Polyspace Bug Finder

Summary Report for Project: Heli_inner_loop

Report Author: bpotter

Polyspace Bug Finder: Summary Report for Project: Heli_inner_loop

by Report Author: bpotter

Published 14-May-2020 06:47:09

Analysis Author(s): bpotter
Polyspace Version(s): Polyspace Bug Finder 3.1 (R2019b)
Project Version(s): 1.0

Result Folder(s):
C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\verification\code_standard_checks\Heli_inner_loop\Heli_inner_loop

Table of Contents

Chapter 1. Polyspace Bug Finder Summary

Table 1.1. Project Summary

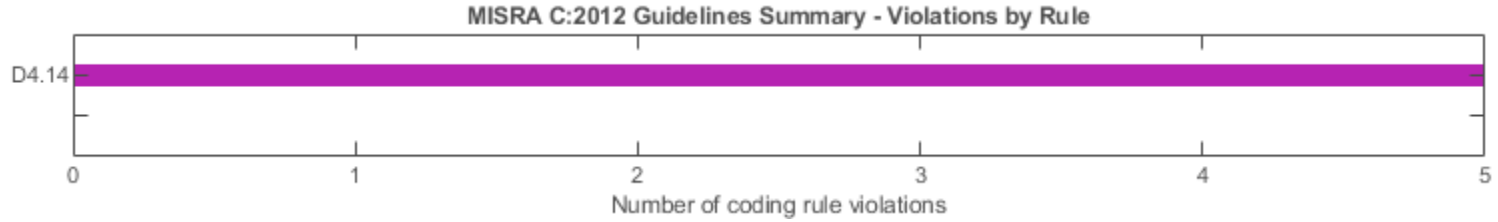
	Count	Reviewed	Unreviewed	Pass/Fail
MISRA C:2012 Guidelines	5	0	5	NA
Total	**5**	**0**	**5**	**NA**

Table 1.2. Summary By File

File	MISRA C:2012 Guidelines (Reviewed)
C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\specification\slprj\ert\Heli_inner_loop\Heli_inner_loop.c	5 (0)
C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\specification\slprj\ert\Heli_inner_loop\Heli_inner_loop.h	0 (0)

Chapter 2. MISRA C:2012 Guidelines

MISRA C:2012 Guidelines Summary - Violations by Rule



MISRA C:2012 Guidelines Summary for Enabled Guidelines

Guideline	Description	Mode	Total
D1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.	required	0
D2.1	All source files shall compile without any compilation errors.	required	0
D4.1	Run-time failures shall be minimized.	required	0
D4.3	Assembly language shall be encapsulated and isolated.	required	0
D4.7	If a function returns error information, then that error information shall be tested.	required	0
D4.10	Precautions shall be taken in order to prevent the contents of a header file being included more than once.	required	0
D4.11	The validity of values passed to library functions shall be checked.	required	0
D4.14	The validity of values received from external sources shall be checked.	required	5
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	0
1.3	There shall be no occurrence of undefined or critical unspecified behaviour.	required	0
2.1	A project shall not contain unreachable code.	required	0
2.2	There shall be no dead code.	required	0
2.7	There should be no unused parameters in functions.	advisory	0
Total			**5**

Guideline	Description	Mode	Total
3.1	The character sequences /* and // shall not be used within a comment.	required	0
3.2	Line-splicing shall not be used in // comments.	required	0
4.1	Octal and hexadecimal escape sequences shall be terminated.	required	0
5.1	External identifiers shall be distinct.	required	0
5.2	Identifiers declared in the same scope and name space shall be distinct.	required	0
5.4	Macro identifiers shall be distinct.	required	0
5.5	Identifiers shall be distinct from macro names.	required	0
5.6	A typedef name shall be a unique identifier.	required	0
5.7	A tag name shall be a unique identifier.	required	0
5.8	Identifiers that define objects or functions with external linkage shall be unique.	required	0
6.1	Bit-fields shall only be declared with an appropriate type.	required	0
6.2	Single-bit named bit fields shall not be of a signed type.	required	0
7.4	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char".	required	0
8.2	Function types shall be in prototype form with named parameters.	required	0
8.3	All declarations of an object or function shall use the same names and type qualifiers.	required	0
8.6	An identifier with external linkage shall have exactly one external definition.	required	0
8.8	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.	required	0
8.10	An inline function shall be declared with the static storage class.	required	0
8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.	required	0
9.1	The value of an object with automatic storage duration shall not be read before it has been set.	mandatory	0
9.4	An element of an object shall not be initialized more than once.	required	0
11.1	Conversions shall not be performed between a pointer to a function and any other type.	required	0
11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type.	required	0
11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type.	required	0
11.6	A cast shall not be performed between pointer to void and an arithmetic type.	required	0
11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type.	required	0
11.8	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.	required	0
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	0
12.5	The sizeof operator shall not have an operand which is a function parameter declared as "array of type".	mandatory	0
Total			**5**

Guideline	Description	Mode	Total
13.1	Initializer lists shall not contain persistent side effects.	required	0
13.2	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.	required	0
13.5	The right hand operand of a logical && or || operator shall not contain persistent side effects.	required	0
13.6	The operand of the sizeof operator shall not contain any expression which has potential side effects.	mandatory	0
14.1	A loop counter shall not have essentially floating type.	required	0
14.2	A for loop shall be well-formed.	required	0
14.3	Controlling expressions shall not be invariant.	required	0
15.6	The body of an iteration-statement or a selection-statement shall be a compound-statement.	required	0
17.1	The features of <stdarg.h> shall not be used.	required	0
17.2	Functions shall not call themselves, either directly or indirectly.	required	0
17.4	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	mandatory	0
17.6	The declaration of an array parameter shall not contain the static keyword between the [ ].	mandatory	0
18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.	required	0
18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	required	0
18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object.	required	0
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	0
18.7	Flexible array members shall not be declared.	required	0
18.8	Variable-length array types shall not be used.	required	0
19.1	An object shall not be assigned or copied to an overlapping object.	mandatory	0
20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.	required	0
20.3	The #include directive shall be followed by either a <filename> or "filename"sequence.	required	0
20.4	A macro shall not be defined with the same name as a keyword.	required	0
20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument.	required	0
20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.	required	0
20.9	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.	required	0
20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.	required	0
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	0
20.13	A line whose first token is # shall be a valid preprocessing directive.	required	0
Total			**5**

Guideline	Description	Mode	Total
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	0
21.1	#define and #undef shall not be used on a reserved identifier or reserved macro name.	required	0
21.2	A reserved identifier or macro name shall not be declared.	required	0
21.3	The memory allocation and deallocation functions of <stdlib.h> shall not be used.	required	0
21.4	The standard header file <setjmp.h> shall not be used.	required	0
21.5	The standard header file <signal.h> shall not be used.	required	0
21.6	The Standard Library input/output functions shall not be used.	required	0
21.7	The atof, atoi, atol, and atoll functions of <stdlib.h> shall not be used.	required	0
21.8	The library functions abort, exit and system of <stdlib.h> shall not be used.	required	0
21.9	The library functions bsearch and qsort of <stdlib.h> shall not be used.	required	0
21.10	The Standard Library time and date functions shall not be used.	required	0
21.11	The standard header file <tgmath.h> shall not be used.	required	0
21.13	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF.	mandatory	0
21.14	The Standard Library function memcmp shall not be used to compare null terminated strings.	required	0
21.15	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or un qualified versions of compatible types.	required	0
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	0
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	0
21.18	The size_t argument passed to any function in <string.h> shall have an appropriate value.	mandatory	0
21.19	The pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they ha ve pointer to const-qualified type.	mandatory	0
21.20	The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerr or shall not be used following a subsequent call to the same function.	mandatory	0
22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.	required	0
22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function.	mandatory	0
22.3	The same file shall not be open for read and write access at the same time on different streams.	required	0
22.4	There shall be no attempt to write to a stream which has been opened as read-only.	mandatory	0
22.5	A pointer to a FILE object shall not be dereferenced.	mandatory	0
22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	mandatory	0
Total			**5**

Guideline	Description	Mode	Total
22.7	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF.	required	0
22.8	The value of errno shall be set to zero prior to a call to an errno-setting-function.	required	0
22.9	The value of errno shall be tested against zero after calling an errno-setting-function.	required	0
22.10	The value of errno shall only be tested when the last function to be called was an errno-setting-function.	required	0
Total			**5**

Chapter 3. Appendix 1 - Configuration Settings

Polyspace Settings

Option	Value
-author	bpotter
-boolean-types	boolean_T
-bug-finder	true
-checkers	
-compiler	generic
-D	main=main_rtwec,__restrict__=,MODEL=Heli_inner_loop,NUMST=1,NCSTATES=0,HAVESTDIO,MODEL_HAS_DYNAMICALLY_LOADED_SFCNS=0,CLASSIC_INTERFACE=0,ALLOCATIONFCN=0,TID01EQ=0,PORTABLE_WORDSIZES,TERMFCN=0,ONESTEPFCN=1,MAT_FILE=0,MULTI_INSTANCE_CODE=0,INTEGER_CODE=0,MT=0
-data-range-specifications	C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\verification\code_standard_checks\Heli_inner_loop\Heli_inner_loop\Heli_inner_loop_drs.xml
-date	14/05/2020
-disable-checkers	all
-dos	true
-double-is-64bits	true
-functions-called-before-loop	Heli_inner_loop_initialize
-functions-called-in-loop	custom=Heli_inner_loop
-functions-to-stub	rtIsNaN,rtIsInf,rtIsNaNF,rtIsInfF
-I	C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\specification,C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\specification\slprj\ert\Heli_inner_loop,C:\Program Files\MATLAB\R2019b\extern\include,C:\Program Files\MATLAB\R2019b\simulink\include,C:\Program Files\MATLAB\R2019b\rtw\c\src,C:\Program Files\MATLAB\R2019b\rtw\c\src\ext_mode\common,C:\Program Files\MATLAB\R2019b\rtw\c\ert,C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\specification\slprj\ert_sharedutils
-import-comments	C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\verification\code_standard_checks\Heli_inner_loop\Heli_inner_loop\comments_bak
-int-is-32bits	true
-lang	C-CPP
-little-endian	true
-long-long-is-64bits	true

Option	Value
-main-generator	true
-mbd	true
-misra3	C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\tools\checks\MISRA_C_2012_ACG
-misra3-agc-mode	true
-pointer-is-32bits	true
-prog	Heli_inner_loop
-results-dir	C:\Users\bpotter\MATLAB\Projects\ARP_DO_Project\DO_04_Code\verification\code_standard_checks\Heli_inner_loop\Heli_inner_loop
-stub-embedded-coder-lookup-table-functions	true
-target	mcpu
-variables-written-before-loop	none
-variables-written-in-loop	none
-verif-version	1.0

Specified Constraints

Constraints - User Functions

Table 3.1. File: C:\ Users\ bpotter\ MATLAB\ Projects\ ARP_DO_Project\ DO_04_Code\ specification\ slprj\ ert\ Heli_inner_loop\ Heli_inner_loop.c

Name	Type	Main Generator Called	Init Mode	Init Range	Initialize Pointer	# Allocated Objects	Init Allocated	Global Assert	Assert Range	Comment
Heli_inner_loop.* arg1	const float64		INIT	-30..30				unsupported	unsupported	
Heli_inner_loop.* arg2	const float64		INIT	-30..30				unsupported	unsupported	
Heli_inner_loop.* arg3	const float64		INIT	-30..30				unsupported	unsupported	
Heli_inner_loop.* arg4	const float64		INIT	-180..180				unsupported	unsupported	
Heli_inner_loop.* arg5	float64		INIT	min..max				unsupported	unsupported	
Heli_inner_loop.arg1	float64 const *		INIT		Not NULL	MULTI	1			
Heli_inner_loop.arg2	float64 const *		INIT		Not NULL	MULTI	1			
Heli_inner_loop.arg3	float64 const *		INIT		Not NULL	MULTI	1			
Heli_inner_loop.arg4	float64 const *		INIT		Not NULL	MULTI	5			

Name	Type	Main Generator Called	Init Mode	Init Range	Initialize Pointer	# Allocated Objects	Init Allocated	Global Assert	Assert Range	Comment
Heli_inner_loop.arg5	float64 *		INIT		Not NULL	MULTI	3			

Coding Standard Configuration

Table 3.2. 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	Not enforceable	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 enforceable	no
D4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous.	advisory	-	no
D4.6	typedefs that indicate size and signedness should be used in place of the basic numerical types.	advisory	-	no
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	Not enforceable	no
D4.9	A function should be used in preference to a function-like macro where they are interchangeable.	advisory	-	no
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	Not enforceable	no
D4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.	advisory	-	no
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	-	no

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	-	no
2.4	A project should not contain unused tag declarations.	advisory	-	no
2.5	A project should not contain unused macro declarations.	advisory	-	no
2.6	A function should not contain unused label declarations.	advisory	-	no
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	-	no
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	-	no
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	-	no
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	-	no
7.2	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.	required	-	no
7.3	The lowercase character "l" shall not be used in a literal suffix.	required	-	no
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	-	no
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	-	no
8.5	An external object or function shall be declared once in one and only one file.	required	-	no
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	-	no
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	-	no
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	-	no
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	-	no
8.14	The restrict type qualifier shall not be used.	required	-	no
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	-	no
9.3	Arrays shall not be partially initialized.	required	-	no
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	-	no
10.1	Operands shall not be of an inappropriate essential type.	required	-	no
10.2	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.	required	-	no
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	-	no
10.4	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.	required	-	no
10.5	The value of an expression should not be cast to an inappropriate essential type.	advisory	-	no
10.6	The value of a composite expression shall not be assigned to an object with wider essential type.	required	-	no
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	-	no
10.8	The value of a composite expression shall not be cast to a different essential type category or a wider essential type.	required	-	no

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	-	no
11.5	A conversion should not be performed from pointer to void into pointer to object.	advisory	-	no
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	-	no
12.1	The precedence of operators within expressions should be made explicit.	advisory	-	no
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	-	no
12.4	Evaluation of constant expressions should not lead to unsigned integer wrap-around.	advisory	-	no
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	-	no
13.4	The result of an assignment operator should not be used.	advisory	-	no
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 shall have essentially Boolean type.	required	-	no
15.1	The goto statement should not be used.	advisory	-	no
15.2	The goto statement shall jump to a label declared later in the same function.	required	-	no
15.3	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.	required	-	no

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	-	no
15.5	A function should have a single point of exit at the end.	advisory	-	no
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	-	no
16.1	All switch statements shall be well-formed.	required	-	no
16.2	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	required	-	no
16.3	An unconditional break statement shall terminate every switch-clause.	required	-	no
16.4	Every switch statement shall have a default label.	required	-	no
16.5	A default label shall appear as either the first or the last switch label of a switch statement.	required	-	no
16.6	Every switch statement shall have at least two switch-clauses.	required	-	no
16.7	A switch-expression shall not have essentially Boolean type.	required	-	no
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	-	no
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	-	no
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	-	no
17.8	A function parameter should not be modified.	advisory	-	no
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	-	no
18.5	Declarations should contain no more than two levels of pointer nesting.	advisory	-	no
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	-	no
20.1	#include directives should only be preceded by preprocessor directives or comments.	advisory	-	no
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	-	no
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	-	no
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	-	no
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	-	no
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

Chapter 4. Appendix 2 - Definitions

Table 4.1. Abbreviations

Abbreviation	Definition
NA	Not Available