MISRA-C++:2008 Standards Model Summary for C++

The LDRA tool suite® is developed and certified to BS EN ISO 9001:2000.

This information is applicable to version 9.4.2 of the LDRA tool suite®.
It is correct as of 25th September 2013.

Compliance is measured against
"MISRA C++:2008 Guidelines for the use of the C++ language in critical systems"
June 2008
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
Required	1	162	29	5	1	**198**
Advisory	0	15	2	1	0	**18**
Document	0	2	2	0	8	**12**
Total	**1**	**179**	**33**	**6**	**9**	**228**

MISRA-C++:2008 Standards Model Compliance for C++

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
0-1-1	Required	A project shall not contain unreachable code.	1 J	Unreachable Code found.
0-1-2	Required	A project shall not contain infeasible paths.	139 S	Construct leads to infeasible code.
0-1-3	Required	A project shall not contain unused variables.	94 D	Named variable declared but not used in code.
0-1-4	Required	A project shall not contain non-volatile POD variables having only one use.	3 X	Variable has only one use.
0-1-5	Required	A project shall not contain unused type declarations.	413 S	User type declared but not used in code analysed.
0-1-6	Required	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.	70 D	DU anomaly, variable value is not used.
			560 S	Scope of variable could be reduced.
0-1-7	Required	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.	382 S	(void) missing for discarded return value.
0-1-8	Required	All functions with void return type shall have external side effect(s).	65 D	Void function has no side effects.
0-1-9	Required	There shall be no dead code.	70 D	DU anomaly, variable value is not used.
0-1-10	Required	Every defined function shall be called at least once.	76 D	Procedure is not called or referenced in code analysed.
			2 U	Procedure not called anywhere in system.
0-1-11	Required	There shall be no unused parameters (named or unnamed) in	1 D	Unused procedure parameter.
			15 D	Unused procedural parameter.

0-1-12	Required	There shall be no unused parameters (named or unnamed) in	1 D	Unused procedure parameter.
			15 D	Unused procedural parameter.
0-2-1	Required	An object shall not be assigned to an overlapping object.	480 S	String function params access same variable.
			545 S	Assignment of overlapping storage.
0-3-1	Document	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis	43 D	Divide by 0 found.
			248 S	Divide by zero in preprocessor directive.
0-3-2	Required	If a function generates error information, then that error information shall be tested.		
0-4-1	Document	Use of scaled-integer or fixed-point arithmetic shall be documented.		
0-4-2	Document	Use of floating-point arithmetic shall be documented.		
0-4-3	Document	Floating-point implementations shall comply with a defined floating-point standard.		
1-0-1	Required	All code shall conform to ISO/IEC 14882:2003 "The C++ Standard Incorporating Technical Corrigendum 1".	145 S	#if has invalid expression.
			404 S	Array initialisation has too many items.
			481 S	Array with no bounds in struct.
			580 S	Macro redefinition without using #undef.
			615 S	Conditional operator has incompatible types.
1-0-2	Document	Multiple compilers shall only be used if they have a common, defined interface.		
1-0-3	Document	The implementation of integer division in the chosen compiler shall be determined and documented.		

2-2-1	Document	The character set and the corresponding encoding shall be documented.		
2-3-1	Required	Trigraphs shall not be used.	81 S	Use of trigraph.
2-5-1	Advisory	Digraphs should not be used.	295 S	Use of digraph.
2-7-1	Required	The character sequence /* shall not be used within a C-style comment.	119 S	Nested comment found.
2-7-2	Required	Sections of code shall not be "commented out" using C-style comments.	302 S	Comment possibly contains code.
2-7-3	Advisory	Sections of code should not be "commented out" using C++ comments.	302 S	Comment possibly contains code.
2-10-1	Required	Different identifiers shall be typographically unambiguous.	217 S	Names only differ by case.
			67 X	Identifier is typographically ambiguous.
2-10-2	Required	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.	128 S	Parameter has same name as global variable.
			131 S	Name reused in inner scope.
2-10-3	Required	A typedef name (including qualification, if any) shall be a unique identifier.	112 S	Typedef name redeclared.
			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.
			21 X	Identifier reuse: typedef vs macro.
			22 X	Identifier reuse: typedef vs component.
			23 X	Identifier reuse: typedef vs enum constant.

			24 X	Identifier reuse: typedef vs procedure.
2-10-4	Required	A class, union or enum name (including qualification, if any) shall be a unique identifier.	552 S	Class, enum, struct or union name reused.
			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.
			12 X	Identifier reuse: tag vs macro.
			13 X	Identifier reuse: tag vs component.
			14 X	Identifier reuse: tag vs enum constant.
			15 X	Identifier reuse: persistent var vs tag.
			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.

2-10-5	Advisory	The identifier name of a non-member object or function with static storage duration should not be reused.	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.
			32 X	Identifier reuse: procedure vs var.
			33 X	Identifier reuse: procedure vs label (MR).
			34 X	Identifier reuse: proc vs macro.
			35 X	Identifier reuse: proc vs component.
			36 X	Identifier reuse: proc vs enum constant.
			37 X	Identifier reuse: persistent var vs macro.
			38 X	Identifier reuse: persistent var vs component.
			39 X	Identifier reuse: persistent var vs enum constant.
2-10-6	Required	If an identifier refers to a type, it shall not also refer to an object or a function in the same scope.	9 X	Identifier reuse: tag vs variable.
			11 X	Identifier reuse: tag vs typedef.
			24 X	Identifier reuse: typedef vs procedure.
2-13-1	Required	Only those escape sequences that are defined in ISO/IEC14882:2003 shall be used.	176 S	Non standard escape sequence in source.
2-13-2	Required	Octal constants (other than zero) and octal escape sequences (other	83 S	Octal number found.
			376 S	Use of octal escape sequence.
2-13-3	Required	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	550 S	Unsuffixed hex or octal is unsigned, add U.
2-13-4	Required	Literal suffixes shall be upper case.	252 S	Lower case suffix to literal number.
2-13-5	Required	Narrow and wide string literals shall not be concatenated.	450 S	Wide string and string concatenated.

3-1-1	Required	It shall be possible to include any header file in multiple translation	286 S	Functions defined in header file.
			287 S	Variable definition in header file.
3-1-2	Required	Functions shall not be declared at block scope.	296 S	Function declared at block scope.
3-1-3	Required	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.	127 S	Array has no bounds specified.
3-2-1	Required	All declarations of an object or function shall have compatible types.	1 X	Declaration types do not match across a system.
			62 X	Function prototype/defn return type mismatch (MR).
3-2-2	Required	The One Definition Rule shall not be violated.	26 D	Variable should be defined once in only one file.
			34 D	Procedure name re-used in different files.
			4 X	Identifier reuse: struct/union tag repeated.
			18 X	Identifier reuse: typedef vs typedef.
3-2-3	Required	A type, object or function that is used in multiple translation units shall be declared in one and only one file.	60 D	External object should be declared only once.
3-2-4	Required	An identifier with external linkage shall have exactly one definition.	26 D	Variable should be defined once in only one file.
			33 D	No real declaration for external variable.
			63 D	No definition in system for prototyped procedure.
3-3-1	Required	Objects or functions with external linkage shall be declared in a header file.	27 D	Variable should be declared static.
			61 D	Procedure should be declared static.
			354 S	Extern declaration is not in header file.

3-3-2	Required	If a function has internal linkage then all re-declarations shall include the static storage class specifier.	553 S	Function and proto should both be static.
3-4-1	Required	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.	25 D	Scope of variable could be reduced.
			560 S	Scope of variable could be reduced.
3-9-1	Required	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.	102 S	Function and prototype return inconsistent (MR).
			103 S	Function and prototype param inconsistent (MR).
			2 X	Ambiguous declaration of variable.
			62 X	Function prototype/defn return type mismatch (MR).
			63 X	Function prototype/defn param type mismatch (MR).
3-9-2	Advisory	typedefs that indicate size and signedness should be used in place of the basic numerical types.	90 S	Basic type declaration used.
3-9-3	Required	The underlying bit representations of floating-point values shall not be used.	345 S	Bit operator with floating point operand.
			554 S	Cast to an unrelated type.
4-5-1	Required	Expressions with type bool shall not be used as operands to built-in operators other than the assignment	136 S	Bit operator with boolean operand.
			506 S	Use of boolean with relational operator.
4-5-2	Required	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [ ], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=.	123 S	Use of underlying enum representation value.

4-5-3	Required	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.	329 S	Operation not appropriate to plain char.
4-10-1	Required	NULL shall not be used as an integer value.	530 S	NULL used in integer context.
4-10-2	Required	Literal zero (0) shall not be used as the null-pointer-constant.	531 S	Literal zero used in pointer context.
5-0-1	Required	The value of an expression shall be the same under any order of evaluation that the standard permits.	35 D	Expression has side effects.
			72 D	Potential side effect problem in expression.
			74 D	Potential side effect from repeated function call.
			1 Q	Call has execution order dependant side effects.
			9 S	Assignment operation in expression.
			30 S	Deprecated usage of ++ or -- operators found.
			134 S	Volatile variable in complex expression.
5-0-2	Advisory	Limited dependence should be placed on C++ operator precedence rules in expressions.	49 S	Logical conjunctions need brackets.
			361 S	Expression needs brackets.
5-0-3	Required	A cvalue expression shall not be implicitly converted to a different underlying type.	452 S	No cast for widening complex int expression.

5-0-4	Required	An implicit integral conversion shall not change the signedness of the underlying type.	96 S	Use of mixed mode arithmetic.
			101 S	Function return type inconsistent.
			107 S	Type mismatch in ternary expression.
			331 S	Literal value requires a U suffix.
			434 S	Signed/unsigned conversion without cast.
5-0-5	Required	There shall be no implicit floating-integral conversions.	101 S	Function return type inconsistent.
			107 S	Type mismatch in ternary expression.
			435 S	Float/integer conversion without cast.
5-0-6	Required	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.	101 S	Function return type inconsistent.
			445 S	Narrower float conversion without cast.
			446 S	Narrower int conversion without cast.
5-0-7	Required	There shall be no explicit floating-integral conversions of a cvalue expression.	507 S	Explicit cast from integral to floating type.
5-0-8	Required	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.	332 S	Widening cast on complex integer expression.
			333 S	Widening cast on complex float expression.
5-0-9	Required	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.	443 S	Unsigned integral type cast to signed.
5-0-10	Required	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	334 S	No cast when ~ or << applied to small types.

5-0-11	Required	The plain char type shall only be used for the storage and use of character values.	431 S	Char used instead of (un)signed char.
			432 S	Inappropriate type - should be plain char.
			433 S	Type conversion without cast.
5-0-12	Required	signed char and unsigned char type shall only be used for the storage and use of numeric values.	431 S	Char used instead of (un)signed char.
			432 S	Inappropriate type - should be plain char.
			433 S	Type conversion without cast.
5-0-13	Required	The condition of an if-statement and the condition of an iteration-statement shall have type bool.	114 S	Expression is not Boolean.
5-0-14	Required	The first operand of a conditional-operator shall have type bool.	114 S	Expression is not Boolean.
5-0-15	Required	Array indexing shall be the only form of pointer arithmetic.	436 S	Declaration does not specify an array.
			567 S	Pointer arithmetic is not on array.
5-0-16	Required	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both	47 S	Array bound exceeded.
			436 S	Declaration does not specify an array.
5-0-17	Required	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	438 S	Pointer subtraction not addressing one array.
5-0-18	Required	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.	437 S	< > <= >= used on different object pointers.
5-0-19	Required	The declaration of objects shall contain no more than two levels of pointer indirection.	80 S	Pointer indirection exceeds 2 levels.
5-0-20	Required	Non-constant operands to a binary bitwise operator shall have the same underlying type.	535 S	Binary bitwise op with different types.

5-0-21	Required	Bitwise operators shall only be applied to operands of unsigned underlying type.	120 S	Use of bit operator on signed type.
5-2-1	Required	Each operand of a logical && or || shall be a postfix-expression.	49 S	Logical conjunctions need brackets.
5-2-2	Required	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.	448 S	Base class pointer cast to derived class.
5-2-3	Advisory	Casts from a base class to a derived class should not be performed on polymorphic types.	551 S	Cast from base to derived for polymorphic type.
5-2-4	Required	C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used.	306 S	Use of C type cast.
5-2-5	Required	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.	203 S	Cast on a constant value.
			242 S	Use of const_cast.
			344 S	Cast on volatile value.
5-2-6	Required	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.	606 S	Cast involving function pointer.
5-2-7	Required	An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.	94 S	Casting operation on a pointer.
			95 S	Casting operation to a pointer.
			554 S	Cast to an unrelated type.
5-2-8	Required	An object with integer type or pointer to void type shall not be converted to	440 S	Cast from integral type to pointer.
			540 S	Cast from pointer to void to pointer.
5-2-9	Advisory	A cast should not convert a pointer type to an integral type.	439 S	Cast from pointer to integral type.
5-2-10	Advisory	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	30 S	Deprecated usage of ++ or -- operators found.

5-2-11	Required	The comma operator, && operator and the || operator shall not be overloaded.	211 S	Overloaded &&, || or comma.
5-2-12	Required	An identifier with array type passed as a function argument shall not	459 S	Array passed as actual parameter.
			534 S	Array has decayed to pointer.
5-3-1	Required	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.	114 S	Expression is not Boolean.
5-3-2	Required	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	52 S	Unsigned expression negated.
5-3-3	Required	The unary & operator shall not be overloaded.	508 S	Operator & overloaded.
5-3-4	Required	Evaluation of the operand to the sizeof operator shall not contain side effects.	54 S	Sizeof operator with side effects.
5-8-1	Required	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	51 S	Shifting value too far.
5-14-1	Required	The right hand operand of a logical && or || operator shall not contain side effects.	406 S	Use of ++ or -- on RHS of && or || operator.
5-17-1	Required	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.		
5-18-1	Required	The comma operator shall not be used.	53 S	Use of comma operator.
5-19-1	Advisory	Evaluation of constant unsigned integer expressions should not lead	493 S	Numeric overflow.
			494 S	Numeric underflow.

6-2-1	Required	Assignment operators shall not be used in sub-expressions.	9 S	Assignment operation in expression.
			132 S	Assignment operator in boolean expression.
6-2-2	Required	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.	56 S	Equality comparison of floating point.
6-2-3	Required	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.	58 S	Null statement found.
6-3-1	Required	The statement forming the body of a switch, while, do...while or for statement shall be a compound statement.	11 S	No brackets to loop body (added by Testbed).
6-4-1	Required	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	12 S	No brackets to then/else (added by Testbed).
6-4-2	Required	All if ... else if constructs shall be terminated with an else clause.	59 S	Else alternative missing in if.
6-4-3	Required	A switch statement shall be a well-formed switch statement.	385 S	MISRA switch statement syntax violation.
6-4-4	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.
6-4-5	Required	An unconditional throw or break statement shall terminate every non-empty switch-clause.	62 S	Switch case not terminated with break.

6-4-6	Required	The final clause of a switch statement shall be the default-clause.	48 S	No default case in switch statement.
			322 S	Default is not last case of switch.
6-4-7	Required	The condition of a switch statement shall not have bool type.	121 S	Use of boolean expression in switch.
6-4-8	Required	Every switch statement shall have at least one case-clause.	61 S	Switch contains default only.
6-5-1	Required	A for loop shall contain a single loop-counter which shall not have floating type.	38 D	More than one control variable for loop.
6-5-2	Required	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.	510 S	Loop counter increment and operator defect.
6-5-3	Required	The loop-counter shall not be modified within condition or statement.	55 D	Modification of loop counter in loop body.
			73 D	Predicate variable modified in condition.
6-5-4	Required	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.	271 S	For loop incrementation is not simple.
6-5-5	Required	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.	537 S	Extra loop control variable changed.
6-5-6	Required	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.	66 D	Non boolean control variable modified in loop.
			542 S	Extra loop control variable is not bool.
6-6-1	Required	Any label referenced by a goto statement shall be declared in the	1 J	Unreachable Code found.
			511 S	Jump into nested block.
6-6-2	Required	The goto statement shall jump to a label declared later in the same function body.	509 S	goto label is backwards.

6-6-3	Required	The continue statement shall only be used within a well-formed for loop.	32 S	Use of continue statement.
			65 X	continue in ill-formed loop.
6-6-4	Required	For any iteration statement there shall be no more than one break or goto statement used for loop termination.	409 S	More than one break or goto statement in loop.
6-6-5	Required	A function shall have a single point of exit at the end of the function.	7 C	Procedure has more than one exit point.
7-1-1	Required	A variable which is not modified shall be const qualified.	59 D	Parameter should be declared const.
			78 D	Global variable should be declared const.
			93 D	Local variable should be declared const.
			603 S	Parameter should be declared * const.
7-1-2	Required	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.	62 D	Pointer parameter should be declared const.
7-2-1	Required	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	411 S	Inappropriate value assigned to enum.
7-3-1	Required	The global namespace shall only contain main, namespace declarations and extern "C" declarations.	517 S	At least one declaration in global namespace.
7-3-2	Required	The identifier main shall not be used for a function other than the global function main.	362 S	main program in a namespace.
7-3-3	Required	There shall be no unnamed namespaces in header files.	512 S	Use of unnamed namespace.
7-3-4	Required	using-directives shall not be used.	513 S	Use of using directive.

7-3-5	Required	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.	518 S	Using declaration is straddled by declarations.
7-3-6	Required	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	514 S	Using directive or declaration in header.
7-4-1	Document	All usage of assembler shall be documented.	17 S	Code insert found.
7-4-2	Required	Assembler instructions shall only be introduced using the asm declaration.	516 S	Assembler does not use asm declaration.
7-4-3	Required	Assembly language shall be encapsulated and isolated.	88 S	Procedure is not pure assembler.
7-5-1	Required	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the	42 D	Local pointer returned in function result.
			564 S	Reference assignment to wider scope.
7-5-2	Required	The address of an object with automatic storage shall not be	71 S	Pointer assignment to wider scope.
			565 S	Assignment to wider scope.
7-5-3	Required	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.	519 S	Return of reference parameter.
7-5-4	Advisory	Functions should not call themselves, either directly or	6 D	Recursion in procedure calls found.
			1 U	Inter-file recursion found.
8-0-1	Required	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.	515 S	More than one variable in declaration.

8-3-1	Required	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	364 S	Inherited default parameter redefined.
8-4-1	Required	Functions shall not be defined using the ellipsis notation.	41 S	Ellipsis used in procedure parameter list.
8-4-2	Required	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	36 D	Prototype and definition name mismatch.
8-4-3	Required	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	36 S	Function has no return statement.
			66 S	Function with empty return expression.
			2 D	Function does not return a value on all paths.
8-4-4	Required	A function identifier shall either be used to call the function or it shall be preceded by & .	99 S	Function use is not a call.
8-5-1	Required	All variables shall have a defined value before they are used.	69 D	UR anomaly, variable used before assignment.
8-5-2	Required	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.	105 S	Initialisation brace { } fault.
			397 S	Array initialisation has insufficient items.
8-5-3	Required	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.	85 S	Incomplete initialisation of enumerator.
9-3-1	Required	const member functions shall not return non-const pointers or references to class-data.	392 S	Class data accessible thru non const member.
9-3-2	Required	Member functions shall not return non-const handles to class-data.	392 S	Class data accessible thru non const member.

9-3-3	Required	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.	46 D	Member function may be declared const.
			79 D	Member function may be declared static.
9-5-1	Required	Unions shall not be used.	74 S	Union declared.
9-6-1	Document	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.	42 S	Use of bit field in structure declaration.
			328 S	Non bit field member in bitfield struct.
9-6-2	Required	Bit-fields shall be either bool type or an explicitly unsigned or signed integral type.	520 S	Bit field is not bool or explicit integral.
9-6-3	Required	Bit-fields shall not have enum type.	520 S	Bit field is not bool or explicit integral.
			536 S	Enum type in bit field.
9-6-4	Required	Named bit-fields with signed integer type shall have a length of more than one bit.	72 S	Signed bit field less than 2 bits wide.
10-1-1	Advisory	Classes should not be derived from virtual bases.	521 S	Class derived from virtual base class.
10-1-2	Required	A base class shall only be declared virtual if it is used in a diamond hierarchy.	543 S	Virtual base not in diamond hierarchy.
10-1-3	Required	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.	357 S	Base class is mixed virtual and non virtual.
10-2-1	Advisory	All accessible entity names within a multiple inheritance hierarchy should be unique.	555 S	Base class member name not unique.
10-3-1	Required	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy.	559 S	Virtual member defined more than once.

10-3-2	Required	Each overriding virtual function shall be declared with the virtual keyword.	214 S	Member not declared virtual.
10-3-3	Required	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.	544 S	Member re-declared pure.
11-0-1	Required	Member data in non-POD class types shall be private.	202 S	Class data is not explicitly private.
12-1-1	Required	An object's dynamic type shall not be used from the body of its constructor or destructor.	92 D	C'tor/d'tor calls virtual function.
			467 S	Virtual member called in ctor/dtor.
			561 S	Use of dynamic type in c'tor/d'tor.
12-1-2	Advisory	All constructors of a class should explicitly call a constructor for all of its immediate base classes and all virtual base classes.	528 S	Base class constructor omitted (added by Testbed).
12-1-3	Required	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	393 S	Single parameter constructor not 'explicit'.
12-8-1	Required	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.	529 S	Static member initialised/assigned in constructor.
12-8-2	Required	The copy assignment operator shall be declared protected or private in an abstract class.	522 S	Public assign operator in abstract class.
14-5-1	Required	A non-member generic function shall only be declared in a namespace that is not an associated namespace.		
14-5-2	Required	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter.	532 S	No copy ctor for templated copy ctor.

14-5-3	Required	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.	533 S	No assgnmt optor for templated assgmnt op.
14-6-1	Required	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->	547 S	Base member not qualified.
14-6-2	Required	The function chosen by overload resolution shall resolve to a function declared previously in the translation unit.	546 S	Overload resolution could be forward.
14-7-1	Required	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once.	538 S	Template class/function/member not instantiated.
14-7-2	Required	For any given template specialization, an explicit instantiation of the template with the template-arguments used in the specialization shall not render the program ill-formed.	558 S	Template may lead to ill-formed program.
14-7-3	Required	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.		
14-8-1	Required	Overloaded function templates shall not be explicitly specialized.	539 S	Specialised overloaded templated function.

14-8-2	Advisory	The viable function set for a function call should either contain no function specializations, or only contain function specializations.		
15-0-1	Document	Exceptions shall only be used for error handling.		
15-0-2	Advisory	An exception object should not have pointer type.	523 S	Exception type is pointer.
15-0-3	Required	Control shall not be transferred into a try or catch block using a goto or a switch statement.	524 S	Jump into try or catch statement.
15-1-1	Required	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.	557 S	Function call can generate throw exception.
15-1-2	Required	NULL shall not be thrown explicitly.	525 S	throw with explicit NULL.
15-1-3	Required	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.	526 S	Empty throw is not in catch statement.
15-3-1	Required	Exceptions shall be raised only after start-up and before termination of the program.	557 S	Function call can generate throw exception.
15-3-2	Advisory	There should be at least one exception handler to catch all otherwise unhandled exceptions	527 S	No master exception handler.
15-3-3	Required	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.	549 S	Catch in c'tor/d'tor references nonstatic member.
15-3-4	Required	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.	56 D	Throw found with no catch in scope.
			71 D	No matching catch for throw in called function.
15-3-5	Required	A class type exception shall always be caught by reference.	455 S	Catch is not by reference.

15-3-6	Required	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.	556 S	Wrong order of catches for derived class.
15-3-7	Required	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.	541 S	Catch-all is not last catch.
15-4-1	Required	If a function is declared with an exception-specification, then all declarations of the same function (in other translation units) shall be declared with the same set of type-ids.		
15-5-1	Required	A class destructor shall not exit with an exception.	453 S	Throw found in destructor.
15-5-2	Required	Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s).		
15-5-3	Required	The terminate() function shall not be called implicitly.		
16-0-1	Required	#include directives in a file shall only be preceded by other preprocessor directives or comments.	75 S	Executable code before an included file.
			338 S	#include preceded by non preproc directives.
16-0-2	Required	Macros shall only be #define'd or #undef'd in the global namespace.	67 S	#define used in a block.
			426 S	#undef used in a block.
16-0-3	Required	#undef shall not be used.	68 S	#undef used.

16-0-4	Required	Function-like macros shall not be defined.	340 S	Use of function like macro.
16-0-5	Required	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	341 S	Preprocessor construct as macro parameter.
16-0-6	Required	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##.	78 S	Macro parameter not in brackets.
16-0-7	Required	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.	337 S	Undefined macro variable in #if.
16-0-8	Required	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.	147 S	Spurious characters after preprocessor directive.
			342 S	Extra chars after preprocessor directive.
16-1-1	Required	The defined preprocessor operator shall only be used in one of the two standard forms.	335 S	Operator defined contains illegal items.
			336 S	#if expansion contains define operator.
16-1-2	Required	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef	126 S	A #if has no #endif in the same file.
			343 S	#else has no #if, etc in the same file.
16-2-1	Required	The pre-processor shall only be used for file inclusion and include guards.	255 S	Found #if, #ifdef, #else, #elif .
			307 S	Use of #line, #error preprocessor directives.
			340 S	Use of function like macro.
16-2-2	Required	C++ macros shall only be used for include guards, type qualifiers, or storage class specifiers.	79 S	Macro contains unacceptable items.
16-2-3	Required	Include guards shall be provided.	243 S	Included file not protected with #define.

16-2-4	Required	The ', ", /* or // characters shall not occur in a header file name.	100 S	#include filename is non conformant.
16-2-5	Advisory	The \ character should not occur in a header file name.	100 S	#include filename is non conformant.
			339 S	#include directive with illegal items.
16-2-6	Required	The #include directive shall be followed by either a <filename> or "filename" sequence.	292 S	No space between #include and filename.
			339 S	#include directive with illegal items.
			427 S	Filename in #include not in < > or " ".
16-3-1	Required	There shall be at most one occurrence of the # or ## operators in a single macro definition.	76 S	More than one of # or ## in a macro.
16-3-2	Advisory	The # and ## operators should not be used.	125 S	Use of ## or # in a macro.
16-6-1	Document	All uses of the #pragma directive shall be documented.	69 S	#pragma used.
17-0-1	Required	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.	44 S	Use of banned function or variable.
			68 S	#undef used.
			86 S	Attempt to define reserved word.
			156 S	Use of 'defined' keyword in macro body.
17-0-2	Required	The names of standard library macros and objects shall not be reused.	218 S	Name is used in standard libraries.
17-0-3	Required	The names of standard library functions shall not be overridden.	218 S	Name is used in standard libraries.
17-0-4	Document	All library code shall conform to MISRA C++.		
17-0-5	Required	The setjmp macro and the longjmp function shall not be used.	43 S	Use of setjmp/longjmp.
18-0-1	Required	The C library shall not be used.	130 S	Included file is not permitted.
18-0-2	Required	The library functions atof, atoi and atol from library <cstdlib> shall not be used.	44 S	Use of banned function or variable.

18-0-3	Required	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.	122 S	Use of abort, exit, etc.
18-0-4	Required	The time handling functions of library <ctime> shall not be used.	130 S	Included file is not permitted.
18-0-5	Required	The unbounded functions of library <cstring> shall not be used.	130 S	Included file is not permitted.
18-2-1	Required	The macro offsetof shall not be used.	44 S	Use of banned function or variable.
18-4-1	Required	Dynamic heap memory allocation shall not be used.	44 S	Use of banned function or variable.
18-7-1	Required	The signal handling facilities of <csignal> shall not be used.	130 S	Included file is not permitted.
19-3-1	Required	The error indicator errno shall not be used.	44 S	Use of banned function or variable.
27-0-1	Required	The stream input/output library <cstdio> shall not be used.	130 S	Included file is not permitted.