MISRA C++

Table of Contents

1 Introduction

MISRA C++ is a set of coding guidelines and standards for the C++ programming language, designed to enhance the safety, reliability, and maintainability of software, particularly in safety-critical and high-integrity systems. Below are some key categories and examples of MISRA C++ rules:

1.1 Environment Rules:

- Rule 2-1-1: A code file shall not be identical to a standard library header file.

1.2 Language Rules:

- Rule 4-0-1: All C++ source code files shall include exactly one definition of the main function.
- Rule 4-12-1: There shall be no global resource leaks.

1.3 Documentation Rules:

- Rule 5-0-1: Each header file shall have a comment block.

1.4 Class Rules:

- Rule 6-0-1: Only one class shall be defined in each file..

1.5 Function Rules:

- Rule 8-0-1: The rightmost parameter of a function declaration shall be at the rightmost of the function declarator.
- Rule 8-5-1: A function parameter shall not have a const-qualified type.

1.6 Variable Rules:

- Rule 9-0-1: There shall be at most one variable declaration per statement.
- Rule 9-3-1: A variable shall not be defined with the same name as a predefined macro.

1.7 Pointer and Reference Rules:

- Rule 17-0-1: A pointer shall point to a const-qualified type whenever possible.

1.8 Exception Handling Rules:

- Rule 16-0-1: All exceptions thrown by a function shall be caught or explicitly propagated out of the function.

1.9 Concurrency Rules:

- Rule 19-0-1: The value of a pointer shall not be used after the memory object that it points to has been reallocated.

1.10 Preprocessor Rules:

- Rule 20-0-1: #include directives in a file should be sorted to groups and should not include other directives.
- Rule 20-3-1: Avoid use of the '# ' and '##' operators.

1.11 Resource Management Rules:

- Rule 21-0-1: The default constructor, copy constructor, copy assignment operator, and destructor shall be non-trivial.

1.12 Complex Expression Rules:

- Rule 10-0-1: All arithmetic type conversions shall be explicit.

1.13 Run-time Environment Rules:

- Rule 22-0-1: All non-const variables of integer type should be explicitly initialized.

1.14 Testing and Validation Rules:

- Rule 25-0-1: Violations of a design rule that cannot be ascertained by the implementation of any of the test cases are unacceptable.

2 MISRA C++ – 2008 Standard

2.1 Mandatory Standard

Sl. No	Rule No	Violations	Sections
1	0-3-1	Minimization of run-time failures shall be ensured use of at least one of: a. Static analysis tools/techniques; b. Dynamic analysis tools/techniques; c. Explicit coding of checks to handle run-time faults	Runtime failures
2	0-4-1	User of scaled-integer or fixed-point arithmetic shall be documented	Arithmetic
3	0-4-2	Use of floating-point arithmetic shall be documented	Arithmetic
4	0-4-3	Floating-point implementation shall comply with a defined floating-point standard	Arithmetic
5	1-0-2	Multiple compilers shall only be used if they have a common, defined interface	Language
6	1-0-3	The implementation of integer division in the chosen compiler shall be determined and documented	Language
7	2-2-1	The character set and the corresponding encoding shall be documented	Character sets
8	7-4-1	All usage od assembler shall be documented	The *asm* declaration
9	9-6-1	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented	Bit-fields
10	15-0-1	Exceptions shall only be used for error handling	Exception handling - General
11	16-6-1	All uses of the #pragma directive shall be documented	Pragma directive
12	17-0-4	All library code shall conform the MISRA C++	Library introduction - General

2.2 Required Standard

Sl.No	Rule No	Violation	
1.	0-1-1	A Project shall not contain unreachable code	Unnecessary Constructs
2.	0-1-2	A Project shall not contain infeasible paths	
3.	0-1-3	A Project shall not contain unused variables	
4.	0-1-4	A Project shall not contain non-volatile POD variables having only use	
5.	0-1-5	A Project shall contain unused type declarations	
6.	0-1-6	A Project shall not contain instances of non-volatile variables being given values that are never subsequently used	
7.	0-1-7	The value returned by a function having a non- void return type that is not an overloaded operator shall always be used.	
8.	0-1-8	All functions with void return type shall have external side effect(s).	
9.	0-1-9	There shall be no dead code	
10.	0-1-10	Every defined functions shall be called at least once	
11.	0-1-11	There shall be no unused parameters (named or unnamed) in non-virtual functions	
12.	0-1-12	There shall be no unused parameters (named or unnamed) in the set of parameters	
13.	0-2-1	An object shall not be assigned to an overlapping object	Storage
14.	0-3-2	If a function generates error information, then that error information shall be tested	Run Time Failures
15.	1-0-1	All code shall confirm to ISO/IEC 14882:2003 “The C++ Standard Incorporating technical Corrigendum 1”	Language
16.	2-3-1	Trigraphs shall not be used	Trigraph Sequences
17.	2-7-1	The Character sequence /* shall not be used within a C-style comment	Comments
18.	2-7-2	Sections of code shall not be	

		"commented out" using C-style comments	
19.	2-10-1	Different Identifiers shall be typographical Unambiguous	Identifiers
20.	2-10-2	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope	
21.	2-10-3	A typedef name(including qualification, if any) shall be an unique identifier	
22.	2-10-4	A class, union or enum name (including qualification, if any) shall be an unique identifier	
23.	2-10-6	If an Identifier refers to a type, it shall not also refer to an object or a function in the same scope	
24.	2-13-1	Only those escape Sequences that are defined in ISO/IEC 14882:2003 shall be used	Literals
25.	2-13-2	Octal constants (other than zero) and octal escape sequences (other than "/0") shall not be used	
26.	2-13-3	A "U" suffix shall be applied to all octal/ Hexadecimal integer literals of unsigned types	
27.	2-13-4	Literals suffixes shall be upper case	
28.	2-13-5	Narrow and wide string literals shall not be concatenated	
29.	3-1-1	It shall be possible to include any header file in multiple translation units without violating the one definition rule	Declarations and Definitions
30.	3-1-2	Functions shall not be declared at block scope	
31.	3-1-3	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization	
32.	3-2-1	All declarations of an object/ function shall have compatible types	One Definition Rule
33.	3-2-2	The One Definition rule shall not be violated	
34.	3-2-3	A type, object/function that is used in multiple translation units shall be declared in one and only one file	
35.	3-2-4	An identifier with external linkage shall have exactly one definition	
36.	3-3-1	Objects or functions with external linkage shall be declared in a header	Declarative regions and scope

		file	
37.	3-3-2	If a function has an internal linkage then all redeclarations shall include the static storage class specifier	
38.	3-4-1	An Identifier declared to be an object / type shall be defined in a block that minimizes its visibility	Name Lookup
39.	3-9-1	The types used for an object, a function return type, or function parameter should be token-for-token identical in all declarations and re-declarations	Types
40.	3-9-3	The Underlying bit representations of floating-point values shall not be used	
41.	4-5-1	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, | |, ! , the Equality Operators ==, and !=, the unary & operator, and the conditional operator	Integral promotions
42.	4-5-2	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [ ] , the assignment operator = , the Equality Operators ==, and !=, the unary & operator, and the relational operator<, >, <=, >=	
43.	4-5-3	Expressions with type (plain)char and wchar_tshall not be used as operands to built-in operators other than the assignment operator =, the Equality Operators ==, and !=, the unary & operator	
44.	4-10-1	NULL Shall not used as an integer value	Pointers Conversions
45.	4-10-2	Literal zero (0) shall not be used as the NULL pointer constant	
46.	5-0-1	The value of an expression shall b ethe same under any order of evaluation that the standard permits	Expressions
47.	5-0-3	A cvalue expression shall not be implicitly converted to an different underlying type	
48.	5-0-4	An implicit integral conversion shall not change the signedness of the underlying type	

49.	5-0-5	There shall be no implicit floating integral conversions	
50.	5-0-6	An implicit integral or floating point conversion shall not reduce the size of the underlying type	
51.	5-0-7	There shall be no explicit floating integral conversions of a cvalue expression	
52.	5-0-8	An explicit integral or floating point conversion shall not increase the size of the underlying type of a cvalue expression	
53.	5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	
54.	5-0-10	If the bitwise operators ~ and << are applied to an operand withy an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand	
55.	5-0-11	The plain char type shall only be used for the storage and use of character values	
56.	5-0-12	signed char and unsigned char type shall only be used for the storage and use of numeric values	
57.	5-0-13	The condition of an if statement and the condition of an iteration statement shall have type bool	
58.	5-0-14	the first operand of a conditional operator shall have type bool	
59.	5-0-15	Array indexing shall be the only form of arithmetic	
60.	5-0-16	A pointer operand and any pointer resulting from the pointer arithmetic using that operand shall both address elements of same array	
61.	5-0-17	subtraction between pointers shall only be applied to pointers that address elements of the same array	
62.	5-0-18	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	
63.	5-0-19	The declaration of objects shall contain no more than two levels of pointer	

		indirection	
64.	5-0-20	Non-constant operands to a binary bit wise operator shall have the same underlying type	
65.	5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type.	
66.	5-2-1	Each operand of a logical && or | | shall be a postfix expression	postfix expressions
67.	5-2-2	A pointer to a virtual base class shall only be a cast to a pointer to a derived class by means of dynamic_cast	
68.	5-2-4	C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used	
69.	5-2-5	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	
70.	5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	
71.	5-2-7	An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly	
72.	5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type	
73.	5-2-11	The comma operator, && operator and the || operator shall not be overloaded.	
74.	5-2-12	An identifier with array type passed as a function argument shall not decay to a pointer.	
75.	5-3-1	Each operand of the ! operator, the logical && or the logical | | operators shall have type bool.	Unary expressions
76.	5-3-2	The Unary minus operator shall not applied to an expression whose underlying type is unsigned.	
77.	5-3-3	The unary & operator shall not be overloaded.	
78.	5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects.	
79.	5-8-1	The right hand operand of a shift	Shift Operators

		operator shall lie between zero andone less than the width in bits of the underlying type of the left handoperand.	
80.	5-14-1	The hand operand of a logical && or || operator shall not contain side effects	Logical AND operator
81.	5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.	Assignment operators
82.	5-18-1	The comma operator shall not be used.	comma operator
83.	6-2-1	Assignment operators shall not be used in sub-expressions.	Expression statement
84.	6-2-2	Floating-point expressions shall not be directly or indirectly testedfor equality or inequality.	
85.	6-2-3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the firstcharacter following the null statement is a white-space character.	
86.	6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Compound statement
87.	6-4-1	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compoundstatement, or another if statement.	Selection statements
88.	6-4-2	All if … else if constructs shall be terminated with an else clause.	
89.	6-4-3	A switch statement shall be a well-formed switch statement.	
90.	6-4-4	A switch-label shall only be used when the most closely-enclosingcompound statement is the body of a switch statement.	
91.	6-4-5	An unconditional throw or break statement shall terminate non-empty switch-clause.	
92.	6-4-6	The final clause of a switch statement shall be the default-clause.	
93.	6-4-7	The condition of a switch statement shall not have bool type.	
94.	6-4-8	Every switch statement shall have at	

		least one case-clause.	
95.	6-5-1	A for loop shall contain a single loop-counter which shall not have floating type.	Iteration statements
96.	6-5-2	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.	
97.	6-5-3	Theloop-counter shall not be modified within condition or statement.	
98.	6-5-4	The loop-counter shall be modified by one of: --, ++, -=n, or +=n;where n remains constant for the duration of the loop.	
99.	6-5-5	A loop-control-variable other than the loop- counter shall not be modified within condition or expression.	
100.	6-5-6	A loop-control-variable other than the loop- counter which is modified in statement shall have type bool.	
101.	6-6-1	Any label referenced by a goto statement shall be declared in thesame block, or in a block enclosing the goto statement.	Jump statements
102.	6-6-2	The goto statement shall jump to a label declared later in the samefunction body.	
103.	6-6-3	The continue statement shall only be used within a well-formed forloop.	
104.	6-6-4	For any iteration statement there shall be no more than one break orgoto statement used for loop termination.	
105.	6-6-5	A function shall have a single point of exit at the end of the function.	
106.	7-1-1	A variable which is not modified shall be const qualified.	Specifiers
107.	7-1-2	A pointer or reference parameter in a function shall be declared aspointer to const or reference to const if the corresponding object is not modified.	
108.	7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	Enumeration declarations
109.	7-3-1	The global namespace shall only contain main, namespace declarations	Namespaces

		and extern "C" declarations.	
110	7-3-2	The identifier main shall not be used for a function other than theglobal function main.	
111	7-3-3	There shall be no unnamed namespaces in header files.	
112	7-3-4	using-directives shall not be used.	
113	7-3-5	Multiple declarations for an identifier in the same namespace shallnot straddle a using-declaration for that identifier.	
114	7-3-6	using-directives and using declarations (excluding class scope or function scope using declarations) shall not be used in header	
115	7-4-2	Assembler language shall only be introduced using the asm declaration.	The asm declaration
116	7-4-3	Assembly language shall be encapsulated and isolated.	
117	7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Linkage specifications
118	7-5-2	The address of an object with automatic storage shall not be assignedto another object that may persist after the first object has ceased toexist.	
119	7-5-3	A function shall not return a reference or a pointer to a parameterthat is passed by reference or const reference.	
120	8-0-1	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.	Declarations-General
121	8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Meaning of declarators
122	8-4-1	Functions shall not be defined using the ellipsis notation	Function Definitions
123	8-4-2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	
124	8-4-3	All exit paths from a function with non-void return type shall have an explicit	

		return statement with an expression.	
125	8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	
126	8-5-1	All variables shall have a defined value before they are used	Initializers
127	8-5-2	Braces shall be used to indicate and match the structure in the non- zero initialization of arrays and structures.	
128	8-5-3	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.	
129	9-3-1	const member functions shall not return non-const pointers or references to class-data.	Member Functions
130	9-3-2	Member functions shall not return non-const handles to class-data.	
131	9-3-3	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.	
132	9-5-1	Unions shall not be used	Unions
133	9-6-2	Bit-fields shall be either bool type or an explicitly unsigned or signed integral type.	Bit Fields
134	9-6-3	Bit-fields shall not have enum type	
135	9-6-4	Named bit-fields with signed integer type shall have a length of more than one bit.	
136	10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy.	Multiple base classes
137	10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.	
138	10-3-1	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy.	Virtual functions
139	10-3-2	Each overriding virtual function shall be declared with the virtual keyword.	
140	10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.	

141	11-0-1	Member data in non-POD class types shall be private.	Member access control-General
142	12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor	Constructors
143	12-1-3	All constructors that are capable with a single argument of fundamental type shall be declared explicit.	
144	12-8-1	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.	Copying class objects
145	12-8-2	The copy assignment operator shall be declared protected or private in an abstract class.	
146	14-5-1	Anon-member generic function shall only be declared in a namespace that is not an associated namespace.	Template Declarations
147	14-5-2	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter	
148	14-5-3	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.	
149	14-6-1	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->	Name resolution
150	14-6-2	The function chosen by overload resolution shall resolve to a function declared previously in the translation unit.	
151	14-7-1	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once.	Template instantiation and specialization
152	14-7-2	For any given template specialization, an explicit instantiation of the template with the template-arguments used in the specialization shall not render the program ill-formed.	
153	14-7-3	All partial and explicit specializations for a template shall be declared	

		in the same file as the declaration of their primary template.	
154	14-8-1	Overloaded function templates shall not be explicitly specialized.	Function template and specialization
155	15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement.	Exception handling-General
156	15-1-1	The assignment-expression of a throw statement shall not itself cause an exception to be thrown	Throwing an exception
157	15-1-2	NULL shall not be thrown explicitly	
158	15-1-3	An empty throw (throw;) shall only be used in the compound- statement of a catch handler.	
159	15-3-1	Exceptions shall be raised only after start-up and before termination of the program.	Handling an exception
160	15-3-3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.	
161	15-3-4	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.	
162	15-3-5	A class type exception shall always be caught by reference	
163	15-3-6	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.	
164	15-3-7	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.	
165	15-4-1	If a function is declared with an exception-specification, then all declarations of the same function (in other translation units) shall be declared with the same set of type-ids.	Exception Specifications
166	15-5-1	A class destructor shall not exit with an exception.	Exception handling — Special functions
167	15-5-2	Where a function's declaration includes an exception-specification, the function	

		shall only be capable of throwing exceptions of the indicated type(s).	
168	15-5-3	The terminate() function shall not be called implicitly	
169	16-0-1	#include directives in a file shall only be preceded by other preprocessor directives or comments.	Preprocessing directives — General
170	16-0-2	Macros shall only be #define'd or #undef'd in the global namespace.	
171	16-0-3	#undef shall not be used	
172	16-0-4	Function-like macros shall not be defined	
173	16-0-5	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	
174	16-0-6	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##.	
175	16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.	
176	16-0-8	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.	
177	16-1-1	The defined preprocessor operator shall only be used in one of the two standard forms.	Conditional inclusion
178	16-1-2	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related	
179	16-2-1	The pre-processor shall only be used for file inclusion and include guards.	Source file inclusion
180	16-2-2	C++ macros shall only be used for: include guards, type qualifiers, or storage class specifiers.	
181	16-2-3	Include guards shall be provided	
182	16-2-4	The ', ", /* or // characters shall not occur in a header file name.	
183	16-2-6	The #include directive shall be followed by either a <filename> or "filename" sequence	
184	16-3-1	There shall be at most one occurrence of the # or ## operators in a	Macro replacement

		single macro definition.	
185	17-0-1	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.	Library Introduction- General
186	17-0-2	The names of standard library macros and objects shall not be reused.	
187	17-0-3	The names of standard library functions shall not be overridden	
188	17-0-5	The setjmp macro and the longjmp function shall not be used.	
189	18-0-1	The C library shall not be used	Language support library- General
190	18-0-2	The library functions atof, atoi and atol from library <cstdlib> shall not be used.	
191	18-0-3	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used	
192	18-0-4	The time handling functions of library <ctime> shall not be used	
193	18-0-5	The unbounded functions of library <cstring> shall not be used	
194	18-2-1	The macro offsetof shall not be used	Language support library — Implementation properties
195	18-4-1	Dynamic heap memory allocation shall not be used	Language support library — Dynamic memory management
196	18-7-1	The signal handling facilities of <csignal> shall not be used.	Language support library — Other runtime support
197	19-3-1	The error indicator errno shall not be used.	Diagnostics library — Error numbers
198	27-0-1	The stream input/output library <cstdio> shall not be used	Input/output library — General

2.3 Advisory Standard

Sl. No	Rule No	Violations	Sections
1	2-5-1	*Digraphs* should not be used	Alternative token)
2	2-7-3	Sections of code should not be “commented out” using C++ comments	Comments
3	2-10-5	The identifier name of a non-member	Identifiers

		object or function with static storage duration should not be reused	
4	3-9-2	Typedefs that indicate size and signedness should be used in place of the basic numerical types	Types
5	5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions.	Expressions
6	5-2-3	Casts from a base class to a derived class should not be performed on polymorphic types	Postfix expressions
7	5-2-9	A cast should not convert a pointer type to ab integer type	Postfix expressions
8	5-2-10	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression	Postfix expressions
9	5-19-1	Evaluation of constant unsigned integer expression should not lead to wrap-around	Constant expressions
10	7-5-4	Functions should not call themselves , either directly or indirectly	Linkage specifications
11	10-1-1	Classes should not be derived from virtual bases	Multiple base classes
12	10-2-1	All accessible entity names within a multiple inheritance hierarchy should be *unique*	Member name lookup
13	12-1-2	All constructors of a class should explicitly call a constructor for all of its immediate base class and all virtual base class	Constructors
14	14-8-2	The viable *function set* for a function call should either contain no function specializations, or only contain function specialization	Function template specialization
15	15-0-2	An exception object should not have pointer type	Exception handling - General
16	15-3-2	There should be at least one exception handler to catch all otherwise unhandled exceptions	Handling an exception
17	16-2-5	The \ character should not occur in a *header file* name.	Source file inclusion
18	16-3-2	The # and ## operators should not be used	Macro replacement

3 Conclusion

MISRA C++ standard are designed to ensure code quality, maintainability, and safety in C++ programming projects. Compliance with MISRA C++ rules is often a requirement in industries where safety-critical software is developed, similar to MISRA C for the C programming language.