Rule-0_1_2	A project shall not contain infeasible paths.	Required	Yes
Rule-0_1_3	A project shall not contain unused variables.	Required	Yes
Rule-0_1_4	A project shall not contain non-volatile POD variables having only one use.	Required	Yes
Rule-0_1_5	A project shall not contain unused type declarations.	Required	Yes
Rule-0_1_6	A project shall not contain instances of non-volatile variables being given values that are never used subsequently.	Required	Yes
Rule-0_1_7	The value returned by a function having a non void return type that is not an overloaded operator shall be used.	Required	Yes
Rule-0_1_8	All functions with void return type shall have external side effect(s).	Required	Yes
Rule-0_1_9	There shall be no dead / redundant code.	Required	Yes

Id	Description	Category	Enforced
Rule-0_1_10	Every defined function shall be called at least once.	Required	Yes
Rule-0_1_11	There shall be no unused parameters (named or unnamed) in non-virtual functions.	Required	Yes
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	No
Rule-0_2_1	An object shall not be assigned to an overlapping object.	Required	Yes
Rule-0_3_2	If a function generates error information, then that error information shall be tested.	Required	Yes
Rule-1_0_1	All code shall conform to ISO 14882:2003 "The C++ Standard Incorporating Technical Corrigendum 1".	Required	Yes
Rule-2_3_1	Trigraphs shall not be used.	Required	Yes
Rule-2_5_1	Digraphs shall not be used.	Advisory	Yes
Rule-2_7_1	The character sequence /* shall not be used within a C-style comment.	Required	Yes
Rule-2_7_2	Sections of code shall not be "commented out" using C-style comments.	Required	Yes
Rule-2_7_3	Sections of code should not be "commented out" using C++ comments.	Advisory	Yes
Rule-2_10_1	Different identifiers shall be typographically unambiguous.	Required	Yes
Rule-2_10_2	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.	Required	Yes
Rule-2_10_3	A typedef name (including qualification, if any) shall be a unique identifier.	Required	Yes
Rule-2_10_4	A class, union or enum name (including qualification, if any) shall be a unique identifier.	Required	No
Rule-2_10_5	The identifier name of a non-member object or function with static storage duration should not be reused.	Advisory	Yes
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	Yes

Id	Description	Category	Enforced
Rule-2_13_1	Only those escape sequences that are defined in the ISO C++ standard shall be used.	Required	Yes
Rule-2_13_2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Required	Yes
Rule-2_13_3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Required	Yes
Rule-2_13_4	Literal suffixes shall be upper case.	Required	Yes
Rule-2_13_5	Narrow and wide string literals shall not be concatenated.	Required	Yes
Rule-3_1_1	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.	Required	Yes
Rule-3_1_2	Functions shall not be declared at block scope.	Required	Yes
Rule-3_1_3	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialisation.	Required	Yes
Rule-3_2_1	All declarations of an object or function shall have compatible types.	Required	Yes
Rule-3_2_2	The One Definition Rule shall not be violated.	Required	Yes
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	Yes
Rule-3_2_4	An identifier with external linkage shall have exactly one definition.	Required	Yes
Rule-3_3_1	Objects or functions with external linkage shall be declared in a header file.	Required	Yes
Rule-3_3_2	If a function has internal linkage then all re-declarations shall include the static storage class specifier.	Required	Yes
Rule-3_4_1	An identifier declared to be an object or type shall be defined in a block that minimises its visibility.	Required	Yes
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	Yes
Rule-3_9_2	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	Advisory	Yes
Rule-3_9_3	The underlying bit representations of floating-point values shall not be used.	Required	Yes

Id	Description	Category	Enforced
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 !=, and the conditional operator.	Required	Yes
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 !=, and the relational operators <, <=, >, >=.	Required	Yes
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 =, and the equality operators == and !=.	Required	Yes
Rule-4_10_1	NULL shall not be used as an integer value.	Required	Yes
Rule-4_10_2	Literal zero (0) shall not be used as the null pointer constant.	Required	Yes
Rule-5_0_1	The value of an expression shall be the same under any order of evaluation that the standard permits.	Required	Yes
Rule-5_0_2	Limited dependence should be placed on C++ operator precedence rules in expressions.	Advisory	Yes
Rule-5_0_3	A cvalue expression shall not be implicitly converted to a different underlying type.	Required	Yes
Rule-5_0_4	An implicit integral conversion shall not change the signedness of the underlying type.	Required	Yes
Rule-5_0_5	There shall be no implicit floating-integral conversions.	Required	Yes
Rule-5_0_6	An implicit integral or floating point conversion shall not reduce the size of the underlying type.	Required	Yes
Rule-5_0_7	There shall be no explicit floating-integral conversions of a cvalue expression.	Required	Yes
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	Yes
Rule-5_0_9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.	Required	Yes
Rule-5_0_10	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	Yes

Id	Description	Category	Enforced
Rule-5_0_11	The plain char type shall only be used for the storage and use of character values.	Required	Yes
Rule-5_0_12	Signed char and unsigned char type shall only be used for the storage and use of numeric values.	Required	Yes
Rule-5_0_13	The condition of an if-statement and the condition of an iteration-statement shall have type bool.	Required	Yes
Rule-5_0_14	The first operand of a conditional-operator shall have type bool.	Required	Yes
Rule-5_0_15	Array indexing shall be the only form of pointer arithmetic.	Required	Yes
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	Yes
Rule-5_0_17	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Required	Yes
Rule-5_0_18	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.	Required	Yes
Rule-5_0_19	The declaration of objects shall contain no more than two levels of pointer indirection.	Required	Yes
Rule-5_0_20	Non constant operands to a binary bitwise operator shall have the same underlying type.	Required	Yes
Rule-5_0_21	Bitwise operators shall only be applied to operands of unsigned underlying type.	Required	Yes
Rule-5_2_1	Each operand of a logical && or || shall be a postfix-expression.	Required	Yes
Rule-5_2_2	A pointer to a virtual base class shall only be cast to a pointer to a derived class using only dynamic_cast.	Required	Yes
Rule-5_2_3	Casts from a base class to a derived class shall not be performed on polymorphic types.	Advisory	Yes
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	Yes
Rule-5_2_5	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.	Required	Yes
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	Yes

Id	Description	Category	Enforced
Rule-5_2_7	An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.	Required	Yes
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	Yes
Rule-5_2_9	A cast should not convert a pointer type to an integral type.	Advisory	Yes
Rule-5_2_10	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	Advisory	Yes
Rule-5_2_11	The comma operator, && operator and the || operator shall not be overloaded.	Required	Yes
Rule-5_2_12	An identifier with array type passed as a function argument shall not decay to a pointer.	Required	Yes
Rule-5_3_1	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.	Required	Yes
Rule-5_3_2	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	Required	Yes
Rule-5_3_3	The unary & operator shall not be overloaded.	Required	Yes
Rule-5_3_4	Evaluation of the operand to the sizeof operator shall not contain side effects.	Required	Yes
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	Yes
Rule-5_14_1	The right hand operand of a logical && or || operator shall not contain side effects.	Required	Yes
Rule-5_17_1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.	Required	Yes
Rule-5_18_1	The comma operator shall not be used.	Required	Yes
Rule-5_19_1	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	Advisory	Yes
Rule-6_2_1	Assignment operators shall not be used in sub-expressions.	Required	Yes
Rule-6_2_2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.	Required	Yes

Id	Description	Category	Enforced
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	Yes
Rule-6_3_1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Required	Yes
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	Yes
Rule-6_4_2	All if ... else if constructs shall be terminated with an else clause.	Required	Yes
Rule-6_4_3	A switch statement shall be a well-formed switch statement.	Required	Yes
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	Yes
Rule-6_4_5	An unconditional throw or break statement shall terminate every non-empty switch clause.	Required	Yes
Rule-6_4_6	The final clause of a switch statement shall be the default clause.	Required	Yes
Rule-6_4_7	The condition of a switch-statement shall not have bool type.	Required	Yes
Rule-6_4_8	Every switch statement shall have at least one case clause.	Required	Yes
Rule-6_5_1	A for loop shall contain a single loop counter which shall not have floating type.	Required	Yes
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	Yes
Rule-6_5_3	The loop counter shall not be modified within condition or statement.	Required	Yes
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	Yes
Rule-6_5_5	A loop control variable other than the loop counter shall not be modified within condition or expression.	Required	Yes
Rule-6_5_6	A loop control variable other than the loop counter which is modified in statement shall have type bool.	Required	Yes
Rule-6_6_1	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.	Required	Yes

Id	Description	Category	Enforced
Rule-6_6_2	The goto statement shall jump to a label declared later in the same function body.	Required	Yes
Rule-6_6_3	The continue statement shall only be used within a well-formed for loop.	Required	Yes
Rule-6_6_4	For any iteration statement there shall be no more than one break or goto statement used for loop termination.	Required	Yes
Rule-6_6_5	A function shall have a single point of exit at the end of the function.	Required	Yes
Rule-7_1_1	A variable which is not modified shall be const qualified.	Required	Yes
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	Yes
Rule-7_2_1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	Required	Yes
Rule-7_3_1	The global namespace shall only contain main, namespace declarations and extern "C" declarations.	Required	Yes
Rule-7_3_2	The identifier main shall not be used for a function other than the global function main.	Required	Yes
Rule-7_3_3	There shall be no unnamed namespaces in header files.	Required	Yes
Rule-7_3_4	using-directives shall not be used.	Required	Yes
Rule-7_3_5	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.	Required	Yes
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	Yes
Rule-7_4_2	Assembler instructions shall only be introduced using the asm declaration.	Required	Yes
Rule-7_4_3	Assembly language shall be encapsulated and isolated.	Required	Yes
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	Yes
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	Yes

Id	Description	Category	Enforced
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	Yes
Rule-7_5_4	Functions shall not call themselves, either directly or indirectly.	Advisory	Yes
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	Yes
Rule-8_3_1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or shall not specify any default arguments.	Required	Yes
Rule-8_4_1	Functions shall not be defined using the ellipsis notation.	Required	Yes
Rule-8_4_2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	Required	Yes
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	Yes
Rule-8_4_4	A function identifier shall either be used to call the function or it shall be preceded by &.	Required	Yes
Rule-8_5_1	All variables shall have a defined value before they are used.	Required	Yes
Rule-8_5_2	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.	Required	Yes
Rule-8_5_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	Yes
Rule-9_3_1	Const member functions shall not return non-const pointers or references to class-data.	Required	Yes
Rule-9_3_2	Member functions shall not return non-const handles to class-data.	Required	Yes
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	Yes
Rule-9_5_1	Unions shall not be used.	Required	Yes
Rule-9_6_2	Bit-fields shall be either bool type or an explicitly unsigned or signed integral type.	Required	Yes
Rule-9_6_3	Bit-fields shall not have enum type.	Required	Yes

Id	Description	Category	Enforced
Rule-9_6_4	Named bit-fields with signed integer type shall have a length of more than one bit.	Required	Yes
Rule-10_1_1	Classes should not be derived from virtual bases.	Advisory	Yes
Rule-10_1_2	A base class shall only be declared virtual if it is used in a diamond hierarchy.	Required	Yes
Rule-10_1_3	Base classes shall not be both virtual and non-virtual in the same hierarchy.	Required	Yes
Rule-10_2_1	All accessible entity names within a multiple inheritance hierarchy should be unique.	Advisory	Yes
Rule-10_3_1	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy to the class in which it is to be used.	Required	Yes
Rule-10_3_2	Each overriding virtual function shall be declared with the virtual keyword.	Required	Yes
Rule-10_3_3	Virtual functions shall only be declared pure when first introduced.	Required	Yes
Rule-11_0_1	Member data in non-POD class types shall be private.	Required	Yes
Rule-12_1_1	An object's dynamic type shall not be used from the body of its constructor or destructor.	Required	Yes
Rule-12_1_2	All constructors of a class shall explicitly call a constructor for all of its immediate base classes and all virtual base classes.	Advisory	Yes
Rule-12_1_3	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Required	Yes
Rule-12_8_1	A copy constructor shall only initialise its base classes and the non-static members of the class of which it is a member.	Required	Yes
Rule-12_8_2	The copy assignment operator shall be declared protected or private in an abstract class.	Required	Yes
Rule-14_5_1	A non-member generic function shall only be declared in a namespace that is not an associated namespace.	Required	Yes
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	Yes

Id	Description	Category	Enforced
Rule-14_5_3	A copy assignment operator shall be declared when there is a template assignment operator whose parameter is a generic parameter.	Required	Yes
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	Yes
Rule-14_6_2	The function or operator chosen by overload resolution shall resolve to a function declared previously in the translation unit.	Required	Yes
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	No
Rule-14_7_2	For any given template specialisation, an explicit instantiation of the template with the template-arguments used in the specialisation shall not render the program ill-formed.	Required	No
Rule-14_7_3	All partial and explicit specialisations for a template shall be declared in the same file as the declaration of their primary template.	Required	Yes
Rule-14_8_1	Overloaded function templates shall not be explicitly specialised.	Required	Yes
Rule-14_8_2	The viable function set for a function call shall either contain no function specialisations, or only contain function specialisations.	Advisory	Yes
Rule-15_0_2	An exception object should not have pointer type.	Advisory	Yes
Rule-15_0_3	Control shall not be transferred into a try or catch block using a goto or a switch statement.	Required	Yes
Rule-15_1_1	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.	Required	Yes
Rule-15_1_2	NULL shall not be thrown explicitly.	Required	Yes
Rule-15_1_3	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.	Required	Yes
Rule-15_3_1	Exceptions shall be raised only after start-up and before termination of the program.	Required	Yes
Rule-15_3_2	There should be at least one exception handler to catch all otherwise unhandled exceptions	Advisory	Yes

Id	Description	Category	Enforced
Rule-15_3_3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.	Required	Yes
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	Yes
Rule-15_3_5	A class type exception shall always be caught by reference.	Required	Yes
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	Yes
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	Yes
Rule-15_4_1	If a function is declared with a throw specification, then all declarations of the same function (in other translation units) shall be declared with the same set of type-ids.	Required	Yes
Rule-15_5_1	A class destructor shall not exit with an exception.	Required	Yes
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	Yes
Rule-15_5_3	The terminate() function shall not be called implicitly.	Required	Yes
Rule-16_0_1	#include directives in a file shall only be preceded by other preprocessor directives or comments.	Required	Yes
Rule-16_0_2	Macros shall only be #define'd or #undef'd in the global namespace.	Required	Yes
Rule-16_0_3	#undef shall not be used.	Required	Yes
Rule-16_0_4	Function-like macros shall not be defined.	Required	Yes
Rule-16_0_5	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	Required	Yes
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	Yes
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	Yes

Id	Description	Category	Enforced
Rule-16_0_8	If the # token appears as the first token on a line, then it must be immediately followed by a preprocessing token.	Required	Yes
Rule-16_1_1	The defined preprocessor operator shall only be used in one of the two standard forms.	Required	Yes
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	Yes
Rule-16_2_1	The pre-processor shall only be used for file inclusion and include guards.	Required	Yes
Rule-16_2_2	C++ macros shall only be used for: include guards, type qualifiers, or storage class specifiers.	Required	Yes
Rule-16_2_3	Include guards shall be provided using one of the following two forms:	Required	Yes
Rule-16_2_4	The ', ", /* or // characters shall not occur in a header file name.	Required	Yes
Rule-16_2_5	The \ character should not occur in a header file name.	Advisory	Yes
Rule-16_2_6	The #include directive shall be followed by either a <filename> or "filename" sequence.	Required	Yes
Rule-16_3_1	There shall be at most one occurrence of the # or ## operators in a single macro definition.	Required	Yes
Rule-16_3_2	The # and ## operators should not be used.	Advisory	Yes
Rule-17_0_1	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.	Required	Yes
Rule-17_0_2	The names of standard library macros and objects shall not be reused.	Required	Yes
Rule-17_0_3	The names of standard library functions shall not be overridden.	Required	Yes
Rule-17_0_5	The setjmp macro and the longjmp function shall not be used.	Required	Yes
Rule-18_0_1	The C library shall not be used.	Required	Yes
Rule-18_0_2	The library functions atof, atoi and atol from library <cstdlib> shall not be used.	Required	Yes
Rule-18_0_3	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.	Required	Yes

Id	Description	Category	Enforced
Rule-18_0_4	The time handling functions of library <ctime> shall not be used.	Required	Yes
Rule-18_0_5	The unbounded functions of library <cstring> shall not be used.	Required	Yes
Rule-18_2_1	The macro offsetof shall not be used.	Required	Yes
Rule-18_4_1	Dynamic heap memory allocation shall not be used.	Required	Yes
Rule-18_7_1	The signal handling facilities of <csignal> shall not be used.	Required	Yes
Rule-19_3_1	The error indicator errno shall not be used.	Required	Yes
Rule-27_0_1	The stream input/output library <cstdio> shall not be used.	Required	Yes

Not Statically Enforceable Rules

Id	Description	Assisted
Rule-0_3_1	Minimisation of run-time failures shall be ensured by the use of at least one of: static analysis tools/techniques, dynamic analysis tools/techniques, or explicit coding of checks to handle run-time faults.	Assisted
Rule-0_4_1	Use of scaled-integer or fixed-point arithmetic shall be documented.	Unassisted
Rule-0_4_2	Use of floating-point arithmetic shall be documented.	Assisted
Rule-0_4_3	Floating-point implementations should comply with a defined floating-point standard.	Unassisted
Rule-1_0_2	Multiple compilers shall only be used if they have a common, defined interface.	Unassisted
Rule-1_0_3	The implementation of integer division in the chosen compiler shall be determined and documented.	Unassisted
Rule-2_2_1	The character set and the corresponding encoding shall be documented.	Unassisted
Rule-7_4_1	All usage of assembler shall be documented.	Assisted
Rule-9_6_1	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.	Unassisted
Rule-15_0_1	Exceptions should only be used for error handling.	Unassisted
Rule-16_6_1	All uses of the #pragma directive shall be documented.	Assisted
Rule-17_0_4	All library code shall conform to MISRA-C++.	Unassisted