BARR-C

Perforce QAC for C 2026.1

(BARR-C https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard)

Rule Enforcement Summary

		Total
a	Total Number of Rules	160
b	Total Number of 'Not Statically Enforceable' Rules (Assisted/Unassisted)	137
c	Total Number of Enforceable Rules (a-b)	23
d	Total Number of Enforced Rules	19
e	Total Number of Unenforced Rules (c-d)	4
f	Enforced Rules Percentage (d/c)	83%
g	Unenforced Rules Percentage (e/c)	17%

Id	Description	Enforced
BARR1-2-a	The width of all lines in a program shall be limited to a maximum of 80 characters.	Yes
BARR1-3-a	Braces shall always surround the blocks of code following if, else, switch, while, do, and for statements.	Yes
BARR1-3-b	Each left brace ({) shall appear by itself on the line below the start of the block it opens. The corresponding right brace (}) shall appear by itself in the same position the appropriate number of lines later in the file.	Yes
BARR1-7-a	The auto keyword shall not be used.	Yes
BARR1-7-b	The register keyword shall not be used.	Yes
BARR2-1-a	Single-line comments in the C++ style are a useful and acceptable alternative to traditional C style comments.	No
BARR2-1-b	Comments shall never contain the preprocessor tokens.	Yes
BARR3-5-a	The tab character (ASCII 0x09) shall never appear within any source code file.	Yes

Id	Description	Enforced
BARR3-6-a	Whenever possible, all source code lines shall end only with the single character 'LF' (ASCII 0x0A), not with the pair 'CR'-'LF' (0x0D 0x0A).	Yes
BARR3-6-b	The only other non-printable character permitted in a source code file is the form feed character 'FF' (ASCII 0x0C).	Yes
BARR4-1-a	All module names shall consist entirely of lowercase letters, numbers, and underscores. No spaces shall appear within the module's header and source file names.	No
BARR4-1-b	All module names shall be unique in their first 8 characters and end with suffices .h and .c for the header and source file names.	Yes
BARR4-1-c	No module header file name shall share the name of a header file from the C Standard Library or C++ Standard Library.	Yes
BARR4-1-d	Any module containing a main() function shall have the word "main" as part of its source file name.	No
BARR4-3-e	Each source file shall be free of unused include files.	No
BARR5-1-b	All new structures, unions, and enumerations shall be named via a typedef.	Yes
BARR5-2-b	The keywords short and long shall not be used.	Yes
BARR5-3-a	Bit-fields shall not be defined within signed integer types.	Yes
BARR5-3-b	None of the bitwise operators shall be used to manipulate signed integer data.	Yes
BARR5-3-c	Signed integers shall not be combined with unsigned integers in comparisons or expressions.	Yes
BARR7-2-a	All variables shall be initialized before use.	Yes
BARR8-5-a	The use of goto statements shall be restricted as per Rule 1.7.c.	Yes
BARR8-5-b	C Standard Library functions abort(), exit(), setjmp(), and longjmp() shall not be used.	Yes

Non-Automated (Not Statically Enforceable Rules)

Id	Description	Assisted
BARR1-1-a	All programs shall be written to comply with the C99 version of the ISO C Programming Language Standard.	Assisted

Id	Description	Assisted
BARR1-1-b	Whenever a C++ compiler is used, appropriate compiler options shall be set to restrict the language to the selected version of ISO C.	Unassisted
BARR1-1-c	The use of proprietary compiler language keyword extensions, #pragma and inline assembly shall be kept to the minimum necessary to get the job done.	Assisted
BARR1-1-d	Preprocessor directive #define shall not be used to alter or rename any keyword or other aspect of the programming language.	Assisted
BARR1-4-a	Do not rely on C operator precedence rules, as they may not be obvious to those who maintain the code. Use parentheses to ensure proper execution order within a sequence of operations.	Assisted
BARR1-4-b	Unless it is a single identifier or constant, each operand of the logical AND (&&) and logical OR (||) operators shall be surrounded by parentheses.	Assisted
BARR1-5-a	Abbreviations and acronyms should generally be avoided unless their meanings are widely and consistently understood in the engineering community.	Unassisted
BARR1-5-b	A table of project-specific abbreviations and acronyms shall be maintained in a version-controlled document.	Unassisted
BARR1-6-a	Each cast shall feature an associated comment describing how the code ensures proper behavior across the range of possible values on the right side.	Assisted
BARR1-7-c	It is a preferred practice to avoid all use of the goto keyword.	Assisted
BARR1-7-d	It is a preferred practice to avoid all use of the continue keyword.	Assisted
BARR1-8-a	The static keyword shall be used to declare all functions and variables that do not need to be visible outside of the module in which they are declared.	Assisted
BARR1-8-b_i	The const keyword shall be used to declare variables that should not be changed after initialization.	Assisted
BARR1-8-b_ii	The const keyword shall be used to define call-by-reference function parameters that should not be modified.	Assisted
BARR1-8-b_iii	The const keyword shall be used to define fields in a struct or union that should not be modified.	Unassisted
BARR1-8-b_iv	The const keyword shall be used as a strongly typed alternative to #define for numerical constants.	Unassisted
BARR1-8-c_i	The volatile keyword shall be used to declare a global variable accessible by any interrupt service routine.	Unassisted

Id	Description	Assisted
BARR1-8-c_ii	The volatile keyword shall be used to declare a global variable accessible by two or more threads.	Unassisted
BARR1-8-c_iii	The volatile keyword shall be used to declare a pointer to a memory-mapped I/O peripheral register set.	Unassisted
BARR1-8-c_iv	The volatile keyword shall be used to declare a delay loop counter.	Unassisted
BARR2-1-c_i	Code shall never be commented out, even temporarily.	Assisted
BARR2-1-c_ii	Any line or block of code that exists specifically to increase the level of debug output information shall be surrounded by #ifndef NDEBUG ... #endif.	Assisted
BARR2-2-a	All comments shall be written in clear and complete sentences, with proper spelling and grammar and appropriate punctuation.	Unassisted
BARR2-2-b	The most useful comments generally precede a block of code that performs one step of a larger algorithm. A blank line shall follow each such code block. The comments in front of the block should be at the same indentation level.	Unassisted
BARR2-2-c	end-of-line comments should only be used where the meaning of that one line of code may be unclear from the variable and function names and operations alone.	Unassisted
BARR2-2-d	The number and length of individual comment blocks shall be proportional to the complexity of the code they describe.	Unassisted
BARR2-2-e	Whenever an algorithm or technical detail is defined in an external reference, a comment shall include a sufficient reference to the original source to allow a reader of the code to locate the document.	Unassisted
BARR2-2-f	Whenever a flow chart or other diagram is needed to sufficiently document the code, the drawing shall be maintained with the source code under version control and the comments should reference the diagram by file name or title.	Unassisted
BARR2-2-g	All assumptions shall be spelled out in comments.	Unassisted
BARR2-2-h	Each module and function shall be commented in a manner suitable for automatic documentation generation.	Unassisted
BARR2-2-i_i	Use the following capitalized comment markers "WARNING:" to highlight alerts a maintainer there is risk in changing this code.	Unassisted
BARR2-2-i_ii	Use the following capitalized comment markers "NOTE:" to highlight descriptive comments about the "why" of a chunk of code—as distinguished from the "how" usually placed in comments.	Unassisted

Id	Description	Assisted
BARR2-2-i_iii	Use the following capitalized comment markers "TODO:" to highlight an area of the code is still under construction and explains what remains to be done.	Unassisted
BARR3-1-a	Each of the keywords if, while, for, switch, return when there is additional program text on the same line.	Unassisted
BARR3-1-b	Each of the assignment operators =, +=, -=, *=, /=, %=, &=, |=, ^=, ~=, != shall always be preceded and followed by one space.	Assisted
BARR3-1-c	Each of the binary operators +, -, *, /, %, <, <=, >, >=, ==,!=, <<, >>, &, |, ^, && shall always be preceded and followed by one space.	Assisted
BARR3-1-d	Each of the unary operators +, -, ++, --, ! , ~ shall be written without a space on the operand side.	Assisted
BARR3-1-e	The pointer operators * and & shall be written with white space on each side within declarations but otherwise without a space on the operand side.	Unassisted
BARR3-1-f	The ? and : characters that comprise the ternary operator shall always be preceded and followed by one space.	Unassisted
BARR3-1-g	The structure pointer and structure member operators -> and . shall be written without a space on the operand side.	Unassisted
BARR3-1-h	The left and right brackets of the array subscript operator ([and]) shall be without surrounding spaces, except as required by another white space rule.	Unassisted
BARR3-1-i	Expressions within parentheses shall always have no spaces adjacent to the left and right parenthesis characters.	Unassisted
BARR3-1-j	The left and right parentheses of the function call operator shall always be without surrounding spaces, except that the function declaration shall feature one space between the function name and the left parenthesis to allow that one particular mention of the function name to be easily located.	Unassisted
BARR3-1-k	Except when at the end of a line, each comma separating function parameters shall always be followed by one space.	Unassisted
BARR3-1-l	Each semicolon separating the elements of a for statement shall always be followed by one space.	Unassisted
BARR3-1-m	Each semicolon shall follow the statement it terminates without a preceding space.	Unassisted
BARR3-2-a	The names of variables within a series of declarations shall have their first characters aligned.	Assisted
BARR3-2-b	The names of struct, union members shall have their first characters aligned.	Unassisted

Id	Description	Assisted
BARR3-2-c	The assignment operators within a block of adjacent assignment statements shall be aligned.	Unassisted
BARR3-2-d	The # in a preprocessor directive shall always be located at the start of a line, though the directives themselves may be indented within a #if or #ifdef sequence.	Assisted
BARR3-3-a	No line of code shall contain more than one statement.	Assisted
BARR3-3-b	There shall be a blank line before and after each natural block of code.	Unassisted
BARR3-3-c	Each source file shall terminate with a comment marking the end of file followed by a blank line.	Assisted
BARR3-4-a	Each indentation level should align at a multiple of 4 characters from the start of the line.	Assisted
BARR3-4-b	Within a switch statement, the case labels shall be aligned and the contents of each case block shall be indented once from there.	Assisted
BARR3-4-c	Whenever a line of code is too long to fit within the maximum line width, indent the second and any subsequent lines in the most readable manner possible.	Unassisted
BARR4-2-a	There shall always be precisely one header file for each source file and they shall always have the same root name.	Unassisted
BARR4-2-b	Each header file shall contain a preprocessor guard against multiple inclusion.	Assisted
BARR4-2-c_i	The header file shall identify only the procedures, constants, and data types about which it is strictly necessary for other modules to be informed. It is a preferred practice that no variable ever be declared (via extern) in a header file.	Assisted
BARR4-2-c_ii	The header file shall identify only the procedures, constants, and data types about which it is strictly necessary for other modules to be informed. No storage for any variable shall be allocated in a header file.	Assisted
BARR4-2-d	No public header file shall contain a #include of any private header file.	Unassisted
BARR4-3-a	Each source file shall include only the behaviors appropriate to control one "entity".	Unassisted
BARR4-3-b	Each source file shall be comprised of some or all of the following sections, in the order listed: comment block, include statements, data type, constant, and macro definitions, static data declarations, private function prototypes, public function bodies, then private function bodies.	Unassisted
BARR4-3-c	Each source file shall always #include the header file of the same name, to allow the compiler to confirm that each public function and its prototype match.	Assisted

Id	Description	Assisted
BARR4-3-d	Absolute paths shall not be used in include file names.	Assisted
BARR4-3-f	No source file shall #include another source file.	Assisted
BARR4-4-a	A set of templates for header files and source files shall be maintained at the project level.	Unassisted
BARR5-1-a	The names of all new data types, shall consist only of lowercase characters and internal underscores and end with '_t'.	Assisted
BARR5-1-c	The name of all public data types shall be prefixed with their module name and an underscore.	Unassisted
BARR5-2-a	Whenever the width of an integer value matters in the program, one of the fixed width data types shall be used in place of char, short, int, long, or long long.	Assisted
BARR5-2-c	Use of the keyword char shall be restricted to the declaration of and operations concerning strings.	Assisted
BARR5-4-a	Avoid the use of floating point constants and variables whenever possible.	Assisted
BARR5-4-b_i	Use the C99 type names for fixed point datatypes.	Assisted
BARR5-4-b_ii	Append an 'f' to all single-precision constants.	Assisted
BARR5-4-b_iii	Ensure that the compiler supports double precision.	Unassisted
BARR5-4-b_iv	Never test for equality or inequality of floating point values.	Assisted
BARR5-4-b_v	Always invoke the isfinite() macro to check that prior calculations have resulted in neither INFINITY nor NAN.	Unassisted
BARR5-5-a	Appropriate care shall be taken to prevent the compiler from inserting padding bytes within struct or union types used to communicate to or from a peripheral or over a bus or network to another processor.	Assisted
BARR5-5-b	Appropriate care shall be taken to prevent the compiler from altering the intended order of the bits within bit-fields.	Assisted
BARR5-6-a	Boolean variables shall be declared as type bool.	Assisted
BARR5-6-b	Non-Boolean values shall be converted to Boolean via use of relational operators, not via casts.	Assisted
BARR6-1-a	No procedure shall have a name that is a keyword of any standard version of the C or C++ programming language.	Assisted

Id	Description	Assisted
BARR6-1-b	No procedure shall have a name that overlaps a function in the C Standard Library.	Assisted
BARR6-1-c	No procedure shall have a name that begins with an underscore.	Assisted
BARR6-1-d	No procedure name shall be longer than 31 characters.	Assisted
BARR6-1-e	No function name shall contain any uppercase letters.	Assisted
BARR6-1-f	No macro name shall contain any lowercase letters.	Assisted
BARR6-1-g	Underscores shall be used to separate words in procedure names.	Unassisted
BARR6-1-h	Each procedure's name shall be descriptive of its purpose.	Unassisted
BARR6-1-i	The names of all public functions shall be prefixed with their module name and an underscore.	Unassisted
BARR6-2-a	All reasonable effort shall be taken to keep the length of each function limited to one printed page, or a maximum of 100 lines.	Assisted
BARR6-2-b	Whenever possible, all functions shall be made to start at the top of a printed page, except when several small functions can fit onto a single page.	Unassisted
BARR6-2-c	It is a preferred practice that all functions shall have just one exit point and it shall be via a return at the bottom of the function.	Assisted
BARR6-2-d	A prototype shall be declared for each public function in the module header file.	Assisted
BARR6-2-e	All private functions shall be declared static.	Assisted
BARR6-2-f	Each parameter shall be explicitly declared and meaningfully named.	Assisted
BARR6-3-a	Parameterized macros shall not be used if a function can be written to accomplish the same behavior.	Assisted
BARR6-3-b_i	Surround the entire macro body with parentheses in parameterized macros.	Assisted
BARR6-3-b_ii	Surround each use of a parameter with parentheses in parameterized macros.	Assisted
BARR6-3-b_iii	Use each parameter no more than once, to avoid unintended side effects in parameterized macros.	Assisted
BARR6-3-b_iv	Never include a transfer of control in parameterized macros.	Assisted
BARR6-4-a	All functions that encapsulate threads of execution shall be given names ending with _thread.	Unassisted

Id	Description	Assisted
BARR6-5-a	Interrupt service routines (ISRs) are not ordinary functions. The compiler must be informed that the function is an ISR by way of a #pragma or compiler-specific keyword.	Unassisted
BARR6-5-b	All functions that implement ISRs shall be given names ending with _isr.	Unassisted
BARR6-5-c	To ensure that ISRs are not inadvertently called from other parts of the software, each ISR function shall be declared static and/or be located at the end of the associated driver module as permitted by the target platform.	Unassisted
BARR6-5-d	A stub or default ISR shall be installed in the vector table at the location of all unexpected or otherwise unhandled interrupt sources. Each such stub could attempt to disable future interrupts of the same type.	Unassisted
BARR7-1-a	No variable shall have a name that is a keyword of C, C++, or any other well-known extension of the C programming language.	Assisted
BARR7-1-b	No variable shall have a name that overlaps with a variable name from the C Standard Library.	Assisted
BARR7-1-c	No variable shall have a name that begins with an underscore.	Assisted
BARR7-1-d	No variable name shall be longer than 31 characters.	Assisted
BARR7-1-e	No variable name shall be shorter than 3 characters.	Assisted
BARR7-1-f	No variable name shall contain any uppercase letters.	Assisted
BARR7-1-g	No variable name shall contain any numeric value that is called out elsewhere.	Unassisted
BARR7-1-h	Underscores shall be used to separate words in variable names.	Unassisted
BARR7-1-i	Each variable's name shall be descriptive of its purpose.	Unassisted
BARR7-1-j	The names of any global variables shall begin with the letter 'g'.	Assisted
BARR7-1-k	The names of any pointer variables shall begin with the letter 'p'.	Assisted
BARR7-1-l	The names of any pointer-to-pointer variables shall begin with the letters 'pp'.	Assisted
BARR7-1-m	The names of all integer variables containing Boolean information shall begin with the letter 'b'.	Assisted
BARR7-1-n	The names of any variables representing non-pointer handles for objects shall begin with the letter 'h'.	Unassisted
BARR7-1-o	In the case of a variable name requiring multiple of the above prefixes, the order of their inclusion before the first underscore shall be [g][p|pp][b|h].	Unassisted

Id	Description	Assisted
BARR7-2-b	It is preferable to define local variables as you need them, rather than all at the top of a function.	Unassisted
BARR7-2-c	If project- or file-global variables are used, their definitions shall be grouped together and placed at the top of a source code file.	Unassisted
BARR7-2-d	Any pointer variable lacking an initial address shall be initialized to NULL.	Unassisted
BARR8-1-a	The comma operator shall not be used within variable declarations.	Unassisted
BARR8-2-a	It is a preferred practice that the shortest (measured in lines of code) of the if and else if clauses should be placed first.	Unassisted
BARR8-2-b	Nested if…else statements shall not be deeper than two levels.	Assisted
BARR8-2-c	Assignments shall not be made within an if or else if test.	Assisted
BARR8-2-d	Any if statement with an else if clause shall end with an else clause.	Assisted
BARR8-3-a	The break for each case shall be indented to align with the associated case, rather than with the contents of the case code block.	Unassisted
BARR8-3-b	All switch statements shall contain a default block.	Assisted
BARR8-3-c	Any case designed to fall through to the next shall be commented to clearly explain the absence of the corresponding break.	Assisted
BARR8-4-a	Magic numbers shall not be used as the initial value or in the endpoint test of a while, do…while, or for loop.	Unassisted
BARR8-4-b	With the exception of the initialization of a loop counter in the first clause of a for statement and the change to the same variable in the third, no assignment shall be made in any loop's controlling expression.	Assisted
BARR8-4-c	Infinite loops shall be implemented via controlling expression for (;;).	Assisted
BARR8-4-d	Each loop with an empty body shall feature a set of braces enclosing a comment to explain why nothing needs to be done until after the loop terminates.	Unassisted
BARR8-6-a	When evaluating the equality of a variable against a constant, the constant shall always be placed to the left of the equal-to operator (==).	Unassisted