Rule Enforcement Report

This report lists all rules, grouping by "Advisory" or "Required" and whether enforced.

Enforced Rules - Advisory

Rule Id	Rule	Enforcement
5.5	No object or function identifier with static storage duration should be reused.	1525, 1526, 1527, 1528, 1529
5.6	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.	0780, 0781
6.3	Typedefs that indicate size and signedness should be used in place of the basic numerical types.	5013
11.3	A cast should not be performed between a pointer type and an integral type.	0303, 0305, 0306, 0360, 0361, 0362
11.4	A cast should not be performed between a pointer to object type and a different pointer to object type.	0310, 0316, 0317
12.1	Limited dependence should be placed on C's operator precedence rules in expressions.	3389, 3391, 3392, 3393, 3394, 3395, 3396, 3397
12.6	The operands of logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, ||, !, =, ==, != and ?:).	3322, 3377, 4101, 4102, 4103, 4104, 4105, 4106, 4107, 4108, 4109, 4110, 4111, 4112, 4113, 4114, 4115, 4116
12.11	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	2910
12.13	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.	3440
13.2	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	3344, 4528, 4529, 4538, 4539, 4548, 4549, 4558, 4559, 4568, 4569
16.7	A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.	3673
17.5	The declaration of objects should contain no more than 2 levels of pointer indirection.	3260, 3261, 3262, 3263
19.1	#include statements in a file should only be preceded by other preprocessor directives or comments.	5087
19.2	Non-standard characters should not occur in header file names in #include directives.	0813, 0814, 0831
19.7	A function should be used in preference to a function-like macro.	3453
19.13	The # and ## preprocessor operators should not be used.	0341, 0342

Count of enforced Advisory rules = 16

Unenforced Rules - Advisory

Rule Id	Rule
5.7	No identifier name should be reused.

Count of unenforced Advisory rules = 1

Not Statically Enforceable Rules - Advisory

Rule Id	Rule
1.5	Floating-point implementations should comply with a defined floating-point standard.
2.4	Sections of code should not be 'commented out'.
3.3	The implementation of integer division in the chosen compiler should be determined, documented and taken into account.

Count of not statically enforceable Advisory rules = 3

Enforced Rules - Required

Rule Id	Rule	Enforcement
1.1	All code shall conform to ISO/IEC 9899:1990 C programming language, ISO	0180, 0232, 0233, 0240, 0241, 0244,

	9899, amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2: 1996192	0246, 0261, 0320, 0321, 0322, 0338, 0410, 0422, 0423, 0426, 0427, 0429, 0430, 0431, 0432, 0435, 0436, 0437, 0446, 0447, 0448, 0449, 0450, 0451, 0452, 0453, 0454, 0456, 0457, 0458, 0466, 0467, 0468, 0469, 0476, 0477, 0478, 0481, 0482, 0483, 0484, 0485, 0486, 0487, 0493, 0494, 0495, 0496, 0513, 0514, 0515, 0536, 0537, 0540, 0541, 0542, 0546, 0547, 0550, 0551, 0554, 0555, 0556, 0557, 0558, 0559, 0560, 0561, 0562, 0563, 0564, 0565, 0580, 0588, 0589, 0590, 0591, 0601, 0604, 0605, 0609, 0611, 0612, 0614, 0616, 0617, 0618, 0619, 0620, 0621, 0622, 0627, 0628, 0629, 0631, 0633, 0635, 0638, 0639, 0640, 0641, 0642, 0643, 0644, 0645, 0646, 0647, 0649, 0650, 0651, 0653, 0655, 0656, 0657, 0659, 0660, 0662, 0664, 0665, 0671, 0673, 0674, 0675, 0677, 0682, 0683, 0684, 0685, 0690, 0698, 0699, 0708, 0709, 0715, 0736, 0737, 0738, 0739, 0746, 0747, 0755, 0756, 0757, 0758, 0766, 0767, 0768, 0774, 0775, 0801, 0802, 0803, 0804, 0810, 0811, 0812, 0821, 0828, 0830, 0831, 0834, 0835, 0844, 0845, 0850, 0851, 0852, 0856, 0857, 0858, 0859, 0866, 0873, 0875, 0877, 0899, 0930, 0940, 0941, 0943, 0944, 0945, 1001, 1002, 1003, 1006, 1008, 1011, 1012, 1014, 1015, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1033, 1034, 1035, 1036, 1037, 1038, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1051, 1052, 1053, 1054, 1055, 1056, 3236, 3237, 3238, 3244, 3664
1.2	No reliance shall be placed on undefined or unspecified behaviour.	0160, 0161, 0162, 0163, 0164, 0165, 0166, 0167, 0168, 0169, 0170, 0171, 0172, 0173, 0174, 0175, 0176, 0177, 0178, 0179, 0184, 0185, 0186, 0190, 0191, 0192, 0193, 0194, 0195, 0196, 0197, 0198, 0199, 0200, 0201, 0203, 0204, 0206, 0207, 0208, 0235, 0275, 0301, 0302, 0304, 0307, 0309, 0337, 0400, 0401, 0402, 0403, 0475, 0543, 0544, 0545, 0602, 0623, 0625, 0626, 0630, 0632, 0636, 0654, 0658, 0661, 0667, 0668, 0672, 0676, 0678, 0680, 0706, 0745, 0777, 0779, 0809, 0813, 0814, 0836, 0837, 0848, 0853, 0854, 0864, 0865, 0867, 0872, 0874, 0885, 0887, 0888, 0914, 0915, 0942, 1509, 1510, 2800, 2810, 2820, 2830, 2840, 3113, 3114, 3239, 3311, 3312, 3319, 3437, 3438
2.1	Assembly language shall be encapsulated and isolated.	3006
2.2	Source code shall only use C-style comments.	1011
2.3	The character sequence /* shall not be used within a comment.	3108
3.1	All usage of implementation-defined behaviour shall be documented.	0202, 0284, 0285, 0286, 0287, 0288, 0289, 0292, 0299, 0303, 0305, 0306, 0308, 0581, 0634, 0878, 2850, 2851, 2852, 2853, 2855, 2856, 2857, 2860, 2861, 2862, 2890, 2895
3.4	All uses of the #pragma directive shall be documented and explained.	3116
4.1	Only those escape sequences that are defined in the ISO C standard shall be used.	0235, 3610
4.2	Trigraphs shall not be used.	3601

5.1	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.	0777, 0779
5.2	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.	2547, 3334
5.3	A typedef name shall be a unique identifier.	1506, 1507, 1508, 3448
5.4	A tag name shall be a unique identifier.	0547
6.1	The plain char type shall be used only for the storage and use of character values.	1810, 1811, 1812, 1813, 4401, 4421, 4431, 4441, 4451, 4510, 4511, 4512, 4513, 4514, 4517, 4518, 4519
6.2	Signed and unsigned char type shall be used only for the storage and use of numeric values.	4413, 4414
6.4	Bit fields shall only be defined to be of type unsigned int or signed int.	0634, 0635
6.5	Bit fields of signed type shall be at least 2 bits long.	3659, 3660, 3665
7.1	Octal constants (other than zero) and octal escape sequences shall not be used.	0336, 0339, 3628
8.1	Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.	3002, 3335, 3450
8.2	Whenever an object or function is declared or defined, its type shall be explicitly stated.	2050, 2051
8.3	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.	0624, 1331, 1332, 1333, 3320, 3675
8.4	If objects or functions are declared more than once their types shall be compatible.	0626, 0627, 0628, 1510
8.5	There shall be no definitions of objects or functions in a header file.	3406, 3480
8.6	Functions shall be declared at file scope.	3221
8.7	Objects shall be defined at block scope if they are only accessed from within a single function.	1514, 3218
8.8	An external object or function shall be declared in one and only one file.	1513, 3222, 3408, 3447, 3451
8.9	An identifier with external linkage shall have exactly one external definition.	0630, 1509
8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.	1504, 1505
8.11	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.	3224
8.12	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.	3684
9.1	All automatic variables shall have been assigned a value before being used.	2883, 2961, 2962, 2971, 2972
9.2	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.	0686, 0693, 0694
9.3	In an enumerator list, the '=' construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.	0723
10.1	The value of an expression of integer type shall not be implicitly converted to a different underlying type if: a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression	0570, 0572, 1256, 1257, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1317, 1461, 1800, 1802, 1803, 1804, 1810, 1811, 1812, 1813, 1820, 1821, 1822, 1823, 1824, 1830, 1831, 1832, 1833, 1834, 1840, 1841, 1842, 1843, 1844, 1850, 1851, 1852, 1853, 1854, 1860, 1861, 1862, 1863, 1864, 1880, 1881, 1882, 1890, 1891, 2900, 4117, 4401, 4402, 4403, 4404, 4405, 4410, 4412, 4413, 4414, 4415, 4420, 4421, 4422, 4423, 4424, 4425, 4430, 4431, 4432, 4434, 4435, 4436, 4437, 4440, 4441, 4442, 4443, 4445, 4446, 4447, 4451, 4460, 4461, 4463, 4464, 4470, 4471, 4480, 4481, 4490, 4491, 4500, 4501, 4502, 4503, 4504, 4505, 4507
10.2	The value of an expression of floating type shall not be implicitly converted to a different type if: a) it is not a conversion to a wider floating type, or b) the expression is complex, or c) the expression is a function argument, or d) the expression is a return expression	1264, 1265, 1266, 1880, 1881, 1882, 1892, 1893, 1894, 1895, 4450, 4452, 4453, 4454, 4462, 4465, 4472, 4482, 4492
10.3	The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.	4118, 4390, 4391, 4393, 4394
10.4	The value of a complex expression of floating type shall only be cast to a floating type that is narrower or of the same size.	4392, 4395
10.5	If the bitwise operators ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	4397, 4398, 4399, 4498, 4499

10.6	A "U" suffix shall be applied to all constants of unsigned type.	1281
11.1	Conversions shall not be performed between a pointer to a function and any type other than an integral type.	0302, 0307, 0313
11.2	Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.	0301
11.5	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.	0311, 0312
12.2	The value of an expression shall be the same under any order of evaluation that the standard permits.	0400, 0401, 0402, 0403
12.3	The sizeof operator shall not be used on expressions that contain side effects.	3307
12.4	The right hand operand of a logical && or || operator shall not contain side effects.	3415
12.5	The operands of a logical && or || shall be primary-expressions.	3398, 3399, 3400
12.7	Bitwise operators shall not be applied to operands whose underlying type is signed.	4532, 4533
12.8	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	0499, 2790
12.9	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	3101, 3102
12.10	The comma operator shall not be used.	3417, 3418
12.12	The underlying bit representations of floating-point values shall not be used.	3629
13.1	Assignment operators shall not be used in expressions that yield a Boolean value.	3326
13.3	Floating-point expressions shall not be tested for equality or inequality.	3341
13.4	The controlling expression of a for statement shall not contain any objects of floating type.	3340, 3342
13.5	The three expressions of a for statement shall be concerned only with loop control.	2462, 2463
13.6	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	2469
13.7	Boolean operations whose results are invariant shall not be permitted.	2990, 2991, 2992, 2993, 2994, 2995, 2996
14.1	There shall be no unreachable code.	0594, 1460, 1503, 2008, 2742, 2744, 2880, 2882, 3219
14.2	All non-null statements shall either (i) have at least one side effect however executed, or (ii) cause control flow to change.	3110, 3112, 3425, 3426, 3427
14.3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.	3138
14.4	The goto statement shall not be used.	2001
14.5	The continue statement shall not be used.	0770
14.6	For any iteration statement there shall be at most one break statement used for loop termination.	0771
14.7	A function shall have a single point of exit at the end of the function.	2889
14.8	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	2212, 2214
14.9	An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	2212, 2214, 3402
14.10	All if ... else if constructs shall be terminated with an else clause.	2004
15.0	The MISRA C switch syntax shall be used.	3234
15.1	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	2019
15.2	An unconditional break statement shall terminate every non-empty switch clause.	2003, 2020
15.3	The final clause of a switch statement shall be the default clause.	2002, 2009
15.4	A switch expression shall not represent a value that is effectively Boolean.	0735
15.5	Every switch statement shall have at least one case clause.	3315
16.1	Functions shall not be defined with a variable number of arguments.	1337
16.2	Functions shall not call themselves, either directly or indirectly.	1520, 3670
16.3	Identifiers shall be given for all of the parameters in a function prototype declaration.	1335, 1336
16.4	The identifiers used in the declaration and definition of a function shall be identical.	1330
16.5	Functions with no parameters shall be declared and defined with the parameter list void.	3001, 3007
16.6	The number of arguments passed to a function shall match the number of	0422, 0423, 3319

	parameters.	
16.8	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	0745, 2887, 2888, 3113, 3114
16.9	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.	3635
16.10	If a function returns error information, then that error information shall be tested.	3200
17.1	Pointer arithmetic shall only be applied to pointers that address an array or array element.	2930, 2931, 2932
17.2	Pointer subtraction shall only be applied to pointers that address elements of the same array.	2771, 2772
17.3	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.	2771, 2772
17.4	Array indexing shall be the only allowed form of pointer arithmetic.	0488, 0489, 0491, 0492
17.6	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	3217, 3225, 3230, 4140
18.1	All structure and union types shall be complete at the end of a translation unit.	0544, 0545, 0623, 0636
18.2	An object shall not be assigned to an overlapping object.	2776, 2777
18.4	Unions shall not be used.	0750, 0759
19.3	The #include directive shall be followed by either a <filename> or "filename" sequence.	0809
19.4	C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.	3409, 3411, 3412, 3413, 3431, 3452, 3458, 3460, 3461
19.5	Macros shall not be #define'd or #undef'd within a block.	0842
19.6	#undef shall not be used.	0841
19.8	A function-like macro shall not be invoked without all of its arguments.	0850, 0856
19.9	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	0853
19.10	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 ##.	3410
19.11	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.	3332
19.12	There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition.	0880, 0881, 0884
19.14	The defined preprocessor operator shall only be used in one of the two standard forms.	0885, 0887, 0888
19.15	Precautions shall be taken in order to prevent the contents of a header file being included twice.	0883
19.16	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.	3115
19.17	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.	3317, 3318
20.1	Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined.	0836, 0848, 0854, 3439, 4600, 4601
20.2	The names of standard library macros, objects and functions shall not be reused.	0602, 4602, 4603, 4604, 4605, 4606, 4607, 4608
20.3	The validity of values passed to library functions shall be checked.	2810, 2811, 2812, 2840, 2841, 2842
20.4	Dynamic heap memory allocation shall not be used.	5118
20.5	The error indicator errno shall not be used.	5119
20.6	The macro offsetof, in library <stddef.h>, shall not be used.	5120
20.7	The setjmp macro and the longjmp function shall not be used.	5122
20.8	The signal handling facilities of <signal.h> shall not be used.	5123
20.9	The input/output library <stdio.h> shall not be used in production code.	5124
20.10	The library functions atof, atoi and atol from library <stdlib.h> shall not be used.	5125
20.11	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used.	5126
20.12	The time handling functions of library <time.h> shall not be used.	5127
21.1	Minimisation of run-time failures shall be ensured by the 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	2791, 2792, 2801, 2802, 2811, 2812, 2821, 2822, 2831, 2832, 2841, 2842, 2845, 2846, 2847, 2871, 2872, 2877, 2891, 2892, 2896, 2897, 2901, 2902, 2911, 2912, 2980, 2981, 2982, 2983, 2984, 2985, 2986

Count of enforced Required rules = 116

Not Statically Enforceable Rules - Required

Rule Id	Rule
1.3	Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.
1.4	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.
3.2	The character set and the corresponding encoding shall be documented.
3.5	If it is being relied upon, the implementation-defined behaviour and packing of bitfields shall be documented.
3.6	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.
18.3	An area of memory shall not be reused for unrelated purposes.

Count of not statically enforceable Required rules = 6

Rule Enforcement Summary

(a) Total Number of Rules	**142**
(b) Total Number of 'Not Statically Enforceable' Rules	**9**
(c) Total Number of Enforceable Rules (a-b)	**133**
(d) Total Number of Enforced Rules	**132**
(e) Total Number of Unenforced Rules (c-d)	**1**
(f) Enforced Rules Percentage (d/c)	**99 %**
(g) Unenforced Rules Percentage (e/c)	**1 %**

End of report