CERT-C++:2016 Standards Model Summary for 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.

Compliance is measured against
"CERT C++ Secure Coding Standard - 2016 Edition"
2017

Further information is available at http://www.securecoding.cert.org

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Rule	22	36	47	45	12	162
Recommendation	27	59	47	20	29	182
Total	49	95	94	65	41	344

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
ARR30-C	Rule	Do not form or use out-of-bounds pointers or array subscripts		45 D	Pointer not checked for null before use.
				141 D	Heap-based array index not checked before use.
				47 S	Array bound exceeded.
				476 S	Array index not unsigned.
				489 S	Insufficient space for operation.
				692 S	Array index is negative.
				64 X	Array bound exceeded at call.
				66 X	Insufficient array space at call.
				68 X	Parameter indexing array too big at call.
				69 X	Global array bound exceeded at use.
				70 X	Array has insufficient space.
				71 X	Insufficient space for copy.
				79 X	Size mismatch in memcpy/memset.
ARR37-C	Rule	Do not add or subtract an integer to a pointer to a non-array object		567 S	Pointer arithmetic is not on array.
ARR38-C	Rule	Guarantee that library functions do not form invalid pointers		64 X	Array bound exceeded at call.
				66 X	Insufficient array space at call.
				68 X	Parameter indexing array too big at call.
				69 X	Global array bound exceeded at use.
				70 X	Array has insufficient space.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				71 X	Insufficient space for copy.
				79 X	Size mismatch in memcpy/memset.
ARR39-C	**Rule**	Do not add or subtract a scaled integer to a pointer		**47 S**	Array bound exceeded.
				489 S	Insufficient space for operation.
				567 S	Pointer arithmetic is not on array. Modifiers :436 = 1
				692 S	Array index is negative.
				64 X	Array bound exceeded at call.
				66 X	Insufficient array space at call.
				68 X	Parameter indexing array too big at call.
				69 X	Global array bound exceeded at use.
				70 X	Array has insufficient space.
				71 X	Insufficient space for copy.
CON00-CPP	**Recommendation**	Avoid assuming functions are thread safe unless otherwise specified			
CON01-CPP	**Recommendation**	Do not use volatile as a synchronization primitive			
CON02-CPP	**Recommendation**	Use lock classes for mutex management			
CON33-C	**Rule**	Avoid race conditions when using library functions			
CON37-C	**Rule**	Do not call signal() in a multithreaded program	The analysis highlights all uses of signal. The user must then manually check that the function is used a permitted by the rule.	**44 S**	Use of banned function, type or variable.
CON40-C	**Rule**	Do not refer to an atomic variable twice in an expression			
CON41-C	**Rule**	Wrap functions that can fail spuriously in a loop			
CON43-C	**Rule**	Avoid race conditions with multiple threads	Previously CON00-C		
CON50-CPP	**Rule**	Do not destroy a mutex while it is locked			
CON51-CPP	**Rule**	Ensure actively held locks are released on exceptional conditions			
CON52-CPP	**Rule**	Prevent data races when accessing bit-fields from multiple threads			
CON53-CPP	**Rule**	Avoid deadlock by locking in a predefined order			
CON54-CPP	**Rule**	Wrap functions that can spuriously wake up in a loop			
CON55-CPP	**Rule**	Preserve thread safety and liveness when using condition variables			
CON56-CPP	**Rule**	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread			
CTR00-CPP	**Recommendation**	Understand when to prefer vectors over arrays			
CTR01-CPP	**Recommendation**	Do not apply the sizeof operator to a pointer when taking the size of an array		**401 S**	Use of sizeof on an array parameter.
				577 S	Sizeof argument is a pointer.
CTR02-CPP	**Recommendation**	Explicitly specify array bounds, even if implicitly defined by an initializer	The analysis highlights that all arrays are given explicit bounds, including character arrays which are permitted by exception. The user must then manually check whether the exception applies.	**127 S**	Array has no bounds specified. Modifiers :285 = 0 (Default) 286 = 1 426 = 0 (Default)
				397 S	Array initialisation has insufficient items. Modifiers :415 = 0 (Default) 445 = 0 (Default) 450 = 0 (Default) 455 = 0 (Default)
				404 S	Array initialisation has too many items. Modifiers :399 = 0 (Default)
CTR03-CPP	**Recommendation**	Do not confuse the find() method with the find() algorithm			
CTR04-CPP	**Recommendation**	Assume responsibility for cleaning up data referenced by a container of pointers			
CTR05-CPP	**Recommendation**	Use explicit cv- and ref-qualifiers on auto declarations in range-based for loops			

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
CTR50-CPP	Rule	Guarantee that container indices and iterators are within the valid range		45 D	Pointer not checked for null before use.
				47 S	Array bound exceeded.
				476 S	Array index not unsigned.
				489 S	Insufficient space for operation.
				692 S	Array index is negative.
				64 X	Array bound exceeded at call.
				66 X	Insufficient array space at call.
				68 X	Parameter indexing array too big at call.
				69 X	Global array bound exceeded at use.
				70 X	Array has insufficient space.
				71 X	Insufficient space for copy.
				79 X	Size mismatch in memcpy/memset.
CTR51-CPP	Rule	Use valid references, pointers, and iterators to reference elements of a container			
CTR52-CPP	Rule	Guarantee that library functions do not form invalid iterators			
CTR53-CPP	Rule	Use valid iterator ranges			
CTR54-CPP	Rule	Do not subtract iterators that do not refer to the same container	The analysis highlights subtractions between pointers when it can determine that they do not point to the same array. The wording rule also refers to pointers comparisons and the analysis also highlights all use of pointer comparisons.	70 S	Logical comparison of pointers.
				87 S	Use of pointer arithmetic. Modifiers :tbend = <Default>
				437 S	< > <= >= used on different object pointers.
				438 S	Pointer subtraction not addressing one array.
CTR55-CPP	Rule	Do not use an additive operator on an iterator if the result would overflow	The analysis highlights all uses of pointer arithmetic that is not on array parameters. The user must then manually check whether the code is compliant with this rule.	567 S	Pointer arithmetic is not on array. Modifiers :436 = 1
CTR56-CPP	Rule	Do not use pointer arithmetic on polymorphic objects	The analysis highlights all uses of pointer arithmetic that is not on array parameters. The user must then manually check whether the code is compliant with this rule.	567 S	Pointer arithmetic is not on array. Modifiers :436 = 1
CTR57-CPP	Rule	Provide a valid ordering predicate			
CTR58-CPP	Rule	Predicate function objects should not be mutable			
DCL00-CPP	Recommendation	const-qualify immutable objects		78 D	Global variable should be declared const.
				93 D	Local variable should be declared const.
				200 S	Define used for numeric constant.
DCL01-CPP	Recommendation	Do not reuse variable names in subscopes		131 S	Name reused in inner scope.
				358 S	Class member name reused. Modifiers :246 = 1
DCL02-CPP	Recommendation	Use visually distinct identifiers		67 X	Identifier is typographically ambiguous.
DCL03-CPP	Recommendation	Use a static assertion to test the value of a constant expression	The analysis highlights all uses of assert. The user must then manually check whether the use is appropriate.	44 S	Use of banned function, type or variable.
DCL04-CPP	Recommendation	Do not declare more than one variable per declaration	This rule also states that each declaration should be followed by a comment.	579 S	More than one variable per declaration.
DCL05-CPP	Recommendation	Use typedefs to improve code readability	The recommendation does not specify when to use typedefs. The mappings can be extended to require basic types to be typedeff'ed using 90 S.	299 S	Pointer to function declared without typedef.
				381 S	Enum, struct or union not typedeffed.
DCL06-CPP	Recommendation	Use meaningful symbolic constants to represent literal values in program logic	Exception DLC06-EX1 permits numeric constants where this aids readability. This is not statically checkable, and therefore the analysis highlights all use of numeric literals in expressions. However, the documentation for 201 S shows how the analysis can be relaxed to permit a range of integer literals.	201 S	Use of numeric literal in expression. Modifiers :217 = 0 (Default) 163 = 1 (Default) 237 = 0 (Default) 207 = 0 (Default)
				604 S	Use of numeric literal as array bound/subscript. Modifiers :217 = 0 (Default) 163 = 1 (Default)
DCL07-CPP	Recommendation	Minimize the scope of variables and methods		25 D	Scope of variable could be reduced.
				40 S	Loop index is not declared locally.
				505 S	Control variable not declared in for loop.
				560 S	Scope of variable could be reduced.
DCL08-CPP	Recommendation	Properly encode relationships in constant definitions			

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
DCL09-CPP	Recommendation	Declare functions that return an errno error code with a return type of errno_t	The analysis checks that when errno is returned that the return type of the function is errno_t.	643 S	Function return type is not errno_t.
DCL10-CPP	Recommendation	Do not overload the ampersand, comma, logical AND or logical OR operators		211 S	Overloaded &&, || or comma.
				508 S	Operator & overloaded.
DCL11-CPP	Recommendation	Preserve operator semantics when overloading operators			
DCL12-CPP	Recommendation	Explicitly declare class template specializations		558 S	Template may lead to ill-formed program.
DCL13-CPP	Recommendation	Declare function parameters that are pointers to values not changed by the function as const		118 D	Object changed via dereferenced pointer.
				120 D	Pointer param should be declared pointer to const.
				582 S	const object reassigned.
DCL14-CPP	Recommendation	Avoid assumptions about the initialization order between translation units	The analysis highlights all functions with persistent local variables, but the user must than manually check that all uses are within the same translation unit	37 D	Function has persistent local side effects.
DCL15-CPP	Recommendation	Declare file-scope objects or functions that do not need external linkage in an unnamed namespace	The analysis highlights all objects and functions whose scope could be made to have internal linkage. The user must htan manually check whether the declarations appear in an unnamed namespace.	27 D	Variable should be declared static.
				61 D	Procedure should be declared static.
				553 S	Function and proto should both be static.
DCL16-CPP	Recommendation	Use "L," not "l," to indicate a long value		252 S	Lower case suffix to literal number. Modifiers :418 = 1
DCL17-CPP	Recommendation	Declare function parameters that are large data structures and are not changed by the function as const references			
DCL19-CPP	Recommendation	Initialize automatic local variables on declaration		64 D	Local not initialised at declaration.
				319 S	Constructor has insufficient initialisers. Modifiers :237 = 0 (Default)
DCL20-CPP	Recommendation	Use volatile for data that cannot be cached			
DCL21-CPP	Recommendation	Overloaded postfix increment and decrement operators should return a const object			
DCL22-CPP	Recommendation	Functions declared with [[noreturn]] must return void			
DCL30-C	Rule	Declare objects with appropriate storage durations		42 D	Local pointer returned in function result.
				77 D	Local structure returned in function result.
				71 S	Pointer assignment to wider scope. Modifiers :222 = 1
				565 S	Assignment to wider scope.
DCL39-C	Rule	Avoid information leakage when passing a structure across a trust boundary			
DCL40-C	Rule	Do not create incompatible declarations of the same function or object	The analysis does not currently check the declaration types of the functions across a system.	17 D	Identifier not unique within *** characters.
				1 X	Declaration types do not match across a system.
DCL50-CPP	Rule	Do not define a C-style variadic function		41 S	Ellipsis used in procedure parameter list. Modifiers :522 = 1
DCL51-CPP	Rule	Do not declare or define a reserved identifier		86 S	Attempt to define reserved word.
				218 S	Name is used in standard libraries.
				219 S	User name starts with underscore. Modifiers :412 = 0 (Default)
				580 S	Macro redefinition without using #undef.
DCL52-CPP	Rule	Never qualify a reference type with const or volatile			
DCL53-CPP	Rule	Do not write syntactically ambiguous declarations		296 S	Function declared at block scope.
DCL54-CPP	Rule	Overload allocation and deallocation functions as a pair in the same scope			
DCL55-CPP	Rule	Avoid information leakage when passing a class object across a trust boundary			
DCL56-CPP	Rule	Avoid cycles during initialization of static objects	The analysis highlights all use of recursion. The user must then manually check whether initialisation of a static object is involved.	6 D	Recursion in procedure calls found.
DCL57-CPP	Rule	Do not let exceptions escape from destructors or deallocation functions		453 S	Throw found in destructor.
DCL58-CPP	Rule	Do not modify the standard namespaces			
DCL59-CPP	Rule	Do not define an unnamed namespace in a header file	This rule also covers the definition of functions in header files if used in more than one translation unit.	286 S	Functions defined in header file. Modifiers :238 = 1
				512 S	Use of unnamed namespace.
DCL60-CPP	Rule	Obey the one-definition rule		286 S	Functions defined in header file. Modifiers :238 = 1
				287 S	Variable definition in header file. Modifiers :250 = 0 (Default)

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
ENV00-CPP	Recommendation	Beware of multiple environment variables with the same effective name			
ENV01-CPP	Recommendation	Sanitize the environment when invoking external programs	The analysis highlights all uses of the system function.	588 S	Use of system function.
ENV02-CPP	Recommendation	Do not call system() if you do not need a command processor		588 S	Use of system function.
ENV03-C	Recommendation	Sanitize the environment when invoking external programs	The analysis highlights uses of system. This can be extended to include the _exec functions as described in the documentation for 44 S.	588 S	Use of system function.
ENV30-C	Rule	Do not modify the object referenced by the return value of certain functions		107 D	Attempt to change system call capture string.
ENV31-C	Rule	Do not rely on an environment pointer following an operation that may invalidate it	The analysis highlights declarations of main which do not match the two standard specifcations given by the language. Adding a third parameter char * envp[] will violate this standard.	118 S	main must be int (void) or int (int,char*[]).
ENV32-C	Rule	All exit handlers must return normally	The analysis highlights all uses of exit and abort, and all jumps out of a procedure. The user must then manually check whether they occur within a call of atexit.	7 S	Jump out of procedure.
				122 S	Use of abort, exit, etc.
ENV33-C	Rule	Do not call system()		588 S	Use of system function.
ENV34-C	Rule	Do not store pointers returned by certain functions		133 D	Pointer from system function used after subsequent call.
ERR00-CPP	Recommendation	Adopt and implement a consistent and comprehensive error-handling policy			
ERR01-CPP	Recommendation	Use ferror() rather than errno to check for FILE stream errors	44 S highlights all uses of errno. The user must then manually check whether the use is related to the File stream.	44 S	Use of banned function, type or variable.
ERR02-CPP	Recommendation	Avoid in-band error indicators			
ERR03-CPP	Recommendation	Use runtime-constraint handlers when calling functions defined by TR24731-1			
ERR04-CPP	Recommendation	Choose an appropriate termination strategy			
ERR05-CPP	Recommendation	Application-independent code should provide error detection without dictating error handling			
ERR06-CPP	Recommendation	Understand the termination behavior of assert() and abort()	The analysis highlights all uses assert and termination functions. The user must then manually check whether the use is compliant with this rule.	44 S	Use of banned function, type or variable.
				122 S	Use of abort, exit, etc.
ERR07-CPP	Recommendation	Use exception handling rather than error codes			
ERR08-CPP	Recommendation	Prefer special-purpose types for exceptions		454 S	Throw type is not a class type.
				523 S	Exception type is pointer.
ERR09-CPP	Recommendation	Throw anonymous temporaries		454 S	Throw type is not a class type.
				455 S	Catch is not by reference.
ERR10-CPP	Recommendation	Check for error conditions		121 D	errno neither set nor checked for errno setting function.
				122 D	errno not checked after being set for errno setting fn.
				382 S	(void) missing for discarded return value. Modifiers :383 = 1
ERR12-CPP	Recommendation	Do not allow exceptions to transmit sensitive information			
ERR30-C	Rule	Set errno to zero before calling a library function known to set errno, and check errno only after the function returns a value indicating failure	111 D checks that errno has been set before library functions, which set errno, are called. 44 S which is mapped to ERR32-C will identify uses of errno which have not been set by a function	111 D	errno checked without having been set for errno setting fn.
				121 D	errno neither set nor checked for errno setting function.
				122 D	errno not checked after being set for errno setting fn.
				132 D	errno checked after call to non-errno setting function.
				134 D	errno not checked before subsequent function call.
ERR32-C	Rule	Do not rely on indeterminate values of errno	44 S highlights all uses of errno. The user must then manually check whether the use is compliant with this rule.	44 S	Use of banned function, type or variable.
ERR33-C	Rule	Detect and handle standard library errors		45 D	Pointer not checked for null before use.
				80 D	Potentially unused function-modified value.
				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.
ERR34-C	Rule	Detect errors when converting a string to a number	The rule suggests not using atoi etc: and sscanf.	44 S	Use of banned function, type or variable.
ERR50-CPP	Rule	Do not abruptly terminate the program	The analysis highlights uses of functions which terminate execution. The user must then manually check whether these functions may result in the call of a function that might throw an exception.	122 S	Use of abort, exit, etc.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
ERR51-CPP	Rule	Handle all exceptions		527 S	No master exception handler.
ERR52-CPP	Rule	Do not use setjmp() or longjmp()		43 S	Use of setjmp/longjmp.
ERR53-CPP	Rule	Do not reference base classes or class data members in a constructor or destructor function-try-block handler		549 S	Catch in c'tor/d'tor references nonstatic member.
ERR54-CPP	Rule	Catch handlers should order their parameter types from most derived to least derived		541 S	Catch-all is not last catch.
				556 S	Wrong order of catches for derived class.
ERR55-CPP	Rule	Honor exception specifications	The analysis highlights all cases of throws without an explicit catch. The user must then manually check whether a suitable exception-specification is present. No checks are made on whether library functions may throw an exception	56 D	Throw found with no catch in scope.
ERR56-CPP	Rule	Guarantee exception safety		56 D	Throw found with no catch in scope.
				71 D	No matching catch for throw in called function.
				527 S	No master exception handler.
ERR57-CPP	Rule	Do not leak resources when handling exceptions		50 D	Memory not freed after last reference.
ERR58-CPP	Rule	Handle all exceptions thrown before main() begins executing			
ERR59-CPP	Rule	Do not throw an exception across execution boundaries			
ERR60-CPP	Rule	Exception objects must be nothrow copy constructible			
ERR61-CPP	Rule	Catch exceptions by lvalue reference		455 S	Catch is not by reference.
ERR62-CPP	Rule	Detect errors when converting a string to a number			
EXP00-CPP	Recommendation	Use parentheses for precedence of operation		49 S	Logical conjunctions need brackets. Modifiers :206 = 1
				361 S	Expression needs brackets. Modifiers :119 = 0 (Default) 264 = 1 414 = 0 (Default) 420 = 0 (Default) 421 = 1
EXP01-CPP	Recommendation	Do not take the size of a pointer to determine the size of the pointed-to type		577 S	Sizeof argument is a pointer.
EXP02-CPP	Recommendation	Be aware of the short-circuit behavior of the logical AND and OR operators		35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				133 S	Assignment operator in RHS of && or ||.
				406 S	Use of ++ or -- on RHS of && or || operator.
				408 S	Volatile variable accessed on RHS of && or ||.
EXP03-CPP	Recommendation	Do not assume the size of a class or struct is the sum of the sizes of its members	578 S highlights all uses of sizeof on an arithmetic expression. The user must then manually check whether the expression violates this rule.	578 S	Sizeof used in arithmetic expression.
				638 S	Memory allocation non-conformant with type.
EXP05-CPP	Recommendation	Do not use C-style casts		306 S	Use of C type cast. Modifiers :458 = 0 (Default)
EXP07-CPP	Recommendation	Do not diminish the benefits of constants by assuming their values in expressions		201 S	Use of numeric literal in expression. Modifiers :217 = 0 (Default) 163 = 1 (Default) 237 = 0 (Default) 207 = 0 (Default)
EXP08-CPP	Recommendation	Ensure pointer arithmetic is used correctly	In addition to more specific, the analysis highlights all uses of pointer arithmetic. The user must then manually check whether the expression violates this rule.	45 D	Pointer not checked for null before use.
				53 D	Attempt to use uninitialised pointer.
				54 D	Unsafe use of function pointer variable.
				87 S	Use of pointer arithmetic. Modifiers :tbend = <Default>
				438 S	Pointer subtraction not addressing one array.
				576 S	Function pointer is of wrong type.
EXP09-CPP	Recommendation	Use sizeof to determine the size of a type or variable	201 S highlights all numeric literals in expressions. The user should manually confirm that sizeof would not be more appropriate.	201 S	Use of numeric literal in expression. Modifiers :217 = 0 (Default) 163 = 1 (Default) 237 = 0 (Default) 207 = 0 (Default)
EXP11-CPP	Recommendation	Do not apply operators expecting one type to data of an incompatible type		554 S	Cast to an unrelated type. Modifiers :439 = 1 440 = 0 (Default)

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
EXP12-CPP	Recommendation	Do not ignore values returned by functions or methods		382 S	(void) missing for discarded return value. Modifiers :383 = 1
EXP13-CPP	Recommendation	Prefer dynamic_cast over static_cast over reinterpret_cast		241 S	Use of reinterpret_cast.
EXP14-CPP	Recommendation	Do not use reinterpret_cast on pointers to class objects with multiple inheritence	The analysis highlights all uses of reinterpret_cast. Standard 241 S is only enabled when the the # symbol is removed from the start of the line in the c/cppreport.dat file.	241 S	Use of reinterpret_cast.
EXP15-CPP	Recommendation	Beware of integer promotion when performing bitwise operations on chars or shorts		334 S	No cast when ~ or << applied to small types (MR). Modifiers :191 = 1
EXP16-CPP	Recommendation	Avoid conversions using void pointers		540 S	Cast from pointer to void to pointer. Modifiers :381 = 0 (Default)
EXP17-CPP	Recommendation	Treat relational and equality operators as if they were nonassociative		433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
EXP18-CPP	Recommendation	Prefer the prefix forms of ++ and --	The analysis highlights all uses of prefix and postfix increment and decrement operators. The user must then manually check that a prefox form has been used.	30 S	Deprecated usage of ++ or -- operators found. Modifiers :122 = 0
EXP19-CPP	Recommendation	Do not perform assignments in conditional expressions		132 S	Assignment operator in boolean expression.
EXP34-C	Rule	Do not dereference null pointers		45 D	Pointer not checked for null before use.
				123 D	File pointer not checked for null before use.
				128 D	Global pointer not checked within this procedure.
				129 D	Global file pointer 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.
				652 S	Object created by malloc used before initialisation.
EXP35-C	Rule	Do not modify objects with temporary lifetime		42 D	Local pointer returned in function result.
				77 D	Local structure returned in function result.
				642 S	Function return type with array field.
EXP36-C	Rule	Do not cast pointers into more strictly aligned pointer types		540 S	Cast from pointer to void to pointer. Modifiers :381 = 0 (Default)
				554 S	Cast to an unrelated type. Modifiers :439 = 1 440 = 0 (Default)
				606 S	Cast involving function pointer. Modifiers :398 = 0 (Default) 433 = 1 439 = 1
				701 S	Pointer cast to pointer to void.
EXP37-C	Rule	Call functions with the correct number and type of arguments		41 D	Procedure call has no prototype declared.
				458 S	Implicit conversion: actual to formal param (MR).
				576 S	Function pointer is of wrong type.
EXP39-C	Rule	Do not access a variable through a pointer of an incompatible type		554 S	Cast to an unrelated type. Modifiers :439 = 1 440 = 0 (Default)
				697 S	Pointer converted to different pointer type. Modifiers :441 = 0 (Default)
EXP42-C	Rule	Do not compare padding data		618 S	Use of memcmp between structures.
EXP45-C	Rule	Do not perform assignments in selection statements	The analysis will also highlight the exceptions to this rule. The user must manually check that the exceptions do not apply.	114 S	Expression is not Boolean. Modifiers :386 = 1 (Default) 528 = 0 (Default)
				132 S	Assignment operator in boolean expression.
EXP46-C	Rule	Do not use a bitwise operator with a Boolean-like operand		136 S	Bit operator with boolean operand.
EXP47-C	Rule	Do not call va_arg with an argument of the incorrect type	44 S highlights all uses of va_arg. The user must then manually check that its use a permitted by the rule	44 S	Use of banned function, type or variable.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
EXP50-CPP	Rule	Do not depend on the order of evaluation for side effects		35 D	Expression has side effects.
				67 D	Void function has global variable 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)
				134 S	Volatile variable in complex expression. Modifiers :460 = 0 (Default) 461 = 0 (Default)
EXP51-CPP	Rule	Do not delete an array through a pointer of the incorrect type			
EXP52-CPP	Rule	Do not rely on side effects in unevaluated operands		54 S	Sizeof operator with side effects.
				133 S	Assignment operator in RHS of && or ||.
EXP53-CPP	Rule	Do not read uninitialized memory		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.
EXP54-CPP	Rule	Do not access an object outside of its lifetime		42 D	Local pointer returned in function result.
				53 D	Attempt to use uninitialised pointer.
				77 D	Local structure returned in function result.
				1 J	Unreachable Code found.
				71 S	Pointer assignment to wider scope. Modifiers :222 = 1
				565 S	Assignment to wider scope.
EXP55-CPP	Rule	Do not access a cv-qualified object through a cv-unqualified type		203 S	Cast on a constant value. Modifiers :390 = 0 (Default)
				242 S	Use of const_cast.
				344 S	Cast on volatile value. Modifiers :395 = 0 (Default)
EXP56-CPP	Rule	Do not call a function with a mismatched language linkage			
EXP57-CPP	Rule	Do not cast or delete pointers to incomplete classes	The analysis highlights all uses of forward reference to class members. The user must manually check that a cast or delete does not occur on a pointer to that member.	169 S	Use of forward reference of class member.
				554 S	Cast to an unrelated type. Modifiers :439 = 1 440 = 0 (Default)
EXP58-CPP	Rule	Pass an object of the correct type to va_start			
EXP59-CPP	Rule	Use offsetof() on valid types and members			
EXP60-CPP	Rule	Do not pass a nonstandard-layout type object across execution boundaries			
EXP61-CPP	Rule	A lambda object must not outlive any of its reference captured objects			
EXP62-CPP	Rule	Do not access the bits of an object representation that are not part of the object's value representation	The analysis highlights all uses of memcmp between structures. The user must then manually check whether the exception is present.	618 S	Use of memcmp between structures.
EXP63-CPP	Rule	Do not rely on the value of a moved-from object			
FIO00-CPP	Recommendation	Take care when creating format strings		486 S	Incorrect number of formats in output function.
				589 S	Format is not appropriate type.
FIO01-CPP	Recommendation	Be careful using functions that use file names for identification		592 S	Use of filename based functions.
FIO02-CPP	Recommendation	Canonicalize path names originating from untrusted sources	This rule is in general undecidable. The analysis only considers those cases that are decidable.	85 D	Filename not verified before fopen.
FIO03-CPP	Recommendation	Do not make assumptions about fopen() and file creation	The analysis highlights all uses of fopen and fopen_s. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO04-CPP	Recommendation	Detect and handle input and output errors		80 D	Potentially unused function-modified value.
				382 S	(void) missing for discarded return value. Modifiers :383 = 1
FIO05-CPP	Recommendation	Identify files using multiple file attributes	The analysis highlights all uses of fopen. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
FIO06-CPP	Recommendation	Create files with appropriate access permissions	The analysis highlights all uses of fopen. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO07-CPP	Recommendation	Prefer fseek() to rewind()		593 S	Use fseek() rather than rewind().
FIO08-CPP	Recommendation	Take care when calling remove() on an open file		81 D	Attempt to remove an open file.
FIO09-CPP	Recommendation	Be careful with binary data when transferring data across systems	The analysis highlights all uses of fread and fwrite. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO10-CPP	Recommendation	Take care when using the rename() function	The analysis highlights all uses of rename. The user must manually check whether they are protected as described in the rule	592 S	Use of filename based functions.
FIO11-CPP	Recommendation	Take care when specifying the mode parameter of fopen()		590 S	Mode fault in fopen.
FIO12-CPP	Recommendation	Prefer setvbuf() to setbuf()		594 S	Use setvbuf() rather than setbuf().
FIO13-CPP	Recommendation	Never push back anything other than one read character		83 D	Potentially repeated call to ungetc.
FIO14-CPP	Recommendation	Understand the difference between text mode and binary mode with file streams			
FIO15-CPP	Recommendation	Ensure that file operations are performed in a secure directory			
FIO17-CPP	Recommendation	Prefer streams to C-style input and output	The analysis highlights the use of cstdio, providing CERT-CPP is added to the appropriate line in the cppvals.dat file	130 S	Included file is not permitted.
FIO18-CPP	Recommendation	Never expect write() to terminate the writing process at a null character	The analysis highlights all uses of write. The user must manually check whether they are protected as described in the rule	44 S	Use of banned function, type or variable.
FIO19-CPP	Recommendation	Do not create temporary files in shared directories	The analysis highlights all uses of tmpnam, tmpfile, mktemp and tmpnam_s. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO20-CPP	Recommendation	Do not rely on an ending null character when using read()			
FIO21-CPP	Recommendation	Do not simultaneously open the same file multiple times		75 D	Attempt to open file pointer more than once.
FIO30-C	Rule	Exclude user input from format strings	The analysis highlights all uses of possibly tainted values from the I/O library functions.	86 D	User input not checked before use.
FIO32-C	Rule	Do not perform operations on devices that are only appropriate for files			
FIO34-C	Rule	Distinguish between characters read from a file and EOF or WEOF	The analysis highlights all implicit conversions on plain char,	433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				662 S	EOF compared with char.
FIO37-C	Rule	Do not assume that fgets() or fgetws() returns a nonempty string when successful	The analysis highlights all uses of fgets and fgetws. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO38-C	Rule	Do not copy a FILE object		591 S	Inappropriate use of file pointer.
FIO39-C	Rule	Do not alternately input and output from a stream without an intervening flush or positioning call		84 D	No fseek or flush before I/O.
FIO40-C	Rule	Reset strings on fgets() or fgetws() failure	The analysis highlights all uses of fgets and fgetws. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
FIO41-C	Rule	Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects		35 D	Expression has side effects.
				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 = 0
				134 S	Volatile variable in complex expression. Modifiers :460 = 0 (Default) 461 = 0 (Default)
FIO42-C	Rule	Close files when they are no longer needed		49 D	File pointer not closed on exit.
FIO44-C	Rule	Only use values for fsetpos() that are returned from fgetpos()		82 D	fsetpos values not generated by fgetpos.
FIO45-C	Rule	Avoid TOCTOU race conditions while accessing files		75 D	Attempt to open file pointer more than once.
FIO46-C	Rule	Do not access a closed file		48 D	Attempt to write to unopened file.
FIO47-C	Rule	Use valid format strings		486 S	Incorrect number of formats in output function.
				589 S	Format is not appropriate type.
FIO50-CPP	Rule	Do not alternately input and output from a file stream without an intervening positioning call			
FIO51-CPP	Rule	Close files when they are no longer needed			

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
FLP00-CPP	Recommendation	Understand the limitations of floating-point numbers			
FLP01-CPP	Recommendation	Take care in rearranging floating-point expressions			
FLP02-CPP	Recommendation	Avoid using floating-point numbers when precise computation is needed		56 S	Equality comparison of floating point. Modifiers :223 = 1
FLP03-CPP	Recommendation	Detect and handle floating point errors		43 D	Divide by zero found.
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				445 S	Narrower float conversion without cast. Modifiers :374 = 0 (Default)
				451 S	No cast for widening complex float expression (MR). Modifiers :191 = 1 529 = 0 (Default)
FLP04-CPP	Recommendation	Check floating point inputs for exceptional values			
FLP05-CPP	Recommendation	Convert integers to floating point for floating point operations	The analysis also highlights situations when one operand has integer type and the other has floating type.	435 S	Float/integer conversion without cast. Modifiers :374 = 0 (Default) 432 = 0 (Default) 453 = 1
FLP30-C	Rule	Do not use floating point variables as loop counters		39 S	Unsuitable type for loop variable.
FLP32-C	Rule	Prevent or detect domain and range errors in math functions			
FLP34-C	Rule	Ensure that floating-point conversions are within range of the new type	453 S highlights all implicit conversions from integer to floating. The user must then manually check whether the conversion is protected. See comments under FLP36-C for further configuration of 435 S.	435 S	Float/integer conversion without cast. Modifiers :374 = 0 (Default) 432 = 0 (Default) 453 = 1
				444 S	Integral type cast to non-integral. Modifiers :509 = 0 (Default) 510 = 1 511 = 0 (Default) 512 = 1 513 = 0 (Default) 529 = 0 (Default)
				441 S	Float cast to non-float. Modifiers :502 = 0 (Default) 503 = 0 (Default) 504 = 0 (Default) 505 = 0 (Default) 506 = 0 (Default) 529 = 0 (Default)
				445 S	Narrower float conversion without cast. Modifiers :374 = 0 (Default)
				693 S	Float expression cast to narrower float type.
FLP36-C	Rule	Preserve precision when converting integral values to floating-point type	453 S highlights all conversions from integer to floating, as this standard is also mapped to recommendation FLP06-C. If recommendation FLP06-C is not being followed, 453 S can be relaxed to permit the conversion of integer to floats when the integer size is less the float size by setting 453 to 0 in all uses of 453 S within the model.	435 S	Float/integer conversion without cast. Modifiers :374 = 0 (Default) 432 = 0 (Default) 453 = 1
FLP37-C	Rule	Do not use object representations to compare floating-point values	The analysis highlights all uses of memcmp on structures. The user must then manually check the the structure does not contains floating point members, or involve padding bits (see EXP42-C)	618 S	Use of memcmp between structures.
INT00-CPP	Recommendation	Understand the data model used by your implementation(s)			
INT01-CPP	Recommendation	Use rsize_t or size_t for all integer values representing the size of an object		458 S	Implicit conversion: actual to formal param (MR).
INT02-CPP	Recommendation	Understand integer conversion rules		52 S	Unsigned expression negated.
				101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default)
				107 S	Type mismatch in ternary expression. Modifiers :428 = c++ : 1 (Default) 429 = 1 470 = 0 (Default) 525 = 0 (Default)
				332 S	Widening cast on complex integer expression. Modifiers :529 = 0 (Default)
				334 S	No cast when ~ or << applied to small types (MR). Modifiers :191 = 1
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				434 S	Signed/unsigned conversion without cast. Modifiers :374 = 0 (Default) 358 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 429 = 1 430 = 0 (Default) 470 = 0 (Default)
				446 S	Narrower int conversion without cast. Modifiers :374 = 0 (Default)
				452 S	No cast for widening complex int expression (MR). Modifiers :191 = 1 529 = 0 (Default)
				457 S	Implicit int widening for function return (MR). Modifiers :191 = 1
				458 S	Implicit conversion: actual to formal param (MR).
INT03-CPP	Recommendation	Use a secure integer library			
INT04-CPP	Recommendation	Enforce limits on integer values originating from untrusted sources			
INT05-CPP	Recommendation	Do not use input functions to convert character data if they cannot handle all possible inputs	The analysis highlights all uses of scanf and related functions. The user must manually check whether they are protected as described in the rule.	44 S	Use of banned function, type or variable.
INT06-CPP	Recommendation	Use strtol() or a related function to convert a string token to an integer	The analysis highlights all uses of atoi and related functions.	44 S	Use of banned function, type or variable.
INT07-CPP	Recommendation	Use only explicitly signed or unsigned char type for numeric values		101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default)
				329 S	Operation not appropriate to plain char. Modifiers :191 = 1 331 = 0 (Default) 391 = 0 (Default)
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
INT08-CPP	Recommendation	Verify that all integer values are in range	The checks on numeric overflow/underflow are restricted to those cases where the value of a variable can be determined statically.	488 S	Value outside range of underlying type. Modifiers :535 = 0 (Default)
				493 S	Numeric overflow.
INT09-CPP	Recommendation	Ensure enumeration constants map to unique values		85 S	Incomplete initialisation of enumerator. Modifiers :377 = 0 (Default)
				630 S	Duplicated enumeration value.
INT10-CPP	Recommendation	Do not assume a positive remainder when using the % operator		584 S	Remainder of % op could be negative.
INT11-CPP	Recommendation	Take care when converting from pointer to integer or integer to pointer		439 S	Cast from pointer to integral type. Modifiers :548 = 0 (Default)
				440 S	Cast from integral type to pointer. Modifiers :397 = 0 (Default) 548 = 0 (Default)
INT12-CPP	Recommendation	Do not make assumptions about the type of a plain int bit-field when used in an expression		520 S	Bit field is not bool or explicit integral. Modifiers :331 = 0 (Default) 350 = 1
INT13-CPP	Recommendation	Use bitwise operators only on unsigned operands		50 S	Use of shift operator on signed type.
				51 S	Shifting value too far. Modifiers :520 = 0 (Default)
				120 S	Use of bit operator on signed type. Modifiers :457 = 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 = 0 (Default)
				345 S	Bit operator with floating point operand.
				403 S	Negative (or potentially negative) shift.
INT14-CPP	Recommendation	Avoid performing bitwise and arithmetic operations on the same data			
INT15-CPP	Recommendation	Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types			
INT16-CPP	Recommendation	Do not make assumptions about representation of		50 S	Use of shift operator on signed type.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
INT16-CPP	Recommendation	signed integers		120 S	Use of bit operator on signed type. Modifiers :457 = 1
INT17-CPP	Recommendation	Define integer constants in an implementation-independent manner			
INT18-CPP	Recommendation	Evaluate integer expressions in a larger size before comparing or assigning to that size		452 S	No cast for widening complex int expression (MR). Modifiers :191 = 1 529 = 0 (Default)
INT30-C	Rule	Ensure that unsigned integer operations do not wrap	The checks on numeric overflow/underflow are restricted to those cases where the value of a variable can be determined statically.	493 S	Numeric overflow.
INT31-C	Rule	Ensure that integer conversions do not result in lost or misinterpreted data		433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				434 S	Signed/unsigned conversion without cast. Modifiers :374 = 0 (Default) 358 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 429 = 1 430 = 0 (Default) 470 = 0 (Default)
INT32-C	Rule	Ensure that operations on signed integers do not result in overflow	The checks on numeric overflow/underflow are restricted to those cases where the value of a variable can be determined statically.	493 S	Numeric overflow.
INT33-C	Rule	Ensure that division and remainder operations do not result in divide-by-zero errors	The analysis will not highlight those cases where a check is present, but is insufficient.	43 D	Divide by zero found.
				127 D	Local or member denominator not checked before use.
				131 D	Global denominator not checked within this procedure.
				248 S	Divide by zero in preprocessor directive.
				629 S	Divide by zero found.
				80 X	Divide by zero found.
INT34-C	Rule	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand		51 S	Shifting value too far. Modifiers :520 = 0 (Default)
				403 S	Negative (or potentially negative) shift.
				479 S	Right shift loses all bits.
INT35-C	Rule	Use correct integer precisions			
INT36-C	Rule	Converting a pointer to integer or integer to pointer	This analysis covers explicit casts from pointers to integers and vice versa. Implicit conversions are constraint errors in the C (See C11 6.5.16.1) and should be caught by your compiler.	439 S	Cast from pointer to integral type. Modifiers :548 = 0 (Default)
				440 S	Cast from integral type to pointer. Modifiers :397 = 0 (Default) 548 = 0 (Default)
INT50-CPP	Rule	Do not cast to an out-of-range enumeration value			
MEM00-CPP	Recommendation	Overloaded new operators should check that their size argument matches the size of their class			
MEM01-CPP	Recommendation	Store a valid value in pointers immediately after deallocation	Storing a value in the pointer is designed to prevent problems with double frees. The analysis highlights the double free.	112 D	Free called twice on same variable.
				484 S	Attempt to use already freed object. Modifiers :350 = 1
MEM02-CPP	Recommendation	Immediately cast the result of a memory allocation function call into a pointer to the allocated type			
MEM03-CPP	Recommendation	Clear sensitive information stored in returned reusable resources			
MEM04-CPP	Recommendation	Do not perform zero-length allocations			
MEM05-CPP	Recommendation	Avoid large stack allocations	The analysis highlights use of recursion and possible infinite loops.	5 C	Procedure contains infinite loop.
				6 D	Recursion in procedure calls found.
				28 D	Potentially infinite loop found.
				26 S	Loop control expression may not terminate loop. Modifiers :425 = 0 (Default)
				140 S	Infeasible loop condition found.
				1 U	Inter-file recursion found.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
MEM06-CPP	Recommendation	Ensure that sensitive data is not written out to disk			
MEM07-CPP	Recommendation	Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t			
MEM08-CPP	Recommendation	Use new and delete rather than raw memory allocation and deallocation		44 S	Use of banned function, type or variable.
MEM09-CPP	Recommendation	Do not assume memory allocation routines initialize memory		652 S	Object created by malloc used before initialisation.
MEM10-CPP	Recommendation	Define and use a pointer validation function	The analysis highlights all pointer comparisons with 0 or NULL. The user must then manually check that the comparison occurs within a dedicated pointer validation function.	159 S	Comparing pointer with zero or NULL.
MEM11-CPP	Recommendation	Allocate and free memory in the same module, at the same level of abstraction		50 D	Memory not freed after last reference.
MEM12-CPP	Recommendation	Do not assume infinite heap space	The analysis highlights use of recursion and possible infinite loops.	5 C	Procedure contains infinite loop.
				6 D	Recursion in procedure calls found.
				28 D	Potentially infinite loop found.
				26 S	Loop control expression may not terminate loop. Modifiers :425 = 0 (Default)
				140 S	Infeasible loop condition found.
				1 U	Inter-file recursion found.
MEM13-CPP	Recommendation	Use smart pointers instead of raw pointers for resource management			
MEM30-C	Rule	Do not access freed memory		51 D	Attempt to read from freed memory.
				112 D	Free called twice on same variable.
				484 S	Attempt to use already freed object. Modifiers :350 = 1
MEM31-C	Rule	Free dynamically allocated memory when no longer needed		50 D	Memory not freed after last reference.
MEM34-C	Rule	Only free memory allocated dynamically		125 D	free called on variable with no allocated space.
				407 S	free used on string.
				483 S	Freed parameter is not heap item.
				644 S	realloc ptr does not originate from allocation function.
				645 S	realloc ptr type does not match target type.
MEM35-C	Rule	Allocate sufficient memory for an object	The analysis highlights all uses of sizeof on a type. The user must then manually check whether the correct type has been chosen	115 D	Copy length parameter not checked before use.
				400 S	Use of sizeof on a type.
				487 S	Insufficient space allocated.
MEM36-C	Rule	Do not modify the alignment of objects by calling realloc()		644 S	realloc ptr does not originate from allocation function.
MEM50-CPP	Rule	Do not access freed memory		483 S	Freed parameter is not heap item.
				484 S	Attempt to use already freed object. Modifiers :350 = 1
MEM51-CPP	Rule	Properly deallocate dynamically allocated resources		64 D	Local not initialised at declaration.
				112 D	Free called twice on same variable.
				232 S	No destructor defined for class.
				236 S	New used in class without assignment op.
				239 S	New used in class without copy constructor.
				407 S	free used on string.
				469 S	No copy constructor for complex destructor.
				470 S	No assignment operator for complex destrtor.
				483 S	Freed parameter is not heap item.
				484 S	Attempt to use already freed object. Modifiers :350 = 1

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				485 S	Array deletion without [].
MEM52-CPP	**Rule**	Detect and handle memory allocation errors		**45 D**	Pointer not checked for null before use.
MEM53-CPP	**Rule**	Explicitly construct and destruct objects when manually managing object lifetime			
MEM54-CPP	**Rule**	Provide placement new with properly aligned pointers to sufficient storage capacity	The analysis highlights all uses of placement. The user must then manually check whether sufficent storage has been allocated.	**597 S**	Use of placement new.
MEM55-CPP	**Rule**	Honor replacement dynamic storage management requirements			
MEM56-CPP	**Rule**	Do not store an already-owned pointer value in an unrelated smart pointer			
MEM57-CPP	**Rule**	Avoid using default operator new for over-aligned types			
MSC00-CPP	**Recommendation**	Compile cleanly at high warning levels			
MSC01-CPP	**Recommendation**	Strive for logical completeness	The analysis highlights missing paths in the control flow structure.	**48 S**	No default case in switch statement. Modifiers :252 = 0 (Default) 233 = 1
				59 S	Else alternative missing in if. Modifiers :530 = 0 (Default)
MSC02-CPP	**Recommendation**	Avoid errors of omission		**99 S**	Function use is not a call. Modifiers :498 = 1
				132 S	Assignment operator in boolean expression.
MSC03-CPP	**Recommendation**	Avoid errors of addition		**5 S**	Empty then clause.
				57 S	Statement with no side effect. Modifiers :458 = 0 (Default)
				58 S	Null statement found.
				59 S	Else alternative missing in if. Modifiers :530 = 0 (Default)
MSC04-CPP	**Recommendation**	Use comments consistently and in a readable fashion	The analysis is currently not configured to catch the following examples a = b //*divisor:*/c +d; or /\ / j();	**119 S**	Nested comment found. Modifiers :413 = 0 (Default) 434 = 0 (Default)
				302 S	Comment possibly contains code.
				611 S	Line splice used in // comment. Modifiers :402 = 0 (Default)
MSC05-CPP	**Recommendation**	Do not manipulate time_t typed values directly		**101 S**	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default)
				107 S	Type mismatch in ternary expression. Modifiers :428 = c++ : 1 (Default) 429 = 1 470 = 0 (Default) 525 = 0 (Default)
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
MSC06-CPP	**Recommendation**	Be aware of compiler optimization when dealing with sensitive data		**65 D**	Void function has no side effects.
				76 D	Procedure is not called or referenced in code analysed.
				105 D	DU anomaly dead code, var value is unused on all paths. Modifiers :391 = 0 (Default)
				151 D	DD data flow anomaly found. Modifiers :391 = 0 (Default)
				1 J	Unreachable Code found.
				3 J	All internal linkage calls unreachable.
				35 S	Static procedure is not explicitly called in code analysed.
				57 S	Statement with no side effect. Modifiers :458 = 0 (Default)

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
				631 S	Declaration not reachable.
MSC07-CPP	Recommendation	Detect and remove dead code	This rule is in general undecidable. The analysis only considers those cases that are deciadable.	65 D	Void function has no side effects.
				105 D	DU anomaly dead code, var value is unused on all paths. Modifiers :391 = 0 (Default)
				151 D	DD data flow anomaly found. Modifiers :391 = 0 (Default)
				1 J	Unreachable Code found.
				57 S	Statement with no side effect. Modifiers :458 = 0 (Default)
				139 S	Construct leads to infeasible code.
				140 S	Infeasible loop condition found.
				631 S	Declaration not reachable.
MSC08-CPP	Recommendation	Functions should validate their parameters	The analysis highlights the use of objects that have been given a value from a standard library IO function, whose value has not been subsequently checked.	85 D	Filename not verified before fopen.
				86 D	User input not checked before use.
MSC09-CPP	Recommendation	Character encoding: Use subset of ASCII for safety	This rule applies to naming files, variables, and other objects, which includes the naming of files via strings. The analysis includes a check for strings that contain characters outside the C++basic character set. The user must then manually check whether the string is used as a filename. In order to minimise false positives, @ and $ are permitted but can be prohibited by using modifier 408 as described in documentation for 113 S.	86 D	User input not checked before use.
				113 S	Non standard character in source. Modifiers :408 = 1 449 = 0 (Default)
MSC10-CPP	Recommendation	Character encoding: UTF8-relateds		176 S	Non standard escape sequence in source.
MSC11-CPP	Recommendation	Incorporate diagnostic tests using assertions			
MSC12-CPP	Recommendation	Detect and remove code that has no effect		65 D	Void function has no side effects.
				105 D	DU anomaly dead code, var value is unused on all paths. Modifiers :391 = 0 (Default)
				57 S	Statement with no side effect. Modifiers :458 = 0 (Default)
MSC13-CPP	Recommendation	Detect and remove unused values	151 D highlights all DD anomalies, which includes variables that are re-assigned down only one path. As this can be noisy in legacy code, it has been classified as Informational and can be switched off.	1 D	Unused procedure parameter.
				15 D	Unused procedural parameter.
				91 D	Function return value potentially unused.
				94 D	Named variable declared but not used in code.
				105 D	DU anomaly dead code, var value is unused on all paths. Modifiers :391 = 0 (Default)
				151 D	DD data flow anomaly found. Modifiers :391 = 0 (Default)
MSC14-CPP	Recommendation	Do not introduce unnecessary platform dependencies	Note: Undefined behaviour is covered by recommendation MSC15-CPP and other more specific rules	17 D	Identifier not unique within *** characters.
				42 S	Use of bit field in structure declaration.
				69 S	#pragma used.
				48 D	Attempt to write to unopened file.
				63 D	No definition in system for prototyped procedure.
				84 D	No fseek or flush before I/O.
				113 D	File closed more than once.
				5 Q	File does not end with new line.
				64 S	Void procedure used in expression. Modifiers :218 = 1
				65 S	Void variable passed as parameter.
				100 S	#include filename is non conformant. Modifiers :120 = \, (Default)
				109 S	Array subscript is not integral.
				156 S	Use of 'defined' keyword in macro body.
				296 S	Function declared at block scope.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
MSC15-CPP	Recommendation	Do not depend on undefined behavior	The mappings in this rule refer to the undefined features that are not covered by more specific rule.	324 S	Macro call has wrong number of parameters.
				335 S	Operator defined contains illegal items.
				336 S	#if expansion contains define operator.
				339 S	#include directive with illegal items.
				412 S	Undefined behaviour, \ before E-O-F.
				427 S	Filename in #include not in < > or " ".
				465 S	Struct/union not completely specified.
				482 S	Incomplete structure referenced.
				497 S	Type is incomplete in translation unit.
				545 S	Assignment of overlapping storage.
				587 S	Const local variable not immediately initialised.
				608 S	Use of explicitly undefined language feature.
				62 X	Function prototype/defn return type mismatch (MR).
				63 X	Function prototype/defn param type mismatch (MR).
MSC16-CPP	Recommendation	Consider encrypting function pointers			
MSC17-CPP	Recommendation	Do not use deprecated or obsolescent functionality			
MSC18-CPP	Recommendation	Finish every set of statements associated with a case label with a break statement	Exception 2 permits missing break statements if that is what is intended. The analysis will highlight all missing break statements, unless there a no statements between two case labels.	62 S	Switch case not terminated with break. Modifiers :283 = 1 384 = c++ : 0 (Default) 551 = 0 (Default)
MSC19-CPP	Recommendation	Do not define static private members	The analysis all static class members. The user must then manually check whether the member has been declared private.	38 S	Use of static class member. Modifiers :423 = 0 (Default)
MSC20-CPP	Recommendation	Do not use a switch statement to transfer control into a complex block		245 S	Case statement in nested block.
MSC21-CPP	Recommendation	Use inequality to terminate a loop whose counter changes by more than one		510 S	Loop counter increment and operator defect.
MSC30-C	Rule	Do not use the rand() function for generating pseudorandom numbers		44 S	Use of banned function, type or variable.
MSC32-C	Rule	Properly seed pseudorandom number generators			
MSC33-C	Rule	Do not pass invalid data to the asctime() function	44 S highlights all uses of asctime. The user must then manually check that the function is used a permitted by the rule.	44 S	Use of banned function, type or variable.
MSC37-C	Rule	Ensure that control never reaches the end of a non-void function		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.
MSC38-C	Rule	Do not treat a predefined identifier as an object if it might only be implemented as a macro			
MSC39-C	Rule	Do not call va_arg() on a va_list that has an indeterminate value			
MSC40-C	Rule	Do not violate constraints	The analysis should be performed after compilation by the user's compiler. The standards mapped to this rule protect from constraint errors, which have been known to go undetected by some compilers.	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 = 0 (Default)
				481 S	Array with no bounds in struct.
				580 S	Macro redefinition without using #undef.
				612 S	inline function should be declared static.
				615 S	Conditional operator has incompatible types.
				646 S	Struct initialisation has too many items.
MSC50-CPP	Rule	Do not use std::rand() for generating pseudorandom numbers	The analysis highlights all uses of the rand function.	44 S	Use of banned function, type or variable.
MSC51-CPP	Rule	Ensure your random number generator is properly seeded			
MSC52-CPP	Rule	Value-returning functions must return a value from all exit paths		2 D	Function does not return a value on all paths.
				36 S	Function has no return statement. Modifiers :553 = 0 (Default)
MSC53-CPP	Rule	Do not return from a function declared [[noreturn]]			

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
MSC54-CPP	Rule	A signal handler must be a plain old function			
OOP00-CPP	Recommendation	Declare data members private		202 S	Class data is not explicitly private.
OOP01-CPP	Recommendation	Be careful with the definition of conversion operators		394 S	Conversion function found.
OOP02-CPP	Recommendation	Do not hide inherited non-virtual member functions		262 S	Non virtual function redefined.
OOP03-CPP	Recommendation	Prefer not to overload virtual functions		601 S	Insufficient overridden members.
OOP04-CPP	Recommendation	Prefer not to give virtual functions default argument initializers	The analysis highlights all uses of default parameters. The user must then manually check whether a virtual function is present	359 S	Default parameter use. Modifiers :401 = 0 (Default)
OOP05-CPP	Recommendation	Avoid deleting this			
OOP06-CPP	Recommendation	Create a private copy constructor and assignment operator for non copyable objects		233 S	No copy constructor for class with pointers.
				234 S	No assignment operator for class with pointers.
OOP07-CPP	Recommendation	Do not inherit from multiple classes that have distinct objects with the same name	The analysis highlights all cases where a base class member that is not unique. The user must then ,anually check whether multiple inheritance is involved.	555 S	Base class member name not unique.
OOP08-CPP	Recommendation	Do not return references to private data		392 S	Class data accessible thru non const member.
				671 S	Class data accessible thru non const handle.
OOP09-CPP	Recommendation	Ensure that single-argument constructors are marked "explicit"		393 S	Single parameter constructor not 'explicit'.
OOP10-CPP	Recommendation	Do not assume that copy constructor invocations will not be elided	The analysis highlights all changes to static members in constructors.	529 S	Static member initialised/assigned in constructor.
OOP11-CPP	Recommendation	Do not copy-initialize members or base classes from a move constructor			
OOP50-CPP	Rule	Do not invoke virtual functions from constructors or destructors		92 D	C'tor/d'tor calls virtual function.
				467 S	Virtual member called in ctor/dtor.
OOP51-CPP	Rule	Do not slice derived objects			
OOP52-CPP	Rule	Do not delete a polymorphic object without a virtual destructor		303 S	Virtual class members need virtual destructor. Modifiers :292 = 0 (Default)
OOP53-CPP	Rule	Write constructor member initializers in the canonical order		206 S	Class initialiser out of order.
OOP54-CPP	Rule	Gracefully handle self-copy assignment			
OOP55-CPP	Rule	Do not use pointer-to-member operators to access nonexistent members			
OOP56-CPP	Rule	Honor replacement handler requirements			
OOP57-CPP	Rule	Prefer special member functions and overloaded operators to C Standard Library functions	The analysis highlights all uses of functions listed in this rule.	44 S	Use of banned function, type or variable.
OOP58-CPP	Rule	Copy operations must not mutate the source object			
PRE00-CPP	Recommendation	Avoid defining macros		79 S	Macro contains unacceptable items. Modifiers :298 = 0 (Default) 331 = 0 (Default)
				340 S	Use of function like macro.
PRE01-CPP	Recommendation	Use parentheses within macros around parameter names	The analysis highlights all missing brackets. The user must then manually check whether they are allowed by the exceptions to this rule.	78 S	Macro parameter not in brackets.
PRE02-CPP	Recommendation	Macro replacement lists should be parenthesized		77 S	Macro replacement list needs parentheses.
PRE03-CPP	Recommendation	Prefer typedefs to defines for encoding types	The analysis restricts the use of macro to those places permitted by MISRA-C:2004 rule 19.4. The user must then manually filter the messages for those relating to PRE03-CPP.	79 S	Macro contains unacceptable items. Modifiers :298 = 0 (Default) 331 = 0 (Default)
PRE04-CPP	Recommendation	Do not reuse a standard header file name		568 S	#include "filename" uses standard library name. Modifiers :344 = 1
PRE05-CPP	Recommendation	Understand macro replacement when concatenating tokens or performing stringification	The analysis highlights all uses of # and ##. The user must then manually check that the use is permitted by the rule.	76 S	More than one of # or ## in a macro.
				125 S	Use of ## or # in a macro.
				637 S	# operand followed by ##.
PRE06-CPP	Recommendation	Enclose header files in an inclusion guard		243 S	Included file not protected with #define.
PRE07-CPP	Recommendation	Avoid using repeated question marks		81 S	Use of trigraph.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
PRE08-CPP	Recommendation	Guarantee that header file names are unique			
PRE09-CPP	Recommendation	Do not replace secure functions with less secure functions			
PRE10-CPP	Recommendation	Do not define unsafe macros	The analysis looks at the result of the expansion and not the declaration of the macro. 562 S highlights all uses of ++ etc in macro parameters. The user must then manually check whether this may lead to side effects.	35 D	Expression has side effects.
				72 D	Potential side effect problem in expression.
				1 Q	Call has execution order dependant side effects.
				134 S	Volatile variable in complex expression. Modifiers :460 = 0 (Default) 461 = 0 (Default)
				562 S	Use of ++,-- or = in macro parameters.
PRE11-CPP	Recommendation	Do not conclude macro definitions with a semicolon		58 S	Null statement found.
				699 S	Macro terminated with semi-colon.
PRE30-C	Rule	Do not create a universal character name through concatenation		573 S	Macro concatenation of uni char names.
PRE31-C	Rule	Avoid side effects in arguments to unsafe macros		35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				562 S	Use of ++,-- or = in macro parameters.
				572 S	Side effect in assert.
PRE32-C	Rule	Do not use preprocessor directives in invocations of function-like macros	The analysis will only detect issues with library functions that are implemented using macros if the analysis scope is set to expand library header files.	341 S	Preprocessor construct as macro parameter.
SIG00-CPP	Recommendation	Mask signals handled by noninterruptible signal handlers		87 D	Illegal shared object in signal handler.
SIG01-CPP	Recommendation	Understand implementation-specific details regarding signal handler persistence		97 D	Signal called from within signal handler.
SIG02-CPP	Recommendation	Avoid using signals to implement normal functionality	44 S highlights all uses of signal. The user must then manually check that the function is used a permitted by the rule.	44 S	Use of banned function, type or variable.
SIG31-C	Rule	Do not access shared objects in signal handlers		87 D	Illegal shared object in signal handler.
SIG34-C	Rule	Do not call signal() from within interruptible signal handlers		97 D	Signal called from within signal handler.
SIG35-C	Rule	Do not return from a computational exception signal handler	44 S highlights all uses of signal. The user must then manually check that the function is used a permitted by the rule.	44 S	Use of banned function, type or variable.
STR00-CPP	Recommendation	Represent characters using an appropriate type		101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default)
				329 S	Operation not appropriate to plain char. Modifiers :191 = 1 331 = 0 (Default) 391 = 0 (Default)
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
STR01-CPP	Recommendation	Adopt and implement a consistent plan for managing strings			
STR02-CPP	Recommendation	Sanitize data passed to complex subsystems		108 D	Tainted argument to unprototyped func ptr.
				109 D	Tainted argument to formatted i/o function.
				588 S	Use of system function.

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
STR03-CPP	Recommendation	Do not inadvertently truncate a null-terminated character array	The analysis highlights use of strncpy, strncat, fgets and snprintf. Other functions can be added as described in the documentation for 44 S.	44 S	Use of banned function, type or variable.
				115 S	String incorrectly terminated.
STR04-CPP	Recommendation	Use plain char for characters in the basic character set		101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 1 470 = 0 (Default)
				329 S	Operation not appropriate to plain char. Modifiers :191 = 1 331 = 0 (Default) 391 = 0 (Default)
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
STR05-CPP	Recommendation	Use pointers to const when referring to string literals		623 S	String assigned to non const object. Modifiers :119 = 0 (Default)
STR06-CPP	Recommendation	Do not assume that strtok() leaves the parse string unchanged	The analysis highlights all uses of strtok with a string parameter. The user must then manually check that the use is permitted by the rule.	602 S	strtok may change the parse string.
STR07-CPP	Recommendation	Don't assume numeric values for expressions with type plain character		329 S	Operation not appropriate to plain char. Modifiers :191 = 1 331 = 0 (Default) 391 = 0 (Default)
STR08-CPP	Recommendation	Do not specify the bound of a character array initialized with a string literal	Exception STR36-EX2 permits bounds if the array's contents might change later. This is not statically checkable, so the analysis permits such initializations and reports only those initialisations where the NULL terminator is not stored.	404 S	Array initialisation has too many items. Modifiers :399 = 0 (Default)
STR30-C	Rule	Do not attempt to modify string literals		157 S	Modification of string literal. Modifiers :438 = 1
				623 S	String assigned to non const object. Modifiers :119 = 0 (Default)
STR31-C	Rule	Guarantee that storage for strings has sufficient space for character data and the null terminator		109 D	Tainted argument to formatted i/o function.
				140 D	Copy source parameter not checked before use.
				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.
STR32-C	Rule	Do not pass a non-null-terminated character sequence to a library function that expects a string	44S which is mapped to other rules will highlight the use of the deprecated function strncpy.	404 S	Array initialisation has too many items. Modifiers :399 = 0 (Default)
				600 S	Argument of strlen is unterminated.
STR34-C	Rule	Cast characters to unsigned char before converting to larger integer sizes		433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 1 428 = c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
STR37-C	Rule	Arguments to character-handling functions must be representable as an unsigned char	The analysis considers that all plain char objects are acceptable. The user should manually perform further checks if char is implemented as signed char.	663 S	Invalid value may be passed to function in <ctype.h>.
STR38-C	Rule	Do not confuse narrow and wide character strings and functions			
STR50-CPP	Rule	Guarantee that storage for strings has sufficient space for character data and the null terminator		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.
STR51-CPP	Rule	Do not attempt to create a std::string from a null pointer			
STR52-CPP	Rule	Use valid references, pointers, and iterators to reference elements of a basic_string			

CERT-C++:2016 Standards Model Compliance for C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
STR53-CPP	Rule	Range check element access			

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.