Polyspace Bug Finder

Coding Standards Report for Project: Misra_SelfTranning

Report Author: PC

Polyspace Bug Finder: Coding Standards Report for Project: Misra_SelfTranning

by Report Author: PC

Published 23-Nov-2023 20:06:17

Analysis Author(s): Anh-Embedded
Polyspace Version(s): Polyspace Bug Finder 3.2 (R2020a)
Project Version(s): 1.0

Result Folder(s):
C:\Users\PC\OneDrive\Documents\Polyspace_Workspace\Misra\Module_1\BF_Result

Table of Contents

Chapter 1. MISRA C:2012 Guidelines

MISRA C:2012 Guidelines Summary for all Files

File	Total
C:\Project\learnMisraC2012\isPowerOfFour.c	0
C:\Project\learnMisraC2012\isPowerOfFour.h	0
Total	**0**

MISRA C:2012 Guidelines Summary for Enabled Guidelines

No violations were found.

MISRA C:2012 Guidelines Violations

No violations were found.

Chapter 2. CERT C Coding Standard

CERT C Coding Standard Summary for all Files

File	Total
C:\Project\learnMisraC2012\isPowerOfFour.c	0
C:\Project\learnMisraC2012\isPowerOfFour.h	0
Total	**0**

CERT C Coding Standard Summary for Enabled Rules

No violations were found.

CERT C Coding Standard Violations

No violations were found.

Chapter 3. Appendix 1 - Configuration Settings

Polyspace Settings

Option	Value
-author	Anh-Embedded
-bug-finder	true
-c-version	c11
-cert-c	all
-compiler	armcc
-date	23/11/2023
-dos	true
-I	C:\Project\learnMisraC2012
-import-comments	C:\Users\PC\OneDrive\Documents\Polyspace_Workspace\Misra\Module_1\BF_Result\comments_bak
-lang	C
-misra3	all
-prog	Misra_SelfTranning
-results-dir	C:\Users\PC\OneDrive\Documents\Polyspace_Workspace\Misra\Module_1\BF_Result
-target	arm
-verif-version	1.0
-checkers	ALIGNMENT_CHANGE, ASSERT, ATOMIC_VAR_ACCESS_TWICE, ATOMIC_VAR_SEQUENCE_NOT_ATOMIC, BAD_EQUAL_EQUAL_USE, BAD_EQUAL_USE, BAD_FREE, BAD_LOCK, BAD_PTR_SCALING, BAD_UNLOCK, CHARACTER_MISUSE, CHAR_EOF_CONFUSED, CLOSED_RESOURCE_USE, CONSTANT_OBJECT_WRITE, DATA_RACE, DATA_RACE_STD_LIB, DEADLOCK, DEAD_CODE, DECL_MISMATCH, DOUBLE_DEALLOCATION, DOUBLE_LOCK, DOUBLE_RESOURCE_CLOSE, DOUBLE_RESOURCE_OPEN, DOUBLE_UNLOCK, ERRNO_MISUSE, FILE_OBJECT_MISUSE, FLEXIBLE_ARRAY_MEMBER_STRUCT_MISUSE, FLOAT_ABSORPTION, FLOAT_CONV_OVFL, FLOAT_STD_LIB, FLOAT_ZERO_DIV, FREED_PTR, FUNC_CAST, IMPROPER_ARRAY_INIT, INLINE_CONSTRAINT_NOT_RESPECTED, INT_CONV_OVFL, INT_STD_LIB, INT_ZERO_DIV, INVALID_ENV_POINTER, INVALID_MEMORY_ASSUMPTION, INVALID_VA_LIST_ARG, IO_INTERLEAVING, LOCAL_ADDR_ESCAPE, MACRO_USED_AS_OBJECT, MEMCMP_PADDING_DATA, MEMCMP_STRINGS, MEM_STD_LIB, MISSING_ERRNO_RESET, MISSING_NULL_CHAR, MISSING_RETURN, NON_INIT_PTR, NON_INIT_VAR, NON_POSITIVE_VLA_SIZE, NULL_PTR, OPERATOR_PRECEDENCE, OTHER_STD_LIB, OUT_BOUND_ARRAY, OUT_BOUND_PTR, PARTIALLY_ACCESSED_ARRAY, PRE_DIRECTIVE_MACRO_ARG, PRE_UCNAME_JOIN_TOKENS, PTR_CAST, PTR_SIZEOF_MISMATCH, PTR_TO_DIFF_ARRAY, PUTENV_AUTO_VAR, READ_ONLY_RESOURCE_WRITE, RESOURCE_LEAK, SIDE_EFFECT_IGNORED, SIGN_CHANGE, SIG_HANDLER_CALLING_SIGNAL, SIG_HANDLER_COMP_EXCP_RETURN, SIG_HANDLER_ERRNO_MISUSE, SIG_HANDLER_SHARED_OBJECT, SIZEOF_MISUSE, STD_FUNC_ARG_MISMATCH, STREAM_WITH_SIDE_EFFECT, STRING_FORMAT, STRLIB_BUFFER_OVERFLOW, STRLIB_BUFFER_UNDERFLOW, STR_FORMAT_BUFFER_OVERFLOW, STR_STD_LIB, TEMP_OBJECT_ACCESS, TOO_MANY_VA_ARG_CALLS, TYPEDEF

Option	Value
	_MISMATCH, UINT_CONV_OVFL, UNPROTOTYPED_FUNC_CALL, UNREACHABLE, USELESS_IF, USELESS_WRITE, VAR_SHADOWING, VA_ARG_INCORRECT_TYPE, VA_START_INCORRECT_TYPE, VA_START_MISUSE

Coding Standard Configuration

Table 3.1. MISRA C:2012 Guidelines Configuration

Guideline	Description	Mode	Comment	Enabled
D1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.	required	-	yes
D2.1	All source files shall compile without any compilation errors.	required	-	yes
D3.1	All code shall be traceable to documented requirements.	required	Not enforceable	no
D4.1	Run-time failures shall be minimized.	required	-	yes
D4.2	All usage of assembly language should be documented.	advisory	Not enforceable	no
D4.3	Assembly language shall be encapsulated and isolated.	required	-	yes
D4.4	Sections of code should not be "commented out".	advisory	Not implemented	no
D4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous.	advisory	-	yes
D4.6	typedefs that indicate size and signedness should be used in place of the basic numerical types.	advisory	-	yes
D4.7	If a function returns error information, then that error information shall be tested.	required	-	yes
D4.8	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden.	advisory	-	yes
D4.9	A function should be used in preference to a function-like macro where they are interchangeable.	advisory	-	yes
D4.10	Precautions shall be taken in order to prevent the contents of a header file being included more than once.	required	-	yes
D4.11	The validity of values passed to library functions shall be checked.	required	-	yes
D4.12	Dynamic memory allocation shall not be used.	required	-	yes
D4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.	advisory	-	yes
D4.14	The validity of values received from external sources shall be checked.	required	-	yes
1.1	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits.	required	-	yes
1.2	Language extensions should not be used.	advisory	-	yes

Guideline	Description	Mode	Comment	Enabled
1.3	There shall be no occurrence of undefined or critical unspecified behaviour.	required	-	yes
2.1	A project shall not contain unreachable code.	required	-	yes
2.2	There shall be no dead code.	required	-	yes
2.3	A project should not contain unused type declarations.	advisory	-	yes
2.4	A project should not contain unused tag declarations.	advisory	-	yes
2.5	A project should not contain unused macro declarations.	advisory	-	yes
2.6	A function should not contain unused label declarations.	advisory	-	yes
2.7	There should be no unused parameters in functions.	advisory	-	yes
3.1	The character sequences /* and // shall not be used within a comment.	required	-	yes
3.2	Line-splicing shall not be used in // comments.	required	-	yes
4.1	Octal and hexadecimal escape sequences shall be terminated.	required	-	yes
4.2	Trigraphs should not be used.	advisory	-	yes
5.1	External identifiers shall be distinct.	required	-	yes
5.2	Identifiers declared in the same scope and name space shall be distinct.	required	-	yes
5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	required	-	yes
5.4	Macro identifiers shall be distinct.	required	-	yes
5.5	Identifiers shall be distinct from macro names.	required	-	yes
5.6	A typedef name shall be a unique identifier.	required	-	yes
5.7	A tag name shall be a unique identifier.	required	-	yes
5.8	Identifiers that define objects or functions with external linkage shall be unique.	required	-	yes
5.9	Identifiers that define objects or functions with internal linkage should be unique.	advisory	-	yes
6.1	Bit-fields shall only be declared with an appropriate type.	required	-	yes
6.2	Single-bit named bit fields shall not be of a signed type.	required	-	yes
7.1	Octal constants shall not be used.	required	-	yes
7.2	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.	required	-	yes
7.3	The lowercase character "l" shall not be used in a literal suffix.	required	-	yes
7.4	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char ".	required	-	yes
8.1	Types shall be explicitly specified.	required	-	yes
8.2	Function types shall be in prototype form with named parameters.	required	-	yes

Guideline	Description	Mode	Comment	Enabled
8.3	All declarations of an object or function shall use the same names and type qualifiers.	required	-	yes
8.4	A compatible declaration shall be visible when an object or function with external linkage is defined.	required	-	yes
8.5	An external object or function shall be declared once in one and only one file.	required	-	yes
8.6	An identifier with external linkage shall have exactly one external definition.	required	-	yes
8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.	advisory	-	yes
8.8	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.	required	-	yes
8.9	An object should be defined at block scope if its identifier only appears in a single function.	advisory	-	yes
8.10	An inline function shall be declared with the static storage class.	required	-	yes
8.11	When an array with external linkage is declared, its size should be explicitly specified.	advisory	-	yes
8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.	required	-	yes
8.13	A pointer should point to a const-qualified type whenever possible.	advisory	-	yes
8.14	The restrict type qualifier shall not be used.	required	-	yes
9.1	The value of an object with automatic storage duration shall not be read before it has been set.	mandatory	-	yes
9.2	The initializer for an aggregate or union shall be enclosed in braces.	required	-	yes
9.3	Arrays shall not be partially initialized.	required	-	yes
9.4	An element of an object shall not be initialized more than once.	required	-	yes
9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.	required	-	yes
10.1	Operands shall not be of an inappropriate essential type.	required	-	yes
10.2	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.	required	-	yes
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.	required	-	yes
10.4	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.	required	-	yes
10.5	The value of an expression should not be cast to an inappropriate essential type.	advisory	-	yes
10.6	The value of a composite expression shall not be assigned to an object with wider essential type.	required	-	yes
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 shall not have wider essential type.	required	-	yes
10.8	The value of a composite expression shall not be cast to a different essential type category or a wider essential type.	required	-	yes

Guideline	Description	Mode	Comment	Enabled
11.1	Conversions shall not be performed between a pointer to a function and any other type.	required	-	yes
11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type.	required	-	yes
11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type.	required	-	yes
11.4	A conversion should not be performed between a pointer to object and an integer type.	advisory	-	yes
11.5	A conversion should not be performed from pointer to void into pointer to object.	advisory	-	yes
11.6	A cast shall not be performed between pointer to void and an arithmetic type.	required	-	yes
11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type.	required	-	yes
11.8	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.	required	-	yes
11.9	The macro NULL shall be the only permitted form of integer null pointer constant.	required	-	yes
12.1	The precedence of operators within expressions should be made explicit.	advisory	-	yes
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.	required	-	yes
12.3	The comma operator should not be used	advisory	-	yes
12.4	Evaluation of constant expressions should not lead to unsigned integer wrap-around.	advisory	-	yes
12.5	The sizeof operator shall not have an operand which is a function parameter declared as "array of type".	mandatory	-	yes
13.1	Initializer lists shall not contain persistent side effects.	required	-	yes
13.2	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.	required	-	yes
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.	advisory	-	yes
13.4	The result of an assignment operator should not be used.	advisory	-	yes
13.5	The right hand operand of a logical && or || operator shall not contain persistent side effects.	required	-	yes
13.6	The operand of the sizeof operator shall not contain any expression which has potential side effects.	mandatory	-	yes
14.1	A loop counter shall not have essentially floating type.	required	-	yes
14.2	A for loop shall be well-formed.	required	-	yes
14.3	Controlling expressions shall not be invariant.	required	-	yes
14.4	The controlling expression of an if statement and the controlling expression of an iteration-statement sha ll have essentially Boolean type.	required	-	yes
15.1	The goto statement should not be used.	advisory	-	yes
15.2	The goto statement shall jump to a label declared later in the same function.	required	-	yes
15.3	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing t he goto statement.	required	-	yes

Guideline	Description	Mode	Comment	Enabled
15.4	There should be no more than one break or goto statement used to terminate any iteration statement.	advisory	-	yes
15.5	A function should have a single point of exit at the end.	advisory	-	yes
15.6	The body of an iteration-statement or a selection-statement shall be a compound-statement.	required	-	yes
15.7	All if ... else if constructs shall be terminated with an else statement.	required	-	yes
16.1	All switch statements shall be well-formed.	required	-	yes
16.2	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	required	-	yes
16.3	An unconditional break statement shall terminate every switch-clause.	required	-	yes
16.4	Every switch statement shall have a default label.	required	-	yes
16.5	A default label shall appear as either the first or the last switch label of a switch statement.	required	-	yes
16.6	Every switch statement shall have at least two switch-clauses.	required	-	yes
16.7	A switch-expression shall not have essentially Boolean type.	required	-	yes
17.1	The features of <stdarg.h> shall not be used.	required	-	yes
17.2	Functions shall not call themselves, either directly or indirectly.	required	-	yes
17.3	A function shall not be declared implicitly.	mandatory	-	yes
17.4	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	mandatory	-	yes
17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.	advisory	-	yes
17.6	The declaration of an array parameter shall not contain the static keyword between the [ ].	mandatory	-	yes
17.7	The value returned by a function having non-void return type shall be used.	required	-	yes
17.8	A function parameter should not be modified.	advisory	-	yes
18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.	required	-	yes
18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	required	-	yes
18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object.	required	-	yes
18.4	The +, -, += and -= operators should not be applied to an expression of pointer type.	advisory	-	yes
18.5	Declarations should contain no more than two levels of pointer nesting.	advisory	-	yes
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.	required	-	yes
18.7	Flexible array members shall not be declared.	required	-	yes

Guideline	Description	Mode	Comment	Enabled
18.8	Variable-length array types shall not be used.	required	-	yes
19.1	An object shall not be assigned or copied to an overlapping object.	mandatory	-	yes
19.2	The union keyword should not be used.	advisory	-	yes
20.1	#include directives should only be preceded by preprocessor directives or comments.	advisory	-	yes
20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.	required	-	yes
20.3	The #include directive shall be followed by either a <filename> or "filename"sequence.	required	-	yes
20.4	A macro shall not be defined with the same name as a keyword.	required	-	yes
20.5	#undef should not be used.	advisory	-	yes
20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument.	required	-	yes
20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.	required	-	yes
20.8	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.	required	-	yes
20.9	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.	required	-	yes
20.10	The # and ## preprocessor operators should not be used.	advisory	-	yes
20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.	required	-	yes
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.	required	-	yes
20.13	A line whose first token is # shall be a valid preprocessing directive.	required	-	yes
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.	required	-	yes
21.1	#define and #undef shall not be used on a reserved identifier or reserved macro name.	required	-	yes
21.2	A reserved identifier or macro name shall not be declared.	required	-	yes
21.3	The memory allocation and deallocation functions of <stdlib.h> shall not be used.	required	-	yes
21.4	The standard header file <setjmp.h> shall not be used.	required	-	yes
21.5	The standard header file <signal.h> shall not be used.	required	-	yes
21.6	The Standard Library input/output functions shall not be used.	required	-	yes
21.7	The atof, atoi, atol, and atoll functions of <stdlib.h> shall not be used.	required	-	yes
21.8	The library functions abort, exit and system of <stdlib.h> shall not be used.	required	-	yes
21.9	The library functions bsearch and qsort of <stdlib.h> shall not be used.	required	-	yes
21.10	The Standard Library time and date functions shall not be used.	required	-	yes

Guideline	Description	Mode	Comment	Enabled
21.11	The standard header file <tgmath.h> shall not be used.	required	-	yes
21.12	The exception handling features of <fenv.h> should not be used.	advisory	-	yes
21.13	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF.	mandatory	-	yes
21.14	The Standard Library function memcmp shall not be used to compare null terminated strings.	required	-	yes
21.15	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types.	required	-	yes
21.16	The pointer arguments to the Standard Library function memcmp shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type.	required	-	yes
21.17	Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters.	mandatory	-	yes
21.18	The size_t argument passed to any function in <string.h> shall have an appropriate value.	mandatory	-	yes
21.19	The pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they have pointer to const-qualified type.	mandatory	-	yes
21.20	The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function.	mandatory	-	yes
22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.	required	-	yes
22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function.	mandatory	-	yes
22.3	The same file shall not be open for read and write access at the same time on different streams.	required	-	yes
22.4	There shall be no attempt to write to a stream which has been opened as read-only.	mandatory	-	yes
22.5	A pointer to a FILE object shall not be dereferenced.	mandatory	-	yes
22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	mandatory	-	yes
22.7	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF.	required	-	yes
22.8	The value of errno shall be set to zero prior to a call to an errno-setting-function.	required	-	yes
22.9	The value of errno shall be tested against zero after calling an errno-setting-function.	required	-	yes
22.10	The value of errno shall only be tested when the last function to be called was an errno-setting-function.	required	-	yes

Table 3.2. CERT C Coding Standard Configuration

Rule	Description	Mode	Comment	Enabled
PRE00-C	Prefer inline or static functions to function-like macros	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
PRE01-C	Use parentheses within macros around parameter names	recommendation	-	yes
PRE02-C	Macro replacement lists should be parenthesized	recommendation	Not implemented	no
PRE03-C	Prefer typedefs to defines for encoding non-pointer types	recommendation	Not implemented	no
PRE04-C	Do not reuse a standard header file name	recommendation	Not implemented	no
PRE05-C	Understand macro replacement when concatenating tokens or performing stringification	recommendation	Not implemented	no
PRE06-C	Enclose header files in an inclusion guard	recommendation	-	yes
PRE07-C	Avoid using repeated question marks	recommendation	-	yes
PRE08-C	Guarantee that header file names are unique	recommendation	Not implemented	no
PRE09-C	Do not replace secure functions with deprecated or obsolescent functions	recommendation	-	yes
PRE10-C	Wrap multistatement macros in a do-while loop	recommendation	-	yes
PRE11-C	Do not conclude macro definitions with a semicolon	recommendation	-	yes
PRE12-C	Do not define unsafe macros	recommendation	Not implemented	no
PRE13-C	Use the Standard predefined macros to test for versions and features.	recommendation	Not implemented	no
PRE30-C	Do not create a universal character name through concatenation	rule	-	yes
PRE31-C	Avoid side effects in arguments to unsafe macros	rule	-	yes
PRE32-C	Do not use preprocessor directives in invocations of function-like macros	rule	-	yes
DCL00-C	Const-qualify immutable objects	recommendation	Not implemented	no
DCL01-C	Do not reuse variable names in subscopes	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
DCL02-C	Use visually distinct identifiers	recommendation	-	yes
DCL03-C	Use a static assertion to test the value of a constant expression	recommendation	Not implemented	no
DCL04-C	Do not declare more than one variable per declaration	recommendation	Not implemented	no
DCL05-C	Use typedefs of non-pointer types only	recommendation	Not implemented	no
DCL06-C	Use meaningful symbolic constants to represent literal values	recommendation	-	yes
DCL07-C	Include the appropriate type information in function declarators	recommendation	-	yes
DCL08-C	Properly encode relationships in constant definitions	recommendation	Not implemented	no
DCL09-C	Declare functions that return errno with a return type of errno_t	recommendation	Not implemented	no
DCL10-C	Maintain the contract between the writer and caller of variadic functions	recommendation	-	yes
DCL11-C	Understand the type issues associated with variadic functions	recommendation	-	yes
DCL12-C	Implement abstract data types using opaque types	recommendation	-	yes
DCL13-C	Declare function parameters that are pointers to values not changed by the function as const	recommendation	-	yes
DCL15-C	Declare file-scope objects or functions that do not need external linkage as static	recommendation	-	yes
DCL16-C	Use 'L,' not 'l,' to indicate a long value	recommendation	-	yes
DCL17-C	Beware of miscompiled volatile-qualified variables	recommendation	Not implemented	no
DCL18-C	Do not begin integer constants with 0 when specifying a decimal value	recommendation	-	yes
DCL19-C	Minimize the scope of variables and functions	recommendation	-	yes
DCL20-C	Explicitly specify void when a function accepts no arguments	recommendation	Not implemented	no

Rule	Description	Mode	Comment	Enabled
DCL21-C	Understand the storage of compound literals	recommendation	Not implemented	no
DCL22-C	Use volatile for data that cannot be cached	recommendation	-	yes
DCL23-C	Guarantee that mutually visible identifiers are unique	recommendation	-	yes
DCL30-C	Declare objects with appropriate storage durations	rule	-	yes
DCL31-C	Declare identifiers before using them	rule	-	yes
DCL36-C	Do not declare an identifier with conflicting linkage classifications	rule	-	yes
DCL37-C	Do not declare or define a reserved identifier	rule	-	yes
DCL38-C	Use the correct syntax when declaring a flexible array member	rule	-	yes
DCL39-C	Avoid information leakage in structure padding	rule	-	yes
DCL40-C	Do not create incompatible declarations of the same function or object	rule	-	yes
DCL41-C	Do not declare variables inside a switch statement before the first case label	rule	-	yes
EXP00-C	Use parentheses for precedence of operation	recommendation	-	yes
EXP02-C	Be aware of the short-circuit behavior of the logical AND and OR operators	recommendation	Not implemented	no
EXP03-C	Do not assume the size of a structure is the sum of the sizes of its members	recommendation	Not implemented	no
EXP05-C	Do not cast away a const qualification	recommendation	-	yes
EXP07-C	Do not diminish the benefits of constants by assuming their values in expressions	recommendation	Not implemented	no
EXP08-C	Ensure pointer arithmetic is used correctly	recommendation	-	yes
EXP09-C	Use sizeof to determine the size of a type or variable	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
EXP10-C	Do not depend on the order of evaluation of subexpressions or the order in which side effects take place	recommendation	-	yes
EXP11-C	Do not make assumptions regarding the layout of structures with bit-fields	recommendation	Not implemented	no
EXP12-C	Do not ignore values returned by functions	recommendation	-	yes
EXP13-C	Treat relational and equality operators as if they were nonassociative	recommendation	-	yes
EXP15-C	Do not place a semicolon on the same line as an if, for, or while statement	recommendation	-	yes
EXP16-C	Do not compare function pointers to constant values	recommendation	Not implemented	no
EXP19-C	Use braces for the body of an if, for, or while statement	recommendation	-	yes
EXP20-C	Perform explicit tests to determine success, true and false, and equality	recommendation	Not implemented	no
EXP30-C	Do not depend on the order of evaluation for side effects	rule	-	yes
EXP32-C	Do not access a volatile object through a nonvolatile reference	rule	-	yes
EXP33-C	Do not read uninitialized memory	rule	-	yes
EXP34-C	Do not dereference null pointers	rule	-	yes
EXP35-C	Do not modify objects with temporary lifetime	rule	-	yes
EXP36-C	Do not cast pointers into more strictly aligned pointer types	rule	-	yes
EXP37-C	Call functions with the correct number and type of arguments	rule	-	yes
EXP39-C	Do not access a variable through a pointer of an incompatible type	rule	-	yes
EXP40-C	Do not modify constant objects	rule	-	yes
EXP42-C	Do not compare padding data	rule	-	yes

Rule	Description	Mode	Comment	Enabled
EXP43-C	Avoid undefined behavior when using restrict-qualified pointers	rule	-	yes
EXP44-C	Do not rely on side effects in operands to sizeof, _Alignof, or _Generic	rule	-	yes
EXP45-C	Do not perform assignments in selection statements	rule	-	yes
EXP46-C	Do not use a bitwise operator with a Boolean-like operand	rule	-	yes
EXP47-C	Do not call va_arg with an argument of the incorrect type	rule	-	yes
INT00-C	Understand the data model used by your implementation(s)	recommendation	-	yes
INT01-C	Use rsize_t or size_t for all integer values representing the size of an object	recommendation	Not implemented	no
INT02-C	Understand integer conversion rules	recommendation	-	yes
INT04-C	Enforce limits on integer values originating from tainted sources	recommendation	-	yes
INT05-C	Do not use input functions to convert character data if they cannot handle all possible inputs	recommendation	Not implemented	no
INT07-C	Use only explicitly signed or unsigned char type for numeric values	recommendation	-	yes
INT08-C	Verify that all integer values are in range	recommendation	-	yes
INT09-C	Ensure enumeration constants map to unique values	recommendation	-	yes
INT10-C	Do not assume a positive remainder when using the % operator	recommendation	-	yes
INT12-C	Do not make assumptions about the type of a plain int bit-field when used in an expression	recommendation	-	yes
INT13-C	Use bitwise operators only on unsigned operands	recommendation	-	yes
INT14-C	Avoid performing bitwise and arithmetic operations on the same data	recommendation	-	yes
INT15-C	Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types	recommendation	Not implemented	no

Rule	Description	Mode	Comment	Enabled
INT16-C	Do not make assumptions about representation of signed integers	recommendation	Not implemented	no
INT17-C	Define integer constants in an implementation-independent manner	recommendation	Not implemented	no
INT18-C	Evaluate integer expressions in a larger size before comparing or assigning to that size	recommendation	-	yes
INT30-C	Ensure that unsigned integer operations do not wrap	rule	-	yes
INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data	rule	-	yes
INT32-C	Ensure that operations on signed integers do not result in overflow	rule	-	yes
INT33-C	Ensure that division and remainder operations do not result in divide-by-zero errors	rule	-	yes
INT34-C	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	rule	-	yes
INT35-C	Use correct integer precisions	rule	-	yes
INT36-C	Converting a pointer to integer or integer to pointer	rule	-	yes
FLP00-C	Understand the limitations of floating-point numbers	recommendation	-	yes
FLP01-C	Take care in rearranging floating-point expressions	recommendation	Not implemented	no
FLP02-C	Avoid using floating-point numbers when precise computation is needed	recommendation	-	yes
FLP03-C	Detect and handle floating-point errors	recommendation	-	yes
FLP04-C	Check floating-point inputs for exceptional values	recommendation	Not implemented	no
FLP05-C	Do not use denormalized numbers	recommendation	Not implemented	no
FLP06-C	Convert integers to floating point for floating-point operations	recommendation	-	yes
FLP07-C	Cast the return value of a function that returns a floating-point type	recommendation	Not implemented	no

Rule	Description	Mode	Comment	Enabled
FLP30-C	Do not use floating-point variables as loop counters	rule	-	yes
FLP32-C	Prevent or detect domain and range errors in math functions	rule	-	yes
FLP34-C	Ensure that floating-point conversions are within range of the new type	rule	-	yes
FLP36-C	Preserve precision when converting integral values to floating-point type	rule	-	yes
FLP37-C	Do not use object representations to compare floating-point values	rule	-	yes
ARR00-C	Understand how arrays work	recommendation	Not implemented	no
ARR01-C	Do not apply the sizeof operator to a pointer when taking the size of an array	recommendation	-	yes
ARR02-C	Explicitly specify array bounds, even if implicitly defined by an initializer	recommendation	-	yes
ARR30-C	Do not form or use out-of-bounds pointers or array subscripts	rule	-	yes
ARR32-C	Ensure size arguments for variable length arrays are in a valid range	rule	-	yes
ARR36-C	Do not subtract or compare two pointers that do not refer to the same array	rule	-	yes
ARR37-C	Do not add or subtract an integer to a pointer to a non-array object	rule	-	yes
ARR38-C	Guarantee that library functions do not form invalid pointers	rule	-	yes
ARR39-C	Do not add or subtract a scaled integer to a pointer	rule	-	yes
STR00-C	Represent characters using an appropriate type	recommendation	Not implemented	no
STR01-C	Adopt and implement a consistent plan for managing strings	recommendation	Not implemented	no
STR02-C	Sanitize data passed to complex subsystems	recommendation	-	yes
STR03-C	Do not inadvertently truncate a string	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
STR04-C	Use plain char for characters in the basic character set	recommendation	Not implemented	no
STR05-C	Use pointers to const when referring to string literals	recommendation	Not implemented	no
STR06-C	Do not assume that strtok() leaves the parse string unchanged	recommendation	Not implemented	no
STR07-C	Use the bounds-checking interfaces for string manipulation	recommendation	-	yes
STR09-C	Don't assume numeric values for expressions with type plain character	recommendation	Not implemented	no
STR10-C	Do not concatenate different type of string literals	recommendation	Not implemented	no
STR11-C	Do not specify the bound of a character array initialized with a string literal	recommendation	-	yes
STR30-C	Do not attempt to modify string literals	rule	-	yes
STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator	rule	-	yes
STR32-C	Do not pass a non-null-terminated character sequence to a library function that expects a string	rule	-	yes
STR34-C	Cast characters to unsigned char before converting to larger integer sizes	rule	-	yes
STR37-C	Arguments to character-handling functions must be representable as an unsigned char	rule	-	yes
STR38-C	Do not confuse narrow and wide character strings and functions	rule	-	yes
MEM00-C	Allocate and free memory in the same module, at the same level of abstraction	recommendation	-	yes
MEM01-C	Store a new value in pointers immediately after free()	recommendation	-	yes
MEM02-C	Immediately cast the result of a memory allocation function call into a pointer to the allocated type	recommendation	-	yes
MEM03-C	Clear sensitive information stored in reusable resources	recommendation	-	yes
MEM04-C	Beware of zero-length allocations	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
MEM05-C	Avoid large stack allocations	recommendation	-	yes
MEM06-C	Ensure that sensitive data is not written out to disk	recommendation	-	yes
MEM07-C	Ensure that the arguments to calloc(), when multiplied, do not wrap	recommendation	Not implemented	no
MEM10-C	Define and use a pointer validation function	recommendation	Not implemented	no
MEM11-C	Do not assume infinite heap space	recommendation	-	yes
MEM12-C	Consider using a goto chain when leaving a function on error when using and releasing resources	recommendation	-	yes
MEM30-C	Do not access freed memory	rule	-	yes
MEM31-C	Free dynamically allocated memory when no longer needed	rule	-	yes
MEM33-C	Allocate and copy structures containing a flexible array member dynamically	rule	-	yes
MEM34-C	Only free memory allocated dynamically	rule	-	yes
MEM35-C	Allocate sufficient memory for an object	rule	-	yes
MEM36-C	Do not modify the alignment of objects by calling realloc()	rule	-	yes
FIO01-C	Be careful using functions that use file names for identification	recommendation	Not implemented	no
FIO02-C	Canonicalize path names originating from tainted sources	recommendation	-	yes
FIO03-C	Do not make assumptions about fopen() and file creation	recommendation	Not implemented	no
FIO05-C	Identify files using multiple file attributes	recommendation	Not implemented	no
FIO06-C	Create files with appropriate access permissions	recommendation	Not implemented	no
FIO08-C	Take care when calling remove() on an open file	recommendation	Not implemented	no

Rule	Description	Mode	Comment	Enabled
FIO09-C	Be careful with binary data when transferring data across systems	recommendation	Not implemented	no
FIO10-C	Take care when using the rename() function	recommendation	Not implemented	no
FIO11-C	Take care when specifying the mode parameter of fopen()	recommendation	-	yes
FIO13-C	Never push back anything other than one read character	recommendation	Not implemented	no
FIO14-C	Understand the difference between text mode and binary mode with file streams	recommendation	Not implemented	no
FIO15-C	Ensure that file operations are performed in a secure directory	recommendation	Not implemented	no
FIO17-C	Do not rely on an ending null character when using fread()	recommendation	Not implemented	no
FIO18-C	Never expect fwrite() to terminate the writing process at a null character	recommendation	Not implemented	no
FIO19-C	Do not use fseek() and ftell() to compute the size of a regular file	recommendation	Not implemented	no
FIO20-C	Avoid unintentional truncation when using fgets() or fgetws()	recommendation	Not implemented	no
FIO21-C	Do not create temporary files in shared directories	recommendation	-	yes
FIO22-C	Close files before spawning processes	recommendation	Not implemented	no
FIO23-C	Do not exit with unflushed data in stdout or stderr	recommendation	Not implemented	no
FIO24-C	Do not open a file that is already open	recommendation	-	yes
FIO30-C	Exclude user input from format strings	rule	-	yes
FIO32-C	Do not perform operations on devices that are only appropriate for files	rule	-	yes
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	rule	-	yes
FIO37-C	Do not assume that fgets() or fgetws() returns a nonempty string when successful	rule	-	yes
FIO38-C	Do not copy a FILE object	rule	-	yes
FIO39-C	Do not alternately input and output from a stream without an intervening flush or positioning call	rule	-	yes
FIO40-C	Reset strings on fgets() or fgetws() failure	rule	-	yes

Rule	Description	Mode	Comment	Enabled
FIO41-C	Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects	rule	-	yes
FIO42-C	Close files when they are no longer needed	rule	-	yes
FIO44-C	Only use values for fsetpos() that are returned from fgetpos()	rule	-	yes
FIO45-C	Avoid TOCTOU race conditions while accessing files	rule	-	yes
FIO46-C	Do not access a closed file	rule	-	yes
FIO47-C	Use valid format strings	rule	-	yes
ENV01-C	Do not make assumptions about the size of an environment variable	recommendation	-	yes
ENV02-C	Beware of multiple environment variables with the same effective name	recommendation	Not implemented	no
ENV03-C	Sanitize the environment when invoking external programs	recommendation	Not implemented	no
ENV30-C	Do not modify the object referenced by the return value of certain functions	rule	-	yes
ENV31-C	Do not rely on an environment pointer following an operation that may invalidate it	rule	-	yes
ENV32-C	All exit handlers must return normally	rule	-	yes
ENV33-C	Do not call system()	rule	-	yes
ENV34-C	Do not store pointers returned by certain functions	rule	-	yes
SIG00-C	Mask signals handled by noninterruptible signal handlers	recommendation	Not implemented	no
SIG01-C	Understand implementation-specific details regarding signal handler persistence	recommendation	Not implemented	no
SIG02-C	Avoid using signals to implement normal functionality	recommendation	Not implemented	no
SIG30-C	Call only asynchronous-safe functions within signal handlers	rule	-	yes
SIG31-C	Do not access shared objects in signal handlers	rule	-	yes
SIG34-C	Do not call signal() from within interruptible signal handlers	rule	-	yes
SIG35-C	Do not return from a computational exception signal handler	rule	-	yes
ERR00-C	Adopt and implement a consistent and comprehensive error-handling policy	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
ERR01-C	Use ferror() rather than errno to check for FILE stream errors	recommendation	Not implemented	no
ERR02-C	Avoid in-band error indicators	recommendation	Not implemented	no
ERR03-C	Use runtime-constraint handlers when calling the bounds-checking interfaces	recommendation	Not implemented	no
ERR04-C	Choose an appropriate termination strategy	recommendation	Not implemented	no
ERR05-C	Application-independent code should provide error detection without dictating error handling	recommendation	Not implemented	no
ERR06-C	Understand the termination behavior of assert() and abort()	recommendation	Not implemented	no
ERR07-C	Prefer functions that support error checking over equivalent functions that don't	recommendation	Not implemented	no
ERR30-C	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	rule	-	yes
ERR32-C	Do not rely on indeterminate values of errno	rule	-	yes
ERR33-C	Detect and handle standard library errors	rule	-	yes
ERR34-C	Detect errors when converting a string to a number	rule	-	yes
API00-C	Functions should validate their parameters	recommendation	Not implemented	no
API01-C	Avoid laying out strings in memory directly before sensitive data	recommendation	Not implemented	no
API02-C	Functions that read or write to or from an array should take an argument to specify the source or target size	recommendation	Not implemented	no
API03-C	Create consistent interfaces and capabilities across related functions	recommendation	Not implemented	no
API04-C	Provide a consistent and usable error-checking mechanism	recommendation	-	yes
API05-C	Use conformant array parameters	recommendation	Not implemented	no
API07-C	Enforce type safety	recommendation	Not implemented	no

Rule	Description	Mode	Comment	Enabled
API08-C	Avoid parameter names in a function prototype	recommendation	Not implemented	no
API09-C	Compatible values should have the same type	recommendation	Not implemented	no
API10-C	APIs should have security options enabled by default	recommendation	Not implemented	no
CON01-C	Acquire and release synchronization primitives in the same module, at the same level of abstraction	recommendation	-	yes
CON02-C	Do not use volatile as a synchronization primitive	recommendation	Not implemented	no
CON03-C	Ensure visibility when accessing shared variables	recommendation	Not implemented	no
CON04-C	Join or detach threads even if their exit status is unimportant	recommendation	Not implemented	no
CON05-C	Do not perform operations that can block while holding a lock	recommendation	-	yes
CON06-C	Ensure that every mutex outlives the data it protects	recommendation	Not implemented	no
CON07-C	Ensure that compound operations on shared variables are atomic	recommendation	Not implemented	no
CON08-C	Do not assume that a group of calls to independently atomic methods is atomic	recommendation	Not implemented	no
CON09-C	Avoid the ABA problem when using lock-free algorithms	recommendation	Not implemented	no
CON30-C	Clean up thread-specific storage	rule	-	yes
CON31-C	Do not destroy a mutex while it is locked	rule	-	yes
CON32-C	Prevent data races when accessing bit-fields from multiple threads	rule	-	yes
CON33-C	Avoid race conditions when using library functions	rule	-	yes
CON34-C	Declare objects shared between threads with appropriate storage durations	rule	-	yes
CON35-C	Avoid deadlock by locking in a predefined order	rule	-	yes

Rule	Description	Mode	Comment	Enabled
CON36-C	Wrap functions that can spuriously wake up in a loop	rule	-	yes
CON37-C	Do not call signal() in a multithreaded program	rule	-	yes
CON38-C	Preserve thread safety and liveness when using condition variables	rule	-	yes
CON39-C	Do not join or detach a thread that was previously joined or detached	rule	-	yes
CON40-C	Do not refer to an atomic variable twice in an expression	rule	-	yes
CON41-C	Wrap functions that can fail spuriously in a loop	rule	-	yes
CON42-C	Don't allow attackers to influence environment variables that control concurrency parameters	rule	Not implemented	no
CON43-C	Do not allow data races in multithreaded code	rule	-	yes
MSC00-C	Compile cleanly at high warning levels	recommendation	Not implemented	no
MSC01-C	Strive for logical completeness	recommendation	-	yes
MSC04-C	Use comments consistently and in a readable fashion	recommendation	-	yes
MSC05-C	Do not manipulate time_t typed values directly	recommendation	Not implemented	no
MSC06-C	Beware of compiler optimizations	recommendation	Not implemented	no
MSC09-C	Character encoding: Use subset of ASCII for safety	recommendation	Not implemented	no
MSC10-C	Character encoding: UTF8-related issues	recommendation	Not implemented	no
MSC11-C	Incorporate diagnostic tests using assertions	recommendation	Not implemented	no
MSC12-C	Detect and remove code that has no effect or is never executed	recommendation	-	yes
MSC13-C	Detect and remove unused values	recommendation	-	yes

Rule	Description	Mode	Comment	Enabled
MSC14-C	Do not introduce unnecessary platform dependencies	recommendation	Not implemented	no
MSC15-C	Do not depend on undefined behavior	recommendation	-	yes
MSC17-C	Finish every set of statements associated with a case label with a break statement	recommendation	-	yes
MSC18-C	Be careful while handling sensitive data, such as passwords, in program code	recommendation	-	yes
MSC19-C	For functions that return an array, prefer returning an empty array over a null value	recommendation	Not implemented	no
MSC20-C	Do not use a switch statement to transfer control into a complex block	recommendation	-	yes
MSC21-C	Use robust loop termination conditions	recommendation	-	yes
MSC22-C	Use the setjmp(), longjmp() facility securely	recommendation	-	yes
MSC23-C	Beware of vendor-specific library and language differences	recommendation	Not implemented	no
MSC24-C	Do not use deprecated or obsolescent functions	recommendation	-	yes
MSC30-C	Do not use the rand() function for generating pseudorandom numbers	rule	-	yes
MSC32-C	Properly seed pseudorandom number generators	rule	-	yes
MSC33-C	Do not pass invalid data to the asctime() function	rule	-	yes
MSC37-C	Ensure that control never reaches the end of a non-void function	rule	-	yes
MSC38-C	Do not treat a predefined identifier as an object if it might only be implemented as a macro	rule	-	yes
MSC39-C	Do not call va_arg() on a va_list that has an indeterminate value	rule	-	yes
MSC40-C	Do not violate constraints	rule	-	yes
MSC41-C	Never hard code sensitive information	rule	-	yes

Rule	Description	Mode	Comment	Enabled
POS01-C	Check for the existence of links when dealing with files	recommendation	Not implemented	no
POS02-C	Follow the principle of least privilege	recommendation	Not implemented	no
POS04-C	Avoid using PTHREAD_MUTEX_NORMAL type mutex locks	recommendation	Not implemented	no
POS05-C	Limit access to files by creating a jail	recommendation	-	yes
POS30-C	Use the readlink() function properly	rule	-	yes
POS33-C	Do not use vfork()	rule	-	yes
POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument	rule	-	yes
POS35-C	Avoid race conditions while checking for the existence of a symbolic link	rule	-	yes
POS36-C	Observe correct revocation order while relinquishing privileges	rule	-	yes
POS37-C	Ensure that privilege relinquishment is successful	rule	-	yes
POS38-C	Beware of race conditions when using fork and file descriptors	rule	-	yes
POS39-C	Use the correct byte ordering when transferring data between systems	rule	-	yes
POS44-C	Do not use signals to terminate threads	rule	-	yes
POS47-C	Do not use threads that can be canceled asynchronously	rule	-	yes
POS48-C	Do not unlock or destroy another POSIX thread's mutex	rule	-	yes
POS49-C	When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed	rule	-	yes
POS50-C	Declare objects shared between POSIX threads with appropriate storage durations	rule	-	yes
POS51-C	Avoid deadlock with POSIX threads by locking in predefined order	rule	-	yes

Rule	Description	Mode	Comment	Enabled
POS52-C	Do not perform operations that can block while holding a POSIX lock	rule	-	yes
POS53-C	Do not use more than one mutex for concurrent waiting operations on a condition variable	rule	-	yes
POS54-C	Detect and handle POSIX library errors	rule	-	yes
WIN00-C	Be specific when dynamically loading libraries	recommendation	-	yes
WIN01-C	Do not forcibly terminate execution	recommendation	Not implemented	no
WIN02-C	Restrict privileges when spawning child processes	recommendation	Not implemented	no
WIN03-C	Understand HANDLE inheritance	recommendation	Not implemented	no
WIN04-C	Consider encrypting function pointers	recommendation	Not implemented	no
WIN30-C	Properly pair allocation and deallocation functions	rule	-	yes

Chapter 4. Appendix 2 - Definitions

Table 4.1. Abbreviations

Abbreviation	Definition
NA	Not Available