MISRA C

Table of Contents

1 Introduction

MISRA C is a set of guidelines and rules for writing C code in a way that promotes safety, reliability, and portability. These rules are widely used in industries where software safety and security are critical, such as automotive, aerospace, and medical device development. Below are some key categories and examples of MISRA C rules:

1.1 Environment Rules:

- Rule 1.1: Code shall not be written in K&R style.
- Rule 1.2: Source file shall only include library headers and shall not include non-library headers.
- Rule 1.3: Unused macro parameters shall be named in the macro definition.

1.2 Language Extensions:

- Rule 6.3: The basic type of an object shall not be changed by an explicit cast.

1.3 Control Flow Rules:

- Rule 14.1: Only those values defined in the Standard, signed integer, or floating-point types shall be used as loop counters.
- Rule 14.3: The final clause of a switch statement shall be the default clause.

1.4 Pointer Rules:

- Rule 17.3: A pointer not used when returned from a function or passed to a function should be tested for nullness before dereferencing.

1.5 Numeric Types and Operations:

- Rule 22.2: The storage class of an object shall not be explicitly specified in more than one place.

1.6 Preprocessor Directives and Macros:

- Rule 19.1: Only those macros that are used in the same source file shall be defined in that file.

1.7 Runtime Fail-Safe:

- Rule 20.1: The #elif and #else preprocessor directives shall not be used.

1.8 Dynamic Memory Management:

- Rule 21.1: All uses of malloc and similar functions shall be matched with corresponding calls to free.

1.9 Complex and Compound Expressions:

- Rule 14.8: The if keyword shall be followed by a left parenthesis and the condition shall be enclosed in parentheses.

1.10 Error Handling and Diagnostics:

- Rule 14.10: The if keyword shall be followed by a left parenthesis and the condition shall be enclosed in parentheses.

2 MISRA C – 2012 Standard

2.1 Mandatory Standard

Rule	Rule Description
R.9.1	The value of an object with automatic storage duration shall not be read before it has been set.
R.13.6	The operand of the sizeof operator shall not contain any expression which has potential side effects
R.17.3	A function shall not be declared implicitly
R.17.4	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
R.17.6	The declaration of an array parameter shall not contain the static keyword between the [ ]
R.19.1	An object shall not be assigned or copied to an overlapping object
R.22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function

R.22.4	There shall be no attempt to write to a stream which has been opened as read-only
R.22.5	A pointer to a FILE object shall not be dereferenced
R.22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed

2.2 Required Standard

D- Directive

Rule	Rule Description
D.1.1	Any implementation-defined behavior on which the output of the program depends shall be documented and understood.
D.2.1	All source files shall compile without any compilation errors
D.3.1	All code shall be traceable to documented requirements
D.4.1	Run-time failures shall be minimized
D.4.3	Assembly language shall be encapsulated and isolated
D.4.7	If a function returns error information, then that error information shall be tested
D.4.10	Precautions shall be taken in order to prevent the contents of a header file being included more than once
D.4.11	The validity of values passed to library functions shall be checked
D.4.12	Dynamic memory allocation shall not be used
R.1.1	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits
R.1.3	There shall be no occurrence of undefined or critical unspecified behavior
R.2.1	A project shall not contain unreachable code
R.2.2	T here shall be no dead cod e
R.3.1	The character sequences / * and / / shall not be used within a comment

R.3.2	Line -splicing shall not be used in // comment s
R.4.1	Octal and hexadecimal escape sequences shall be terminated
R.5.1	External identifiers shall be distinct
R.5.2	Identifiers declared in the same scope and name space shall be distinct
R.5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
R.5.4	Macro identifiers shall be distinct
R.5.5	Identifiers shall be distinct from macro names
R.5.6	A type def name shall be a unique identifier
R.5.7	A tag name shall be a unique identifier
R.5.8	Identifiers that define objects or functions with external linkage shall be unique
R.6.1	Bi t - fields shall only be declared with an appropriate type
R.6.2	Single - bit named bit fields shall not be of a signed type
R.7.1	Octal constants shall not be used
R.7.2	A "u" or " U " suffix shall be applied to all integer constants that are represented in an un signed type
R.7.3	The lowercase character 'l' shall not be used in a literal suffix
R.7.4	A string literal shall not be assigned to an object unless the object ' s type is "pointer to const-qualified char"
R.8.1	Types shall be explicitly specified
R.8.2	Function types shall be in prototype form with named parameters
R.8.3	All declarations of an object or function shall use the same names and type qualifiers
R.8.4	A compatible declaration shall be visible when an object or function with external linkage is defined
R.8.5	An external object or function shall be declared once in one and only one file
R.8.6	An identifier with external linkage shall have exactly one external

	definition
R.8.8	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage
R.8.10	An inline function shall be declared with the static storage class
R.8.11	When an array with external linkage is declared, its size should be explicitly specified
R.8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique
R.8.14	The restrict type qualifier shall not be used
R.9.2	The 7esign7ize for an aggregate or union shall be enclosed in braces
R.9.3	Arrays shall not be partially initialized
R.9.4	An element of an object shall not be 7esign7ized more than once
R.9.5	Where designated initialisers are used to initialize an array object the size of the array shall be specified explicitly
R.10.1	Operands shall not be of an inappropriate essential type
R.10.2	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations
R.10.3	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category
R.10.4	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category
R.10.6	The value of a composite expression shall not be assigned to an object with wider essential type
R.10.7	If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand
R.10.8	The value of a composite expression shall not be cast to a different essential type category or a wider essential type
R.11.1	Conversions shall not be performed between a pointer to a function

	and any other type
R.11.2	Conversions shall not be performed between a pointer to incomplete and any other type
R.11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type
R.11.6	A cast shall not be performed between pointer to void and an arithmetic type
R.11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type
R.11.8	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer
R.11.9	The macro NULL shall be the only permitted form of integer null pointer constant
R.12.2	The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand
R.13.1	Initialiser lists shall not contain persistent side effects
R.13.2	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders
R.13.5	The right hand operand of a logical && or || operator shall not contain persistent side effects
R.14.1	A loop counter shall not have essentially floating type
R.14.2	A for loop shall be well-formed
R.14.3	Controlling expressions shall not be invariant
R.14.4	The controlling expression of an if statement and the controlling expression of an iteration - statement shall have essentially Boolean type
R.15.2	The goto statement shall jump to a label declared later in the same function
R.15.3	Any label referenced by a goto statement shall be declared in the

	same block , or in any block enclosing the goto statement
R.15.6	The body of an iteration - statement or a selection - statement shall be a compound statement
R.15.7	All if . . else if constructs shall be terminated with an else statement
R.16.1	All switch statements shall be well - formed
R.16.2	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement
R.16.3	An unconditional break statement shall terminate every switch-clause
R.16.4	Every switch statement shall have a default label
R.16.5	A default label shall appear as either the first or the last switch label of a switch statement
R.16.6	Every switch statement shall have at least two switch-clauses
R.16.7	A switch-expression shall not have essentially Boolean type
R.17.1	The features of shall not be used
R.17.2	Functions shall not call themselves, either directly or indirectly
R.17.7	The value returned by a function having non-void return type shall be used
R.18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand
R.18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array
R.18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object
R.18.6	The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist
R.18.7	Flexible array members shall not be declared
R.18.8	Variable-length array types shall not be used
R.20.2	The ‘, “ or \ characters and the /* or // character sequences shall not occur in a header file name

R.20.3	The #include directive shall be followed by either a or “filename” sequence
R.20.4	A macro shall not be defined with the same name as a keyword
R.20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument
R.20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses
R.20.8	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1
R.20.9	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define’d before evaluation
R.20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator
R.20.12	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators
R.20.13	A line whose first token is # shall be a valid preprocessing directive
R.20.14	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related
R.21.1	#define and #undef shall not be used on a reserved identifier or reserved macro name
R.21.2	A reserved identifier or macro name shall not be declared
R.21.3	The memory allocation and deallocation functions of <stdlib.h> shall not be used
R.21.4	The standard header file <setjmp.h> shall not be used
R.21.5	The standard header file <signal.h> shall not be used
R.21.6	The Standard Library input/output routines shall not be used.
R.21.7	The atof, atoi, atol and atoll functions of shall not be used
R.21.8	The library functions abort, exit, getenv and system of <stdlib.h>

R.21.9	The library functions bsearch and qsort of <stdlib.h> shall not be used
R.21.10	The Standard Library time and date routines shall not be used
R.21.11	The standard header file <tgmath.h>shall not be used
R.22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released
R.22.3	The same file shall not be open for read and write access at the same time on different streams

2.3 Advisory Standard

Rule	Rule Description
D.4.2	All usage of assembly language should be documented
D.4.4	Sections of code should not be 'commented out'
D.4.5	Identifiers in the same namespace with overlapping visibility should be typographically unambiguous
D.4.6	typedefs that indicate size and signedness should be used in place of the basic numerical types
D.4.7	If a pointer to a structure or union is n ever dereferenced within a translation unit , t h en the implementation of the object should be hidden
D.4.8	A function should be used in preference to a function –like macro where they are interchangeable
D.4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence
R.1.2	Language extensions should not be used
R.2.3	A project should not contain unused type declarations
R.2.4	A project should not contain unused tag declarations

R.2.5	A project should not contain unused macro declarations
R.2.6	A function should not contain unused label declarations
R.2.7	T here should be no unused parameters in functions
R.4.2	Trigraphs should not be used
R.5.9	Identifiers that define objects or functions with Internal linkage should be unique
R.8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
R.8.9	An object should be defined at block scope if its identifier only appears in a single function
R.8.13	A pointer should point to a const qualified type whenever possible
R.10.5	The value of an expression should not be cast to an in appropriate essential type
R.11.4	A conversion should not be performed between a pointer to object and an integer type
R.11.5	A conversion should not be performed from pointer to void into pointer to object
R.12.1	The precedence of operators within expressions should be made explicit
R.12.3	The comma operator should not be us e d
R.12.4	Evaluation of constant expressions should not lead to unsigned integer
R.13.3	A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator
R.13.4	The result of an assignment operator should not be used
R.15.1	The goto statement should not be used
R.15.4	T here should be no more than one break or goto statement used to terminate any iteration statement
R.15.5	A function should have a single point of exit at the end

R.17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements
R.17.8	A function parameter should not be modified
R.18.4	The +, -, += and -= operators should not be applied to an expression of a pointer type.
R.18.5	Declarations should contain no more than two levels of pointer nesting
R.19.2	The union keyword should not be used
R.20.1	#include directives should only be preceded by preprocessor directives or comments
R.20.5	#undef should not be used
R.20.10	The # and ## preprocessor operators should not be used
R.21.12	The exception handling features of <fenv.h> should not be used

3 Conclusion

MISRA C standard and they are designed to ensure code quality, maintainability, and safety in C programming projects. Developers and teams typically use static analysis tools and linters that check code against these rules to enforce compliance and catch potential issues early in the development process. Compliance with MISRA C rules is often a requirement in industries where safety-critical software is developed.