

MISRA C:2023 Standards Model Summary for C / C++

The LDRA tool suite® is developed and certified to BS EN ISO 9001:2015, TÜV SÜD and SGS-TÜV Saar.

This information is applicable to version 10.2.0 of the LDRA tool suite®.
It is correct as of 11th September 2023.
© Copyright 2023 LDRA Ltd. All rights reserved.

Compliance is measured against
"MISRA C:2023 Guidelines for the use of the C language in critical systems (Third Edition, first revision), MISRA C:2012 Amendment 3, MISRA C:2012 Technical Corrigendum 2, MISRA C:2012 Amendment 2, MISRA C:2012 Amendment 4 updates for ISO/IEC 9899:2011/2018 Phase 3 - Multi-threading and Atomics"
2023
Copyright © MISRA

Further information is available at <http://www.misra.org.uk>

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Mandatory	0	12	6	2	0	20
Required	11	111	18	13	2	155
Advisory	7	33	5	1	0	46
Total	18	156	29	16	2	221

MISRA C:2023 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
D.1.1	Required	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	This directive is mainly intended for the user to document the behaviour of implementation dependent features. Some of these features are highlighted using these LDRA standards.	69 S	#pragma used.
				584 S	Remainder of % op could be negative.
D.2.1	Required	All source files shall compile without any compilation errors	Coverage of C constraint errors is covered by rule 1.1. This rule applies to other errors that might be produced by the user's compiler.		
D.3.1	Required	All code shall be traceable to documented requirements			
D.4.1	Required	Run-time failures shall be minimised	This directive cannot be completely checked using static analysis. Dynamic analysis and coverage analysis can be used to minimise run-time errors. Errors such as exceeding array bounds are covered by more specific rules. The checks on numeric overflow/underflow are restricted to those cases where the value of a variable can be determined statically	43 D	Divide by zero found.
				45 D	Pointer not checked for null before use.
				115 D	Copy length parameter not checked before use.
				123 D	File pointer not checked for null before use.
				127 D	Local or member denominator not checked before use.
				128 D	Global pointer not checked within this procedure.
				129 D	Global file pointer not checked within this procedure.
				131 D	Global denominator not checked within this procedure.
				135 D	Pointer assigned to NULL may be dereferenced.
				136 D	Global pointer assigned to NULL may be dereferenced.
				137 D	Parameter used as denominator not checked before use.
				248 S	Divide by zero in preprocessor directive.
				493 S	Numeric overflow.
				629 S	Divide by zero found.
				80 X	Divide by zero found.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
D.4.2	Advisory	All usage of assembly language should be documented		17 S	Code insert found.
D.4.3	Required	Assembly language shall be encapsulated and isolated		88 S	Procedure is not pure assembler.
D.4.4	Advisory	Sections of code should not be 'commented out'		302 S	Comment possibly contains code.
D.4.5	Advisory	Identifiers in the same namespace with overlapping visibility should be typographically unambiguous	This directive allows a static analysis tool the choice of what is considered ambiguous. The analysis checks for the following: change in case, presence/absence of underscore, Ov0, Iv1, Iv, Iv1, Sv5, Zv2, nvh and Bv8. The analysis highlights all typographically ambiguous identifiers in the project regardless of namespace or scope. The user must then manually check whether they are in the same namespace with overlapping visibility.	217 S	Names only differ by case.
				67 X	Identifier is typographically ambiguous.
D.4.6	Advisory	typedefs that indicate size and signedness should be used in place of the basic numerical types	The analysis highlights all uses of basic types. This includes their use in bit-fields and as the parameter/return types of main, which are allowed by this MISRA rule.	90 S	Basic type declaration used. Modifiers: 219 = 1 462 = 0 (Default)
				495 S	Typedef name has no size indication. Modifiers: 392 = 1
D.4.7	Required	If a function returns error information, then that error information shall be tested	The analysis highlights all function return values that are not used. The user must then manually check for which unused function returns include error information.	91 D	Function return value potentially unused.
				124 D	Var set by std lib func return not checked before use.
				130 D	Global set by std lib func return not checked before use.
D.4.8	Advisory	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden		104 D	Structure implementation not hidden.
D.4.9	Advisory	A function should be used in preference to a function-like macro where they are interchangeable	The analysis highlights all uses of function-like macros. The user must then manually check whether they can be replaced by a function without causing a compiler error.	340 S	Use of function like macro.
D.4.10	Required	Precautions shall be taken in order to prevent the contents of a header file being included more than once	The documentation for 243 S describes the forms accepted by the tool.	243 S	Included file not protected with #define.
D.4.11	Required	The validity of values passed to library functions shall be checked			
D.4.12	Required	Dynamic memory allocation shall not be used	The analysis is configured to check for malloc, calloc, realloc and free. Other functions may be added to cvals.dat in a similar manner.	44 S	Use of banned function, type or variable.
D.4.13	Advisory	Functions which are designed to provide operations on a resource should be called in an appropriate sequence	Some checks are performed by other MISRA guidelines. For example the use of close after open.		
D.4.14	Required	The validity of values received from external sources shall be checked	This directive cannot be completely checked using static analysis. Dynamic analysis and coverage analysis can be used to minimise run-time errors. Errors such as exceeding array bounds are covered by more specific rules. The checks on numeric overflow/underflow are restricted to those cases where the value of a variable can be determined statically.	43 D	Divide by zero found.
				45 D	Pointer not checked for null before use.
				85 D	Filename not verified before fopen.
				86 D	User input not checked before use.
				123 D	File pointer not checked for null before use.
				127 D	Local or member denominator not checked before use.
				128 D	Global pointer not checked within this procedure.
				129 D	Global file pointer not checked within this procedure.
				131 D	Global denominator not checked within this procedure.
				248 S	Divide by zero in preprocessor directive.
				493 S	Numeric overflow.
				488 S	Value outside range of underlying type. Modifiers: 535 = 0 (Default)
				629 S	Divide by zero found.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				80 X	Divide by zero found.
D.4.15	Required	Evaluation of floating-point expressions shall not lead to the undetected generation of infinities and NaNs			
D.5.1	Required	There shall be no data races between threads.			
D.5.2	Required	There shall be no deadlocks between threads.			
D.5.3	Required	There shall be no dynamic thread creation.		48 V	Thread created in a thread entry point.
R.1.1	Required	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits	The analysis should be performed after compilation by the user's compiler. The standards mapped to this rule protect from the following constraint errors, which have been known to go undetected by some compilers. 21 S 6.5.2.2 145 S 6.10 323 S 6.8.4.2 345 S 6.5.7 6.5.10-12 387 S 6.7.2.2 404 S 6.7.8 481 S 6.7.2.1 580 S 6.10.3 615 S 6.5.15 646 S 6.7.8	21 S	Number of parameters does not match.
				145 S	#if has invalid expression.
				323 S	Switch has more than one default case.
				345 S	Bit operator with floating point operand.
				387 S	Enum init not integer-constant-expression.
				404 S	Array initialisation has too many items. Modifiers :399 = 1
				481 S	Array with no bounds in struct.
				580 S	Macro redefinition without using #undef.
				615 S	Conditional operator has incompatible types.
				646 S	Struct initialisation has too many items.
R.1.2	Advisory	Language extensions should not be used	Single line comments are permitted if using C99. 110 S and 632 S are disabled by default if the 350 modifier is set to 1 by the model.	110 S	Use of single line comment //.
				143 S	Curly brackets used in expression.
				293 S	Non ANSI/ISO construct used.
				632 S	Use of // comment in pre-processor directive or macro defn.
			The mappings in this rule refer to the undefined features that are not covered by more specific rule. C90_4 608S C90_11 176 S C90_13 450 S C90_49 324 S C99_2 412 S, 5 Q C99_4 118 S C99_5 113 S C99_20 64 S, 65 S C99_23 576 S C99_35 642 S C99_36 21 S C99_56 465 S, 482 S, 497 S C99_57 296 S C99_58 497 S C99_59 649 S, 650 S C99_61 582 S, 587 S C99_73 576 S C99_84 335 S, 336 S C99_89 573 S C99_94 71 X C99_124 89 D C99_125 87 D C99_136 44 S C99_141 84 D C99_142 590 S C99_143 84 D C99_145 486 S, 589 S C99_162 83 D C99_163 84 D C99_166 82 D C99_169 645 S C99_180 487 S, 489 S, 66 X, 70 X, 71 X, 79 X	82 D	fsetpos values not generated by fgetpos.
				83 D	Potentially repeated call to ungetc.
				84 D	No fseek or flush before I/O.
				87 D	Illegal shared object in signal handler.
				89 D	Illegal use of raise in signal handler.
				5 Q	File does not end with new line.
				21 S	Number of parameters does not match.
				44 S	Use of banned function, type or variable.
				64 S	Void procedure used in expression. Modifiers : 218 = 1
				65 S	Void variable passed as parameter.
				113 S	Non standard character in source. Modifiers : 408 = 1 449 = 0 (Default)
				118 S	main must be int (void) or int (int,char*[]).
				176 S	Non standard escape sequence in source.
				296 S	Function declared at block scope.
				324 S	Macro call has wrong number of parameters.
				335 S	Operator defined contains illegal items.
				336 S	#if expansion contains define operator.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.1.3	Required	There shall be no occurrence of undefined or critical unspecified behaviour		412 S	Undefined behaviour, \ before E-O-F.
				450 S	Wide string and string concatenated.
				465 S	Struct/union not completely specified.
				482 S	Incomplete structure referenced.
				486 S	Incorrect number of formats in output function.
				487 S	Insufficient space allocated.
				489 S	Insufficient space for operation.
				497 S	Type is incomplete in translation unit.
				573 S	Macro concatenation of uni char names.
				576 S	Function pointer is of wrong type.
				582 S	const object reassigned.
				587 S	Const local variable not immediately initialised.
				589 S	Format is not appropriate type.
				590 S	Mode fault in fopen.
				608 S	Use of explicitly undefined language feature.
				642 S	Function return type with array field.
				645 S	realloc ptr type does not match target type.
				649 S	Use of unallocated flexible array.
				650 S	Flexible array copy ignores last member.
				697 S	Pointer converted to different pointer type. Modifiers : 441 = 1
R.1.4	Required	Emergent language features shall not be used	This rule bans the following: * stdatomic.h, thread.h * Bounds Checking functions in Annex K * _Thread_local keywords	44 S	Use of banned function, type or variable.
				130 S	Included file is not permitted.
				15 V	Use of banned keyword.
R.1.5	Required	Obsolescent language features shall not be used		63 S	Empty parameter list to procedure/function.
				68 S	#undef used.
				135 S	Parameter list is KR.
				575 S	Linkage differs from previous declaration.
				678 S	Type specifier before non-type specifier.
				28 D	Potentially infinite loop found.
				76 D	Procedure is not called or referenced in code analysed.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.2.1	Required	A project shall not contain unreachable code	This MISRA rule is undecidable. The analysis will detect code that is never reached regardless of the input values. This includes not reaching the end of a function if an infinite loop is detected	1 J	Unreachable Code found.
				3 J	All internal linkage calls unreachable.
				35 S	Static procedure is not explicitly called in code analysed.
				631 S	Declaration not reachable.
R.2.2	Required	A project shall not contain dead code	This MISRA rule is undecidable. D151 highlights individual instances of variables involved in DD anomalies, which includes variables that are re-assigned down only one path. An exception is made for variables which are not used between initialisation and reassignment, as this is not considered to be dead code by the MISRA-C:2012 guidelines.	65 D	Void function has no side effects.
				105 D	DU anomaly dead code, var value is unused on all paths. Modifiers : 391 = 1
				151 D	DD data flow anomaly found. Modifiers :391 = 1
				57 S	Statement with no side effect.
R.2.3	Advisory	A project should not contain unused type declarations	The analysis highlights unused types in each translation unit. The user must then manually check whether this message is issued for all translation units and hence whether the type is not used in the project. By default, the analysis is configured to not check types declared in header files. This can be altered as described in the documentation for 413 S.	413 S	User type declared but not used in code analysed. Modifiers : 284 = 1
R.2.4	Advisory	A project should not contain unused tag declarations	The analysis permits the presence of a tag if the associated type is used. The analysis highlights unused tags in each translation unit. The user must then manually check whether this message is issued for all translation units and hence whether the tag is not used in the project.	413 S	User type declared but not used in code analysed. Modifiers : 284 = 1
R.2.5	Advisory	A project should not contain unused macro definitions	The analysis highlights unused macros in each translation unit. The user must then manually check whether this message is issued for all translation units and hence whether the macro is not used in the project.	628 S	Macro not used in translation unit.
R.2.6	Advisory	A function should not contain unused label declarations		610 S	Label is unused.
R.2.7	Advisory	A function should not contain unused parameters		1 D	Unused procedure parameter.
				15 D	Unused procedural parameter.
R.2.8	Required	A project should not contain unused object definitions.		94 D	Named variable declared but not used in code.
				413 S	User type declared but not used in code analysed. Modifiers : 284 = 1
R.3.1	Required	The character sequences /* and // shall not be used within a comment		119 S	Nested comment found. Modifiers: 413 = 0 (Default) 434 = 0 (Default)
R.3.2	Required	Line-splicing shall not be used in // comments		611 S	Line splice used in // comment. Modifiers : 402 = 1
R.4.1	Required	Octal and hexadecimal escape sequences shall be terminated		176 S	Non standard escape sequence in source.
R.4.2	Advisory	Trigraphs should not be used		81 S	Use of trigraph.
R.5.1	Required	External identifiers shall be distinct		17 D	Identifier not unique within *** characters.
				61 X	Identifier match in *** chars.
R.5.2	Required	Identifiers declared in the same scope and name space shall be distinct		17 D	Identifier not unique within *** characters.
				61 X	Identifier match in *** chars.
R.5.3	Required	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope	This rule overlaps with the rules 5.6, 5.7, 5.8 and 5.9. Where a standard could be mapped to more than one rule, it is only mapped to the more specific rule.	17 D	Identifier not unique within *** characters.
				92 S	Duplicate use of a name in an enumeration.
				128 S	Parameter has same name as global variable.
				131 S	Name reused in inner scope.
				61 X	Identifier match in *** chars.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.5.4	Required	Macro identifiers shall be distinct		384 S	Identifier matches macro name up to max significant chars. Modifiers : 350 = 1
				622 S	Macro parameters are not unique within limits. Modifiers : 350 = 1
				61 X	Identifier match in *** chars.
R.5.5	Required	Identifiers shall be distinct from macro names		383 S	Identifier name matches macro name.
				384 S	Identifier matches macro name up to max significant chars. Modifiers : 350 = 1
				12 X	Identifier reuse: tag vs macro.
				21 X	Identifier reuse: typedef vs macro.
				34 X	Identifier reuse: proc vs macro.
				37 X	Identifier reuse: persistent var vs macro.
				47 X	Identifier reuse: component vs macro.
				48 X	Identifier reuse: label vs macro (MR).
				50 X	Identifier reuse: var vs macro.
				53 X	Identifier reuse: proc param vs macro.
				57 X	Identifier reuse: macro vs enum constant.
				61 X	Identifier match in *** chars.
R.5.6	Required	A typedef name shall be a unique identifier		112 S	Typedef name redeclared. Modifiers : 417 = 1
				374 S	Name conflict with typedef. Modifiers : 191 = 1
				11 X	Identifier reuse: tag vs typedef.
				16 X	Identifier reuse: typedef vs variable.
				17 X	Identifier reuse: typedef vs label (MR).
				18 X	Identifier reuse: typedef vs typedef.
				19 X	Identifier reuse: typedef vs procedure parameter.
				20 X	Identifier reuse: persistent var vs typedef.
				22 X	Identifier reuse: typedef vs component.
				23 X	Identifier reuse: typedef vs enum constant.
				24 X	Identifier reuse: typedef vs procedure.
R.5.7	Required	A tag name shall be a unique identifier		325 S	Inconsistent use of tag.
				4 X	Identifier reuse: struct/union tag repeated.
				5 X	Identifier reuse: struct vs union.
				6 X	Identifier reuse: struct/union tag vs enum tag.
				7 X	Identifier reuse: tag vs procedure.
				8 X	Identifier reuse: tag vs procedure parameter.
				9 X	Identifier reuse: tag vs variable.
				10 X	Identifier reuse: tag vs label (MR).
				11 X	Identifier reuse: tag vs typedef.
				13 X	Identifier reuse: tag vs component.
				14 X	Identifier reuse: tag vs enum constant.
				15 X	Identifier reuse: persistent var vs tag.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.5.8	Required	Identifiers that define objects or functions with external linkage shall be unique		1 S	Procedure name reused.
				7 X	Identifier reuse: tag vs procedure.
				15 X	Identifier reuse: persistent var vs tag.
				20 X	Identifier reuse: persistent var vs typedef.
				24 X	Identifier reuse: typedef vs procedure.
				25 X	Identifier reuse: procedure vs procedure param.
				26 X	Identifier reuse: persistent var vs label (MR).
				27 X	Identifier reuse: persist var vs persist var.
				28 X	Identifier reuse: persistent var vs var.
				29 X	Identifier reuse: persistent var vs procedure.
				30 X	Identifier reuse: persistent var vs proc param.
				31 X	Identifier reuse: procedure vs procedure.
				32 X	Identifier reuse: procedure vs var.
				33 X	Identifier reuse: procedure vs label (MR).
				35 X	Identifier reuse: proc vs component.
				36 X	Identifier reuse: proc vs enum constant.
				38 X	Identifier reuse: persistent var vs component.
				39 X	Identifier reuse: persistent var vs enum constant.
R.5.9	Advisory	Identifiers that define objects or functions with internal linkage should be unique	The analysis highlights all re-use of identifiers with static storage duration. Users will have to manually confirm that the re-use is not against a permitted locally declared static identifier. Such identifiers are defined as having no linkage.	1 S	Procedure name reused.
				7 X	Identifier reuse: tag vs procedure.
				15 X	Identifier reuse: persistent var vs tag.
				20 X	Identifier reuse: persistent var vs typedef.
				24 X	Identifier reuse: typedef vs procedure.
				25 X	Identifier reuse: procedure vs procedure param.
				26 X	Identifier reuse: persistent var vs label (MR).
				27 X	Identifier reuse: persist var vs persist var.
				28 X	Identifier reuse: persistent var vs var.
				29 X	Identifier reuse: persistent var vs procedure.
				30 X	Identifier reuse: persistent var vs proc param.
				31 X	Identifier reuse: procedure vs procedure.
				32 X	Identifier reuse: procedure vs var.
				33 X	Identifier reuse: procedure vs label (MR).
				35 X	Identifier reuse: proc vs component.
				36 X	Identifier reuse: proc vs enum constant.
				38 X	Identifier reuse: persistent var vs component.
				39 X	Identifier reuse: persistent var vs enum constant.
R.6.1	Required	Bit-fields shall only be declared with an appropriate type	73 S should be used when running with C90 and 520 S when running with C99. The choice of standard is determined by the setting of modifier 350. By default, the C99 setting is used. C99 permits the use of other bit-field types, but they are implementation dependent. The analysis highlights the use of such types.	73 S	Bit field not signed or unsigned int. Modifiers : 234 = 0 (Default) 331 = 0 (Default) 350 = 1
				520 S	Bit field is not bool or explicit integral. Modifiers : 331 = 0 (Default) 350 = 1
R.6.2	Required	Single-bit named bit fields shall not be of a signed type		72 S	Signed bit field less than 2 bits wide. Modifiers : 416 = 1
R.6.3	Required	A bit field shall not be declared as a member of a union		710 S	Use of bit field in union declaration.
R.7.1	Required	Octal constants shall not be used		83 S	Octal number found.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.7.2	Required	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type		550 S	Unsuffix hex or octal is unsigned, add U.
R.7.3	Required	The lowercase character "l" shall not be used in a literal suffix		252 S	Lower case suffix to literal number. Modifiers : 418 = 1
R.7.4	Required	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"		157 S	Modification of string literal. Modifiers : 438 = 1
				623 S	String assigned to non const object. Modifiers : 119 = 0 (Default)
R.7.5	Mandatory	The argument of an integer-constant macro shall have an appropriate form		711 S	Inappropriate argument to integer-constant macro.
R.7.6	Required	The small integer variants of the minimum-width integer constant macros shall not be used.		712 S	Size of integer-constant macro is less than integer.
R.8.1	Required	Types shall be explicitly specified		20 S	Parameter not declared explicitly.
				135 S	Parameter list is KR.
				326 S	Declaration is missing type.
R.8.2	Required	Function types shall be in prototype form with named parameters		37 S	Procedure parameter has a type but no identifier.
				63 S	Empty parameter list to procedure/function.
				135 S	Parameter list is KR.
R.8.3	Required	All declarations of an object or function shall use the same names and type qualifiers	The analysis does not currently include a check on type qualifiers when comparing function declarations.	36 D	Prototype and definition name mismatch.
				63 X	Function prototype/defn param type mismatch (MR).
R.8.4	Required	A compatible declaration shall be visible when an object or function with external linkage is defined		36 D	Prototype and definition name mismatch.
				106 D	No prototype for non-static function.
				102 S	Function and prototype return inconsistent (MR). Modifiers : 331 = 0 (Default)
				103 S	Function and prototype param inconsistent (MR).
				1 X	Declaration types do not match across a system.
				62 X	Function prototype/defn return type mismatch (MR).
				63 X	Function prototype/defn param type mismatch (MR).
R.8.5	Required	An external object or function shall be declared once in one and only one file		60 D	External object should be declared only once.
				110 D	More than one prototype for same function.
				172 S	Variable declared multiply.
R.8.6	Required	An identifier with external linkage shall have exactly one external definition		26 D	Variable should be defined once in only one file.
				33 D	No real declaration for external variable.
				34 D	Procedure name re-used in different files.
				63 D	No definition in system for prototyped procedure.
R.8.7	Advisory	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit		27 D	Variable should be declared static.
				61 D	Procedure should be declared static.
R.8.8	Required	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage		27 D	Variable should be declared static.
				61 D	Procedure should be declared static.
				461 S	Identifier with ambiguous linkage.
				553 S	Function and proto should both be static.
				575 S	Linkage differs from previous declaration.
R.8.9	Advisory	An object should be declared at block scope if its identifier only appears in a single function		25 D	Scope of variable could be reduced.
R.8.10	Required	An inline function shall be declared with the static storage class		612 S	inline function should be declared static.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.8.11	Advisory	When an array with external linkage is declared, its size should be explicitly specified		127 S	Array has no bounds specified. Modifiers : 285 = 0 (Default) 286 = 1 426 = 1
R.8.12	Required	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique		630 S	Duplicated enumeration value.
R.8.13	Advisory	A pointer should point to a const-qualified type whenever possible	This MISRA rule is undecidable. The analysis does not check array parameters declared with a size.	120 D	Pointer param should be declared pointer to const.
R.8.14	Required	The restrict type qualifier shall not be used		613 S	Use of restrict keyword. Modifiers : 350 = 1
R.8.15	Required	All declarations of an object with an explicit alignment specification shall specify the same alignment		40 V	Conflicting alignment specifiers in variable declarations.
R.8.16	Advisory	The alignment specification of zero should not appear in an object declaration		37 V	Variable declaration has alignment specifier of 0.
R.8.17	Advisory	At most one explicit alignment specifier should appear in an object declaration		38 V	Variable declaration has multiple alignment specifiers.
R.9.1	Mandatory	The value of an object with automatic storage duration shall not be read before it has been set	This MISRA rule is undecidable.	53 D	Attempt to use uninitialised pointer.
				69 D	UR anomaly, variable used before assignment.
				631 S	Declaration not reachable.
				652 S	Object created by malloc used before initialisation.
R.9.2	Required	The initializer for an aggregate or union shall be enclosed in braces	The analysis highlights all arrays that are not fully initialised or are initialised with {0}. The user must then check manually whether the MISRA rule is satisfied or not.	105 S	Initialisation brace { } fault.
R.9.3	Required	Arrays shall not be partially initialized		397 S	Array initialisation has insufficient items. Modifiers : 415 = 1 445 = 450 = 455 = 0 (Default) 1
R.9.4	Required	An element of an object shall not be initialised more than once	The analysis highlights all uses of initialisers which contains both both positional and designational initialisers. The user must then manually check whether the MISRA rule is satisfied or not.	620 S	Initialisation designator duplicated.
				627 S	Initialiser both positional and designational.
R.9.5	Required	Where designated initialisers are used to initialize an array object the size of the array shall be specified explicitly		127 S	Array has no bounds specified. Modifiers : 285 = 0 (Default) 286 = 1 426 = 1
R.9.6	Required	An initializer using chained designators shall not contain initializers without designators.			
R.9.7	Required	Atomic objects shall be appropriately initialized before being accessed.		51 V	Atomic object not initialised with atomic_init.
R.10.1	Required	Operands shall not be of an inappropriate essential type		50 S	Use of shift operator on signed type.
				52 S	Unsigned expression negated.
				109 S	Array subscript is not integral.
				114 S	Expression is not Boolean. Modifiers : 386 = 1 (Default) 526 = 1
				120 S	Use of bit operator on signed type. Modifiers : 457 = 0 (Default)
				123 S	Use of underlying enum representation value. Modifiers : 191 = 1 428 = 1 391 = 1
				136 S	Bit operator with boolean operand.
				249 S	Shift operator with boolean operand.
				329 S	Operation not appropriate to plain char. Modifiers : 191 = 1 331 = 0 (Default) 391 = 1
				345 S	Bit operator with floating point operand.
				389 S	Bool value incremented/decremented.
				403 S	Negative (or potentially negative) shift.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				506 S	Use of boolean with relational operator.
				706 S	++ or -- with _Complex operand.
				707 S	Relational operator with _Complex operand.
				708 S	Equality operator with floating point operand.
				710 S	Use of bit field in union declaration.
R.10.2	Required	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations		329 S	Operation not appropriate to plain char. Modifiers : 191 = 1 331 = 0 (Default) 391 = 1
				433 S	Type conversion without cast. Modifiers : 374 = 0 (Default) 391 = 1 431 = 1 428 = 1 470 = 0 (Default) 386 = 1 (Default) 549 = 1
R.10.3	Required	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category		101 S	Function return type inconsistent. Modifiers : 374 = 0 (Default) 428 = 1 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default) 427 = 1
				104 S	Struct field initialisation incorrect. Modifiers : 358 = 0 (Default) 430 = 0 (Default) 431 = 1 432 = 0 (Default) 453 = 1 470 = 0 (Default) 427 = 1 428 = 1
				123 S	Use of underlying enum representation value. Modifiers : 191 = 1 428 = 1
				276 S	Case is not part of switch enumeration.
				331 S	Literal value requires a U suffix. Modifiers : 309 = 0 (Default) 358 = 0 (Default) 516 = 1
				411 S	Inappropriate value assigned to enum.
				433 S	Type conversion without cast. Modifiers : 374 = 0 (Default) 391 = 1 431 = 1 428 = 1 470 = 0 (Default) 386 = 1 (Default) 549 = 1
				434 S	Signed/unsigned conversion without cast. Modifiers : 374 = 0 (Default) 358 = 0 (Default) 427 = 1 428 = 1 429 = 1 430 = 0 (Default) 470 = 0 (Default)
				435 S	Float/integer conversion without cast. Modifiers : 374 = 0 (Default) 432 = 0 (Default) 453 = 1
				445 S	Narrower float conversion without cast. Modifiers : 374 = 0 (Default)
				446 S	Narrower int conversion without cast. Modifiers : 374 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
				488 S	Value outside range of underlying type.
				709 S	_Complex assigned to float without cast.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.10.4	Required	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category		107 S	Type mismatch in ternary expression. Modifiers : 428 = 1 429 = 1 470 = 0 (Default) 525 = 0 (Default)
				123 S	Use of underlying enum representation value. Modifiers : 191 = 1 428 = 1
				331 S	Literal value requires a U suffix. Modifiers : 309 = 0 (Default) 358 = 0 (Default)
				433 S	Type conversion without cast. Modifiers : 374 = 0 (Default) 391 = 1 431 = 1 428 = 1 470 = 0 (Default) 386 = 1 (Default) 549 = 1
				434 S	Signed/unsigned conversion without cast. Modifiers : 374 = 0 (Default) 358 = 0 (Default) 427 = 1 428 = 1 429 = 1 430 = 0 (Default) 470 = 0 (Default)
				435 S	Float/integer conversion without cast. Modifiers : 374 = 0 (Default) 432 = 0 (Default) 453 = 1
				488 S	Value outside range of underlying type. Modifiers : 535 = 0 (Default)
R.10.5	Advisory	The value of an expression should not be cast to an inappropriate essential type		123 S	Use of underlying enum representation value. Modifiers : 191 = 1 428 = 1 391 = 1
				441 S	Float cast to non-float. Modifiers : 502 = 1 503 = 0 (Default) 504 = 0 (Default) 505 = 0 (Default) 506 = 0 (Default) 529 = 0 (Default)
				442 S	Signed integral type cast to unsigned. Modifiers : 470 = 0 (Default) 507 = 1 529 = 0 (Default)
				443 S	Unsigned integral type cast to signed. Modifiers : 508 = 1 529 = 0 (Default)
				444 S	Integral type cast to non-integral. Modifiers : 509 = 1 510 = 0 (Default) 511 = 0 (Default) 512 = 0 (Default) 513 = 1 529 = 0 (Default)
				695 S	Char expression cast to non-char type. Modifiers : 514 = 1 529 = 0 (Default)
R.10.6	Required	The value of a composite expression shall not be assigned to an object with wider essential type		451 S	No cast for widening complex float expression (MR). Modifiers : 191 = 1 529 = 0 (Default)
				452 S	No cast for widening complex int expression (MR). Modifiers : 191 = 1 529 = 0 (Default)
R.10.7	Required	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		451 S	No cast for widening complex float expression (MR). Modifiers : 191 = 1 529 = 0 (Default)
				452 S	No cast for widening complex int expression (MR). Modifiers : 191 = 1 529 = 0 (Default)
				332 S	Widening cast on complex integer expression. Modifiers : 529 = 0 (Default)
				333 S	Widening cast on complex float expression. Modifiers : 529 = 0 (Default)

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.10.8	Required	The value of a composite expression shall not be cast to a different essential type category or a wider essential type		441 S	Float cast to non-float. Modifiers : 502 = 1 503 = 0 (Default) 504 = 0 (Default) 505 = 0 (Default) 506 = 0 (Default) 529 = 0 (Default)
				442 S	Signed integral type cast to unsigned. Modifiers : 470 = 0 (Default) 507 = 1 529 = 0 (Default)
				443 S	Unsigned integral type cast to signed. Modifiers : 508 = 1 529 = 0 (Default)
				444 S	Integral type cast to non-integral. Modifiers : 509 = 1 510 = 0 (Default) 511 = 0 (Default) 512 = 0 (Default) 513 = 1 529 = 0 (Default)
R.11.1	Required	Conversions shall not be performed between a pointer to a function and any other type		606 S	Cast involving function pointer. Modifiers : 398 = 0 (Default) 433 = 1 439 = 1
R.11.2	Required	Conversions shall not be performed between a pointer to incomplete and any other type		439 S	Cast from pointer to integral type. Modifiers : 548 = 1
				440 S	Cast from integral type to pointer. Modifiers : 397 = 0 (Default) 548 = 1
				554 S	Cast to an unrelated type. Modifiers : 439 = 1 440 = 1
R.11.3	Required	A conversion shall not be performed between a pointer to object type and a pointer to a different object type		554 S	Cast to an unrelated type. Modifiers : 439 = 1 440 = 1
R.11.4	Advisory	A conversion should not be performed between a pointer to object and an integer type		439 S	Cast from pointer to integral type. Modifiers : 548 = 1
				440 S	Cast from integral type to pointer. Modifiers : 397 = 0 (Default) 548 = 1
R.11.5	Advisory	A conversion should not be performed from pointer to void into pointer to object		540 S	Cast from pointer to void to pointer. Modifiers : 381 = 0 (Default)
				697 S	Pointer converted to different pointer type. Modifiers : 441 = 1
R.11.6	Required	A cast shall not be performed between pointer to void and an arithmetic type		439 S	Cast from pointer to integral type. Modifiers : 548 = 1
				440 S	Cast from integral type to pointer. Modifiers : 397 = 0 (Default) 548 = 1
				635 S	Cast from pointer to float type.
				636 S	Cast from float type to pointer.
R.11.7	Required	A cast shall not be performed between pointer to object and a non-integer arithmetic type		439 S	Cast from pointer to integral type. Modifiers : 548 = 1
				440 S	Cast from integral type to pointer. Modifiers : 397 = 0 (Default) 548 = 1
				635 S	Cast from pointer to float type.
				636 S	Cast from float type to pointer.
R.11.8	Required	A conversion shall not remove any const or volatile qualification from the type pointed to by a pointer		203 S	Cast on a constant value. Modifiers : 390 = 1
				344 S	Cast on volatile value. Modifiers : 395 = 1
R.11.9	Required	The macro NULL shall be the only permitted form of integer null pointer constant		531 S	Literal zero used in pointer context.
R.11.10	Required	The _Atomic qualifier shall not be applied to the incomplete type void.		47 V	_Atomic qualifier applied to an incomplete type void.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.12.1	Advisory	The precedence of operators within expressions should be made explicit		49 S	Logical conjunctions need brackets. Modifiers :206 = 1
				361 S	Expression needs brackets. Modifiers :119 = 0 (Default) 264 = 0 (Default) 414 = 1 420 = 1 421 = 1
R.12.2	Required	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	This MISRA rule is undecidable	51 S	Shifting value too far. Modifiers :520 = 1
				403 S	Negative (or potentially negative) shift.
R.12.3	Advisory	The comma operator should not be used		53 S	Use of comma operator. Modifiers :263 = 1
R.12.4	Advisory	Evaluation of constant expressions should not lead to unsigned integer wrap-around		493 S	Numeric overflow.
				488 S	Value outside range of underlying type. Modifiers :535 = 0 (Default)
R.12.5	Mandatory	The sizeof operator shall not have an operand which is a function parameter declared as "array of type"		401 S	Use of sizeof on an array parameter.
R.12.6	Required	Structure and union members of atomic objects shall not be directly accessed.		52 V	Direct access of a member of an atomic structure or union.
R.13.1	Required	Initialiser lists shall not contain persistent side effects	This MISRA rule is undecidable.	35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				9 S	Assignment operation in expression.
				30 S	Deprecated usage of ++ or -- operators found.
				132 S	Assignment operator in boolean expression.
				134 S	Volatile variable in complex expression.
R.13.2	Required	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders and shall be independent from thread interleaving	This MISRA rule is undecidable. In particular the mapping of S9 should be treated as an informational message, for which further investigation should occur manually.	35 D	Expression has side effects.
				72 D	Potential side effect problem in expression.
				1 Q	Call has execution order dependant side effects.
				9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				30 S	Deprecated usage of ++ or -- operators found. Modifiers :122 = 1
				134 S	Volatile variable in complex expression. Modifiers :460 = 1 461 = 1
R.13.3	Advisory	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	The analysis highlights all uses of ++ and -- in expressions when combined with other arithmetic operators. This is a stricter check than required by the MISRA C:2012 rule.	30 S	Deprecated usage of ++ or -- operators found. Modifiers :122 = 1
R.13.4	Advisory	The result of an assignment operator should not be used		9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				132 S	Assignment operator in boolean expression.
R.13.5	Required	The right hand operand of a logical && or operator shall not contain persistent side effects	This MISRA rule is undecidable	35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				406 S	Use of ++ or -- on RHS of && or operator.
				408 S	Volatile variable accessed on RHS of && or .
R.13.6	Required	The operand of the sizeof operator shall not contain any expression which has potential side effects		54 S	Sizeof operator with side effects.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.14.1	Required	A loop counter shall not have essentially floating type	The determination of a loop counter as defined in the MISRA document is statically undecidable. The analysis definition of loop counter is restricted to for loops that satisfy rule 14.2	39 S	Unsuitable type for loop variable.
R.14.2	Required	A for loop shall be well-formed	This MISRA rule is undecidable. The analysis highlights all uses of function calls in the second clause. The user must then manually check that such calls do not have persistent side-effects. The analysis definition of a loop counter is stricter than that of MISRA and does not permit the use of structure members as loop counters.	55 D	Modification of loop counter in loop body.
				270 S	For loop initialisation is not simple. <i>Modifiers</i> :249 = 1
				271 S	For loop incrementation is not simple. <i>Modifiers</i> :331 = 0 (Default)
				429 S	Empty middle expression in for loop.
				430 S	Inconsistent usage of loop control variable.
				581 S	Loop conditions are independent.
R.14.3	Required	Controlling expressions shall not be invariant	This MISRA rule is undecidable.	139 S	Construct leads to infeasible code.
				140 S	Infeasible loop condition found.
R.14.4	Required	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type		114 S	Expression is not Boolean. <i>Modifiers</i> :386 = 1 (Default) 528 = 1
R.15.1	Advisory	The goto statement should not be used		13 S	goto detected.
R.15.2	Required	The goto statement shall jump to a label declared later in the same function		509 S	goto label is backwards.
R.15.3	Required	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement		511 S	Jump into nested block.
R.15.4	Advisory	There should be no more than one break or goto statement used to terminate any iteration statement		409 S	More than one break or goto statement in loop.
R.15.5	Advisory	A function should have a single point of exit at the end		7 C	Procedure has more than one exit point.
R.15.6	Required	The body of an iteration-statement or a selection-statement shall be a compound statement		11 S	No brackets to loop body (added by Testbed).
				12 S	No brackets to then/else (added by Testbed). <i>Modifiers</i> :bend F
				428 S	No {} for switch (added by Testbed).
R.15.7	Required	All if . . else if constructs shall be terminated with an else statement		59 S	Else alternative missing in if. <i>Modifiers</i> :530 = 0 (Default)
				477 S	Empty else clause following else if.
R.16.1	Required	All switch statements shall be well-formed		385 S	MISRA switch statement syntax violation.
R.16.2	Required	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement		245 S	Case statement in nested block.
R.16.3	Required	An unconditional break statement shall terminate every switch-clause		62 S	Switch case not terminated with break. <i>Modifiers</i> :283 = 0 (Default) 384 = c++ : 0 (Default) 551 = 0 (Default)
R.16.4	Required	Every switch statement shall have a default label		48 S	No default case in switch statement. <i>Modifiers</i> :252 = 0 (Default) 233 = 0 (Default)
				410 S	Switch empty default has no comment (MR).
R.16.5	Required	A default label shall appear as either the first or the last switch label of a switch statement		322 S	Default is not last case of switch. <i>Modifiers</i> :419 = 1
R.16.6	Required	Every switch statement shall have at least two switch-clauses		60 S	Empty switch statement.
				61 S	Switch contains default only. <i>Modifiers</i> :437 = 1

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.16.7	Required	A switch-expression shall not have essentially Boolean type		121 S	Use of boolean expression in switch.
R.17.1	Required	The standard header file <stdarg.h> shall not be used		44 S	Use of banned function, type or variable.
				130 S	Included file is not permitted.
R.17.2	Required	Functions shall not call themselves, either directly or indirectly	This MISRA rule is undecidable.	6 D	Recursion in procedure calls found.
				1 U	Inter-file recursion found.
R.17.3	Mandatory	A function shall not be declared implicitly		496 S	Function call with no prior declaration.
R.17.4	Mandatory	All exit paths from a function with non-void return type shall have an explicit return statement with an expression		2 D	Function does not return a value on all paths.
				36 S	Function has no return statement. Modifiers :553 = 0 (Default)
				66 S	Function with empty return expression.
R.17.5	Required	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	This MISRA rule is undecidable.	64 X	Array bound exceeded at call.
R.17.6	Mandatory	The declaration of an array parameter shall not contain the static keyword between the []		614 S	Use of static keyword in array parameter. Modifiers :350 = 1
R.17.7	Required	The value returned by a function having non-void return type shall be used	382 S will also be output when a function result is used in an operation that it-self is not used. This violates rule 2.2. By default the tool is configured to not output a warning on the printf family of functions. Instructions for enabling the check on printf functions can be found in the 382 S documentation.	382 S	(void) missing for discarded return value. Modifiers :383 = 1
R.17.8	Advisory	A function parameter should not be modified	This MISRA rule is undecidable.	14 D	Attempt to change parameter passed by value.
				149 S	Reference parameter to procedure is reassigned.
R.17.9	Mandatory	A function declared with a _Noreturn function specifier shall not return to its caller		34 V	Function declared with _Noreturn has return statement.
				35 V	Function declared with _Noreturn may return on some paths.
R.17.10	Required	A function declared with a _Noreturn function specifier shall have void return type		32 V	Function declared _Noreturn does not have void return type.
R.17.11	Advisory	A function that never returns should be declared with a _Noreturn function specifier		33 V	Function that does not return requires _Noreturn specifier.
R.17.12	Advisory	A function identifier should only be used with either a preceding &, or with a parenthesised parameter list		36 V	Function identifier used without & or parameter list.
R.17.13	Required	A function type shall not be type qualified.		39 V	Function type is type qualified.
R.18.1	Required	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand	The analysis does not treat the conversion of a pointer to type T to a pointer to char, signed char or unsigned char as creating an array of sizeof(T) elements. The user must then manually check for such permitted uses. This MISRA rule is undecidable.	47 S	Array bound exceeded.
				436 S	Declaration does not specify an array. Modifiers :400 = 0 (Default)
				567 S	Pointer arithmetic is not on array. Modifiers :436 = 1
				692 S	Array index is negative.
				64 X	Array bound exceeded at call.
				68 X	Parameter indexing array too big at call.
				69 X	Global array bound exceeded at use.
				72 X	Parameter indexing array too small at call.
R.18.2	Required	Subtraction between pointers shall only be applied to pointers that address elements of the same array	This MISRA rule is undecidable.	438 S	Pointer subtraction not addressing one array.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.18.3	Required	The relational operators >, >=, < and <= shall not be applied to expressions of pointer type except where they point into the same object	This MISRA rule is undecidable.	437 S	< > <= >= used on different object pointers.
R.18.4	Advisory	The +, -, += and -= operators should not be applied to an expression of pointer type	The analysis highlights all uses of arithmetic on pointers. The user must then manually check for the permitted uses of ++ and pointer subtraction.	87 S	Use of pointer arithmetic. Modifiers :tend = <Default>
				567 S	Pointer arithmetic is not on array. Modifiers :436 = 1
R.18.5	Advisory	Declarations should contain no more than two levels of pointer nesting		80 S	Pointer indirection exceeds 2 levels.
R.18.6	Required	The address of an object with automatic storage or thread-local storage shall not be copied to another object that persists after the first object has ceased to exist		42 D	Local pointer returned in function result.
				77 D	Local structure returned in function result.
				71 S	Pointer assignment to wider scope.
				565 S	Assignment to wider scope.
R.18.7	Required	Flexible array members shall not be declared		481 S	Array with no bounds in struct.
R.18.8	Required	Variable-length arrays shall not be used		621 S	Variable-length array declared.
R.18.9	Required	An object with temporary lifetime shall not undergo array-to-pointer conversion			
R.18.10	Mandatory	Pointers to variably-modified array types shall not be used.		54 V	Parameter is pointer to a variably-modified array type.
R.19.1	Mandatory	An object shall not be assigned or copied to an overlapping object	This MISRA rule is undecidable.	480 S	String function params access same variable.
				545 S	Assignment of overlapping storage.
				647 S	Overlapping data items in memcopy.
R.19.2	Advisory	The union keyword should not be used		74 S	Union declared.
R.20.1	Advisory	#include directives should only be preceded by preprocessor directives or comments		75 S	Executable code before an included file.
				338 S	#include preceded by non preproc directives.
R.20.2	Required	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name		100 S	#include filename is non conformant. :120 = \, (Default) Modifiers
R.20.3	Required	The #include directive shall be followed by either a <filename> or "filename" sequence		427 S	Filename in #include not in < > or " ".
R.20.4	Required	A macro shall not be defined with the same name as a keyword		86 S	Attempt to define reserved word.
				580 S	Macro redefinition without using #undef.
				626 S	#define of keyword. Modifiers :391 = 1
R.20.5	Advisory	#undef should not be used		68 S	#undef used.
				426 S	#undef used in a block.
R.20.6	Required	Tokens that look like a preprocessing directive shall not occur within a macro argument		341 S	Preprocessor construct as macro parameter.
R.20.7	Required	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	The analysis highlights all uses of macro parameters that are not enclosed in parenthesis. The user must then manually check whether adding parenthesis would produce a constraint error. In particular, 361 S highlights those macros that when expanded could have order of precedence issues.	78 S	Macro parameter not in brackets. Modifiers :454 = 0 (Default)
				361 S	Expression needs brackets. Modifiers :119 = 0 (Default) 264 = 0 (Default) 414 = 1 420 = 1 421 = 1
R.20.8	Required	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1		616 S	Preprocessor result not 0 or 1.
R.20.9	Required	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation		337 S	Undefined macro variable in #if.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.20.10	Advisory	The # and ## preprocessor operators should not be used		125 S	Use of ## or # in a macro.
R.20.11	Required	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator		637 S	# operand followed by ##.
R.20.12	Required	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	The analysis highlights occurrences of ## or #. However, the user must manually confirm that the macro parameter used as the operand to ## or # is not used elsewhere in the macro.	125 S	Use of ## or # in a macro.
R.20.13	Required	A line whose first token is # shall be a valid preprocessing directive		147 S	Spurious characters after preprocessor directive.
				342 S	Extra chars after preprocessor directive.
R.20.14	Required	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		126 S	A #if has no #endif in the same file.
				343 S	#else has no #if, etc in the same file.
R.21.1	Required	#define and #undef shall not be used on a reserved identifier or reserved macro name		86 S	Attempt to define reserved word.
				156 S	Use of 'defined' keyword in macro body.
				219 S	User name starts with underscore. Modifiers :412 = 1
R.21.2	Required	A reserved identifier or reserved macro name shall not be declared		218 S	Name is used in standard libraries.
				219 S	User name starts with underscore. Modifiers :412 = 1
R.21.3	Required	The memory allocation and deallocation functions of <stdlib.h> shall not be used		44 S	Use of banned function, type or variable.
R.21.4	Required	The standard header file <setjmp.h> shall not be used		43 S	Use of setjmp/longjmp.
R.21.5	Required	The standard header file <signal.h> shall not be used		130 S	Included file is not permitted.
R.21.6	Required	The Standard Library input/output routines shall not be used.		44 S	Use of banned function, type or variable.
				130 S	Included file is not permitted.
R.21.7	Required	The Standard Library functions atof, atoi, atol and atoll of <stdlib.h> shall not be used		44 S	Use of banned function, type or variable.
R.21.8	Required	The Standard Library termination functions of <stdlib.h> shall not be used		122 S	Use of abort, exit, etc.
R.21.9	Required	The Standard Library functions bsearch and qsort of <stdlib.h> shall not be used		44 S	Use of banned function, type or variable.
R.21.10	Required	The Standard Library time and date routines shall not be used		44 S	Use of banned function, type or variable.
				130 S	Included file is not permitted.
R.21.11	Advisory	The standard header file <tgmath.h> should not be used		130 S	Included file is not permitted.
R.21.12	Required	The standard header file <fenv.h> shall not be used		130 S	Included file is not permitted.
R.21.13	Mandatory	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF		663 S	Invalid value may be passed to function in <ctype.h>.
R.21.14	Required	The Standard Library function memcmp shall not be used to compare null terminated strings		661 S	memcmp used to compare null terminated strings.
R.21.15	Required	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types		655 S	Standard library copy/compare objects have different types.
R.21.16	Required	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		618 S	Use of memcmp between structures.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.21.17	Mandatory	Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters		489 S	Insufficient space for operation.
				600 S	Argument of strlen is unterminated.
				140 D	Copy source parameter not checked before use.
				66 X	Insufficient array space at call.
				70 X	Array has insufficient space.
				71 X	Insufficient space for copy.
R.21.18	Mandatory	The size_t argument passed to any function in <string.h> shall have an appropriate value		489 S	Insufficient space for operation.
				66 X	Insufficient array space at call.
				70 X	Array has insufficient space.
				71 X	Insufficient space for copy.
				79 X	Size mismatch in memcpy/memset.
R.21.19	Mandatory	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		107 D	Attempt to change system call capture string.
R.21.20	Mandatory	The value returned by a call of one of 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		133 D	Pointer from system function used after subsequent call.
R.21.21	Required	The Standard Library function system of <stdlib.h> shall not be used		588 S	Use of system function.
R.21.22	Mandatory	All operand arguments to any type-generic macros declared in <tgmath.h> shall have an appropriate essential type			
R.21.23	Required	All operand arguments to any multi-argument type-generic macros declared in <tgmath.h> shall have the same standard type			
R.21.24	Required	The random number generator functions of <stdlib.h> shall not be used		44 S	Use of banned function, type or variable.
R.21.25	Required	All memory synchronization operations shall be executed in sequentially consistent order.			
R.21.26	Required	The Standard Library function mtx_timedlock() shall only be invoked on mutex objects of appropriate mutex type.		55 V	Call mtx_timedlock for mutex that does not support timeout.
R.22.1	Required	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	This MISRA rule is undecidable.	49 D	File pointer not closed on exit.
				50 D	Memory not freed after last reference.
				75 D	Attempt to open file pointer more than once.
R.22.2	Mandatory	A block of memory shall only be freed if it was allocated by means of a Standard Library function	This MISRA rule is undecidable.	51 D	Attempt to read from freed memory.
				125 D	free called on variable with no allocated space.
				407 S	free used on string.
				483 S	Freed parameter is not heap item.
				484 S	Attempt to use already freed object.
				644 S	realloc ptr does not originate from allocation function.
R.22.3	Required	The same file shall not be open for read and write access at the same time on different streams	This MISRA rule is undecidable.	103 D	File opened both read and write.

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.22.4	Mandatory	There shall be no attempt to write to a stream which has been opened as read-only	This MISRA rule is undecidable.	98 D	Attempt to write to file opened read only.
R.22.5	Mandatory	A pointer to a FILE object shall not be dereferenced	This MISRA rule is undecidable.	591 S	Inappropriate use of file pointer.
R.22.6	Mandatory	The value of a pointer to a FILE shall not be used after the associated stream has been closed	This MISRA rule is undecidable.	48 D	Attempt to write to unopened file.
				113 D	File closed more than once.
R.22.7	Required	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF		662 S	EOF compared with char.
R.22.8	Required	The value of errno shall be set to zero prior to a call to an errno-setting-function		111 D	errno checked without having been set for errno setting fn.
				121 D	errno neither set nor checked for errno setting function.
R.22.9	Required	The value of errno shall be tested against zero after calling an errno-setting-function	The analysis highlights all cases where an errno is not tested after errno-setting-function is called. This includes the situation covered by the exception to this rule.	121 D	errno neither set nor checked for errno setting function.
				122 D	errno not checked after being set for errno setting fn.
				134 D	errno not checked before subsequent function call.
R.22.10	Required	The value of errno shall only be tested when the last function to be called was an errno-setting-function		132 D	errno checked after call to non-errno setting function.
R.22.11	Required	A thread that was previously either joined or detached shall not be subsequently joined nor detached.			
R.22.12	Required	Thread objects, thread synchronization objects, and thread-specific storage pointers shall only be accessed by the appropriate Standard Library functions.		53 V	Direct assignment or check of thread type object.
R.22.13	Required	Thread objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration.		50 V	Thread related objects have automatic storage duration.
R.22.14	Mandatory	Thread synchronization objects shall be initialized before being accessed.			
R.22.15	Required	Thread synchronization objects and thread-specific storage pointers shall not be destroyed until after all threads accessing them have terminated.		56 V	Dependant thread objects destroyed in thread entry points.
R.22.16	Required	All mutex objects locked by a thread shall be explicitly unlocked by the same thread.			
R.22.17	Required	No thread shall unlock a mutex or call <code>cnd_wait()</code> or <code>cnd_timedwait()</code> for a mutex it has not locked before.			
R.22.18	Required	Non-recursive mutexes shall not be recursively locked.			
R.22.19	Required	A condition variable shall be associated with at most one mutex object.			
R.22.20	Required	Thread-specific storage pointers shall be created before being accessed.		49 V	Thread-specific storage pointer created in thread entry.
R.23.1	Advisory	A generic selection should only be expanded from a macro		41 V	Use of generic selection applied directly.
R.23.2	Required	A generic selection that is not expanded from a macro shall not contain potential side effects in the controlling expression		42 V	Control expression in generic selection has side effects. Modifiers :447 = 0 (Default)
R.23.3	Advisory	A generic selection should contain at least one non-default association		43 V	Generic selection contains no non-default association. Modifiers :447 = 0 (Default)

MISRA C:2023 Standards Model Compliance for C / C++					
Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
R.23.4	Required	A generic association shall list an appropriate type		45 V	Generic association is not an appropriate type. Modifiers :447 = 0 (Default)
R.23.5	Advisory	A generic selection should not depend on implicit pointer type conversion		46 V	Generic selection depends on implicit ptr type conversion. Modifiers :447 = 0 (Default)
R.23.6	Required	The controlling expression of a generic selection shall have an essential type that matches its standard type			
R.23.7	Advisory	A generic selection that is expanded from a macro should evaluate its argument only once			
R.23.8	Required	A default association shall appear as either the first or the last association of a generic selection		44 V	Default association not first or last in generic selection. Modifiers :447 = 0 (Default)

General Compliance Notes

Enhanced Enforcement: LDRA checks additional cases to those specified by the mapped rule for enhanced safety and security.
Fully Implemented: LDRA checks all statically checkable aspects of the mapped rule.
Partially Implemented: LDRA checks certain aspects of the rule.

The assessment of whether a rule is fully or partially implemented is based on whether the mapped LDRA standards cover all statically checkable aspects of the rule with a high level of coverage or only cover certain statically checkable aspects of the rule. If a rule is undecidable then this assessment is based on what it is deemed reasonable for a static analysis tool to check.