Polyspace Results in Polyspace Bug Finder

Review Analysis Results

Polyspace Bug Finder Results

Defects

Numerical Defects	
`Absorption of float operand`	One addition or subtraction operand is absorbed by the other operand
`Bitwise operation on negative value`	Undefined behavior for bitwise operations on negative values
`Float conversion overflow`	Overflow when converting between floating point data types
`Float division by zero`	Dividing floating point number by zero
`Float overflow`	Overflow from operation between floating points
`Integer conversion overflow`	Overflow when converting between integer types
`Integer division by zero`	Dividing integer number by zero
`Integer overflow`	Overflow from operation between integers
`Invalid use of standard library floating point routine`	Wrong arguments to standard library function
`Invalid use of standard library integer routine`	Wrong arguments to standard library function
`Shift of a negative value`	Shift operator on negative value
`Shift operation overflow`	Overflow from shifting operation
`Sign change integer conversion overflow`	Overflow when converting between signed and unsigned integers
`Unsigned integer conversion overflow`	Overflow when converting between unsigned integer types
`Unsigned integer overflow`	Overflow from operation between unsigned integers
`Use of plain char type for numerical value`	Plain char variable in arithmetic operation without explicit signedness

Static Memory Defects	
`Arithmetic operation with NULL pointer`	Arithmetic operation performed on NULL pointer
`Array access out of bounds`	Array index outside bounds during array access
`Buffer overflow from incorrect string format specifier`	String format specifier causes buffer argument of standard library functions to overflow
`Destination buffer overflow in string manipulation`	Function writes to buffer at offset greater than buffer size
`Destination buffer underflow in string manipulation`	Function writes to buffer at a negative offset from beginning of buffer
`Invalid use of standard library memory routine`	Standard library memory function called with invalid arguments
`Invalid use of standard library string routine`	Standard library string function called with invalid arguments
`Null pointer`	NULL pointer dereferenced
`Pointer access out of bounds`	Pointer dereferenced outside its bounds
`Pointer or reference to stack variable leaving scope`	Pointer to local variable leaves the variable scope
`Subtraction or comparison between pointers to different arrays`	Subtraction or comparison between pointers causes undefined behavior
`Unreliable cast of function pointer`	Function pointer cast to another function pointer with different argument or return type
`Unreliable cast of pointer`	Pointer implicitly cast to different data type
`Use of automatic variable as putenv-family function argument`	putenv-family function argument not accessible outside its scope
`Use of path manipulation function without maximum sized buffer checking`	Destination buffer of getwd or realpath is smaller than PATH_MAX bytes
`Wrong allocated object size for cast`	Allocated memory does not match destination pointer

Dynamic Memory Defects	
`Alignment changed after memory reallocation`	Memory reallocation changes the originally stricter alignment of an object
`Deallocation of previously deallocated pointer`	Memory freed more than once without allocation
`Invalid free of pointer`	Pointer deallocation without a corresponding dynamic allocation
`Invalid deletion of pointer`	Pointer deallocation using delete without corresponding allocation using new
`Memory leak`	Memory allocated dynamically not freed
`Mismatched alloc/dealloc functions on Windows`	Improper deallocation function causes memory corruption issues
`Unprotected dynamic memory allocation`	Pointer returned from dynamic allocation not checked for NULL value
`Use of previously freed pointer`	Memory accessed after deallocation

Programming Defects	
`Abnormal termination of exit handler`	Exit handler function interrupts the normal execution of a program
`Alternating input and output from a stream without flush or positioning call`	Undefined behavior for input or output stream operations
`Assertion`	Failed assertion statement
`Bad file access mode or status`	Access mode argument of function in fopen or open group is invalid
`Call through non-prototyped function pointer`	Function pointer declared without its type or number of parameters causes unexpected behavior
`Call to memset with unintended value`	memset or wmemset used with possibly incorrect arguments
`Character value absorbed into EOF`	Data type conversion makes a valid character value same as End-of-File (EOF)
`Copy of overlapping memory`	Source and destination arguments of a copy function have overlapping memory
`Declaration mismatch`	Mismatch between function or variable declarations
`Errno not reset`	errno not reset before calling a function that sets errno
`Exception caught by value`	catch statement accepts an object by value
`Exception handler hidden by previous handler`	catch statement is not reached because of an earlier catch statement for the same exception
`Floating point comparison with equality operators`	Imprecise comparison of floating-point variables

Programming Defects - continued	
Format string specifiers and arguments mismatch	String specifiers do not match corresponding arguments
Function called from signal handler not asynchronous-safe	Call to interrupted function causes undefined program behavior
Function called from signal handler not asynchronous-safe (strict)	Call to interrupted function causes undefined program behavior
Improper array initialization	Incorrect array initialization when using initializers
Incorrect pointer scaling	Implicit scaling in pointer arithmetic might be ignored
Invalid assumptions about memory organization	Address is computed by adding or subtracting from address of a variable
Invalid file position	fsetpos() is invoked with a file position argument not obtained from fgetpos()
Invalid use of = (assignment) operator	Assignment in conditional statement
Invalid use of == (equality) operator	Equality operation in assignment statement
Invalid use of standard library routine	Wrong arguments to standard library function
Invalid va_list argument	Variable argument list used after invalidation with va_end or not initialized with va_start or va_copy
Memory comparison of padding data	memcmp compares data stored in structure padding
Memory comparison of strings	memcmp compares data stored in strings after the null terminator
Missing byte reordering when transfering data	Different endianness of host and network
Missing null in string array	String does not terminate with null character
Misuse of a FILE object	Use of copy of FILE object
Misuse of structure with flexible array member	Memory allocation ignores flexible array member
Misuse of sign-extended character value	Data type conversion with sign extension causes unexpected behavior
Misuse of return value from nonreentrant standard function	Pointer to static buffer from previous call is used despite a subsequent call that modifies the buffer
Misuse of errno	errno incorrectly checked for error conditions
Modification of internal buffer returned from nonreentrant standard function	Function attempts to modify internal buffer returned from a nonreentrant standard function
Overlapping assignment	Memory overlap between left and right sides of an assignment
Possible misuse of sizeof	Use of sizeof operator can cause unintended results

Programming Defects - continued	
`Possibly unintended evaluation of expression because of operator precedence rules`	Operator precedence rules cause unexpected evaluation order in arithmetic expression
`Qualifier removed in conversion`	Variable qualifier is lost during conversion
`Return from computational exception signal handler`	Undefined behavior when signal handler returns normally from program error
`Signal call from within signal handler`	Nonpersistent signal handler calling signal() in Windows system causes race condition
`Shared data access within signal handler`	Access or modification of shared data causes inconsistent state
`Standard function call with incorrect arguments`	Argument to a standard function does not meet requirements for use in the function
`Typedef mismatch`	Mismatch between typedef statements
`Unsafe conversion between pointer and integer`	Misaligned or invalid results from conversions between pointer and integer types
`Unsafe conversion from string to numerical value`	String to number conversion without validation checks
`Use of indeterminate string`	Use of buffer from fgets-family function
`Use of memset with size argument zero`	Size argument of function in memset family is zero
`Variable length array with nonpositive size`	Size of variable-length array is zero or negative
`Writing to const qualified object`	Object declared with a const qualifier is modified
`Wrong type used in sizeof`	sizeof argument does not match pointed type

Data Flow Defects	
`Code deactivated by constant false condition`	Code segment deactivated by #if 0 directive or if(0) condition
`Dead code`	Code does not execute
`Missing return statement`	Function does not return value though return type is not void
`Non-initialized variable`	Variable not initialized before use
`Non-initialized pointer`	Pointer not initialized before dereference
`Partially accessed array`	Array partly read or written before end of scope
`Pointer to non-initialized value converted to const pointer`	Pointer to constant assigned address that does not contain a value
`Static uncalled function`	Function with static scope not called in file
`Unreachable code`	Code following control-flow statements
`Useless if`	Unnecessary if conditional
`Variable shadowing`	Variable hides another variable of same name with nested scope
`Write without a further read`	Variable never read after assignment

Security Defects	
`File access between time of check and use (TOCTOU)`	File or folder might change state due to access race
`File descriptor exposure to child process`	Copied file descriptor used in multiple processes
`File manipulation after chroot without chdir`	Path-related vulnerabilities for file manipulated after call to chroot
`Unsafe call to a system function`	Unsanitized command argument has exploitable vulnerabilities
`Use of non-secure temporary file`	Temporary generated file name not secure
`Vulnerable path manipulation`	Path argument with /../, /abs/path/, or other unsecure elements
`Bad order of dropping privileges`	Dropped higher elevated privileges before dropping lower elevated privileges
`Privilege drop not verified`	Verify privilege relinquishment was successful
`Umask used with chmod-style arguments`	Argument to umask allows external user too much control
`Vulnerable permission assignments`	Argument gives read/write/search permissions to external users
`Errno not checked`	errno is not checked for error conditions following function call
`Function pointer assigned with absolute address`	Constant expression is used as function address is vulnerable to code injection

Security Defects - continued	
Incorrect order of network connection operations	Socket is not correctly established due to bad order of connection steps or missing steps
Mismatch between data length and size	Data size argument is not computed from actual data length
Missing case for switch condition	switch variable not covered by cases and default case is missing
Misuse of readlink()	Third argument of readlink does not leave space for null terminator in buffer
Returned value of a sensitive function not checked	Sensitive functions called without checking for unexpected return values and errors
Sensitive data printed out	Function prints sensitive data
Sensitive heap memory not cleared before release	Sensitive data not cleared or released by memory routine
Uncleared sensitive data in stack	Variable in stack is not cleared and contains sensitive data
Unsafe standard encryption function	Function is not reentrant or uses a risky encryption algorithm
Unsafe standard function	Function unsafe for security-related purposes
Use of dangerous standard function	Dangerous functions cause possible buffer overflow in destination buffer
Use of obsolete standard function	Obsolete routines can cause security vulnerabilities and portability issues
Deterministic random output from constant seed	Seeding routine uses a constant seed making the output deterministic
Execution of a binary from a relative path can be controlled by an external actor	Command with relative path is vulnerable to malicious attack
Load of library from a relative path can be controlled by an external actor	Library loaded with relative path is vulnerable to malicious attacks
Predictable random output from predictable seed	Seeding routine uses a predictable seed making the output predictable
Vulnerable pseudo-random number generator	Using a cryptographically weak pseudo-random number generator
Constant block cipher initialization vector	Initialization vector is constant instead of randomized
Predictable block cipher initialization vector	Initialization vector is generated from a weak random number generator
Missing block cipher initialization vector	Non-NULL initialization vector is not associated with the cipher context for encryption or decryption
Constant cipher key	Encryption or decryption key is constant instead of randomized

Security Defects - continued	
`Predictable cipher key`	Encryption or decryption key is generated from a weak random number generator
`Missing cipher key`	Non-NULL key is not associated with the cipher context for encryption or decryption
`Inconsistent cipher operations`	Encryption and decryption steps occur in succession with the same cipher context without a reinitialization in between
`Missing cipher data to process`	Final encryption or decryption step is performed without previous update steps
`Missing cipher final step`	You do not perform a final step after update steps for encrypting or decrypting data
`Missing cipher algorithm`	An encryption or decryption algorithm is not associated with the cipher context
`Weak cipher algorithm`	Encryption algorithm associated with the cipher context is weak
`Weak cipher mode`	Encryption mode associated with the cipher context is weak

Tainted Data Defects	
`Array access with tainted index`	Array index from unsecure source possibly outside array bounds
`Command executed from externally controlled path`	Path argument from an unsecure source
`Execution of externally controlled command`	Command argument from an unsecure source vulnerable to operating system command injection
`Host change using externally controlled elements`	Changing host ID from an unsecure source
`Library loaded from externally controlled path`	Using a library argument from an externally controlled path
`Loop bounded with tainted value`	Loop controlled by a value from an unsecure source
`Memory allocation with tainted size`	Size argument to memory function is from an unsecure source
`Pointer dereference with tainted offset`	Offset is from an unsecure source and dereference may be out of bounds
`Tainted division operand`	Division / operands from an unsecure source
`Tainted modulo operand`	Remainder % operands are from an unsecure source
`Tainted NULL or non-null-terminated string`	Argument is from an unsecure source and may be NULL or not NULL-terminated
`Tainted sign change conversion`	Value from an unsecure source changes sign
`Tainted size of variable length array`	Size of the variable-length array (VLA) is from an unsecure source and may be zero, negative, or too large

Tainted Data Defects - continued	
`Tainted string format`	Input format argument is from an unsecure source
`Use of externally controlled environment variable`	Value of environment variable from an unsecure source
`Use of tainted pointer`	Pointer from an unsecure source may be NULL or point to unknown memory

Concurrency Defects	
`Data race`	Multiple tasks perform unprotected non-atomic operations on shared variable
`Data race including atomic operations`	Multiple tasks perform unprotected operations on shared variable
`Data race through standard library function call`	Multiple tasks make unprotected calls to thread-unsafe standard library function
`Deadlock`	Call sequence to lock functions cause two tasks to block each other
`Destruction of locked mutex`	Task tries to destroy a mutex in the locked state
`Double lock`	Lock function is called twice in a task without an intermediate call to unlock function
`Double unlock`	Unlock function is called twice in a task without an intermediate call to lock function
`Missing lock`	Unlock function without lock function
`Missing unlock`	*Lock function without unlock function*

Object Oriented Defects	
`*this not returned in copy assignment operator`	operator= method does not return a pointer to the current object
`Base class assignment operator not called`	Copy assignment operator does not call copy assignment operators of base subobjects
`Base class destructor not virtual`	Class cannot behave polymorphically for deletion of derived class objects
`Copy constructor not called in initialization list`	Copy constructor does not call copy constructors of some members or base classes
`Incompatible types prevent overriding`	Derived class method hides a virtual base class method instead of overriding it
`Member not initialized in constructor`	Constructor does not initialize some members of a class
`Missing explicit keyword`	Constructor missing the explicit specifier
`Missing virtual inheritance`	A base class is inherited virtually and nonvirtually in the same hierarchy
`Object slicing`	Derived class object passed by value to function with base class parameter

Object Oriented Defects - continued	
`Partial override of overloaded virtual functions`	Class overrides fraction of inherited virtual functions with a given name
`Return of non const handle to encapsulated data member`	Method returns pointer or reference to internal member of object
`Self assignment not tested in operator`	Copy assignment operator does not test for self-assignment

Resource Management Defects	
`Closing a previously closed resource`	Function closes a previously closed stream
`Opening previously opened resource`	Opening an already opened file
`Resource leak`	File stream not closed before FILE pointer scope ends or pointer is reassigned
`Use of previously closed resource`	Function operates on a previously closed stream
`Writing to read-only resource`	File initially opened as read only is modified

Good Practice Defects	
`Bitwise and arithmetic operation on the same data`	Statement with mixed bitwise and arithmetic operations
`Delete of void pointer`	delete operates on a void* pointer pointing to an object
`Hard-coded buffer size`	Size of memory buffer is a numerical value instead of symbolic constant
`Hard-coded loop boundary`	Loop boundary is a numerical value instead of symbolic constant
`Hard-coded object size used to manipulate memory`	Memory manipulation with hard-coded size instead of sizeof
`Large pass-by-value argument`	Large argument passed by value between functions
`Line with more than one statement`	Multiple statements on a line
`Missing break of switch case`	No comments at the end of switch case without a break statement
`Missing reset of a freed pointer`	Pointer free not followed by a reset statement to clear leftover data
`Unused parameter`	Function prototype has parameters not read or written in function body
`Use of setjmp/longjmp`	setjmp and longjmp cause deviation from normal control flow

Coding Rules

Custom Coding Rules

Group 1: Files

Group 2: Preprocessing

Group 3: Type definitions

Group 4: Structures

Group 5: Classes (C++)

Group 6: Enumerations

Group 7: Functions

Group 8: Constants

Group 9: Variables

Group 10: Name spaces (C++)

Group 11: Class templates (C++)

Group 12: Function templates (C++)

MISRA C:2012 Directives and Rules	
`MISRA C:2012 Dir 1.1`	Any implementation-defined behavior on which the output of the program depends shall be documented and understood
`MISRA C:2012 Dir 2.1`	All source files shall compile without any compilation errors
`MISRA C:2012 Dir 4.1`	Run-time failures shall be minimized
`MISRA C:2012 Dir 4.3`	Assembly language shall be encapsulated and isolated
`MISRA C:2012 Dir 4.5`	Identifiers in the same name space with overlapping visibility should be typographically unambiguous
`MISRA C:2012 Dir 4.6`	typedefs that indicate size and signedness should be used in place of the basic numerical types
`MISRA C:2012 Dir 4.7`	If a function returns error information, then that error information shall be tested
`MISRA C:2012 Dir 4.9`	A function should be used in preference to a function-like macro where they are interchangeable
`MISRA C:2012 Dir 4.10`	Precautions shall be taken in order to prevent the contents of a header file being included more than once
`MISRA C:2012 Dir 4.11`	The validity of values passed to library functions shall be checked
`MISRA C:2012 Dir 4.13`	Functions which are designed to provide operations on a resource should be called in an appropriate sequence
`MISRA C:2012 Dir 4.14`	The validity of values received from external sources shall be checked

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 1.1	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits
MISRA C:2012 Rule 1.2	Language extensions should not be used
MISRA C:2012 Rule 1.3	There shall be no occurrence of undefined or critical unspecified behaviour
MISRA C:2012 Rule 2.1	A project shall not contain unreachable code
MISRA C:2012 Rule 2.2	There shall be no dead code
MISRA C:2012 Rule 2.3	A project should not contain unused type declarations
MISRA C:2012 Rule 2.4	A project should not contain unused tag declarations
MISRA C:2012 Rule 2.5	A project should not contain unused macro declarations
MISRA C:2012 Rule 2.6	A function should not contain unused label declarations
MISRA C:2012 Rule 2.7	There should be no unused parameters in functions
MISRA C:2012 Rule 3.1	The character sequences /* and // shall not be used within a comment
MISRA C:2012 Rule 3.2	Line-splicing shall not be used in // comments
MISRA C:2012 Rule 4.1	Octal and hexadecimal escape sequences shall be terminated
MISRA C:2012 Rule 4.2	Trigraphs should not be used
MISRA C:2012 Rule 5.1	External identifiers shall be distinct
MISRA C:2012 Rule 5.2	Identifiers declared in the same scope and name space shall be distinct
MISRA C:2012 Rule 5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
MISRA C:2012 Rule 5.4	Macro identifiers shall be distinct
MISRA C:2012 Rule 5.5	Identifiers shall be distinct from macro names
MISRA C:2012 Rule 5.6	A typedef name shall be a unique identifier
MISRA C:2012 Rule 5.7	A tag name shall be a unique identifier
MISRA C:2012 Rule 5.8	Identifiers that define objects or functions with external linkage shall be unique
MISRA C:2012 Rule 5.9	Identifiers that define objects or functions with internal linkage should be unique
MISRA C:2012 Rule 6.1	Bit-fields shall only be declared with an appropriate type
MISRA C:2012 Rule 6.2	Single-bit named bit fields shall not be of a signed type
MISRA C:2012 Rule 7.1	Octal constants shall not be used

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 7.2	A “u” or “U” suffix shall be applied to all integer constants that are represented in an unsigned type
MISRA C:2012 Rule 7.3	The lowercase character “l” shall not be used in a literal suffix
MISRA C:2012 Rule 7.4	A string literal shall not be assigned to an object unless the object’s type is “pointer to const-qualified char”
MISRA C:2012 Rule 8.1	Types shall be explicitly specified
MISRA C:2012 Rule 8.2	Function types shall be in prototype form with named parameters
MISRA C:2012 Rule 8.3	All declarations of an object or function shall use the same names and type qualifiers
MISRA C:2012 Rule 8.4	A compatible declaration shall be visible when an object or function with external linkage is defined
MISRA C:2012 Rule 8.5	An external object or function shall be declared once in one and only one file
MISRA C:2012 Rule 8.6	An identifier with external linkage shall have exactly one external definition
MISRA C:2012 Rule 8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
MISRA C:2012 Rule 8.8	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage
MISRA C:2012 Rule 8.9	An object should be defined at block scope if its identifier only appears in a single function
MISRA C:2012 Rule 8.10	An inline function shall be declared with the static storage class
MISRA C:2012 Rule 8.11	When an array with external linkage is declared, its size should be explicitly specified
MISRA C:2012 Rule 8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique
MISRA C:2012 Rule 8.13	A pointer should point to a const-qualified type whenever possible
MISRA C:2012 Rule 8.14	The restrict type qualifier shall not be used
MISRA C:2012 Rule 9.1	The value of an object with automatic storage duration shall not be read before it has been set
MISRA C:2012 Rule 9.2	The initializer for an aggregate or union shall be enclosed in braces
MISRA C:2012 Rule 9.3	Arrays shall not be partially initialized
MISRA C:2012 Rule 9.4	An element of an object shall not be initialized more than once
MISRA C:2012 Rule 9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly
MISRA C:2012 Rule 10.1	Operands shall not be of an inappropriate essential type
MISRA C:2012 Rule 10.2	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 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
MISRA C:2012 Rule 10.4	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category
MISRA C:2012 Rule 10.5	The value of an expression should not be cast to an inappropriate essential type
MISRA C:2012 Rule 10.6	The value of a composite expression shall not be assigned to an object with wider essential type
MISRA C:2012 Rule 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
MISRA C:2012 Rule 10.8	The value of a composite expression shall not be cast to a different essential type category or a wider essential type
MISRA C:2012 Rule 11.1	Conversions shall not be performed between a pointer to a function and any other type
MISRA C:2012 Rule 11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type
MISRA C:2012 Rule 11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type
MISRA C:2012 Rule 11.4	A conversion should not be performed between a pointer to object and an integer type
MISRA C:2012 Rule 11.5	A conversion should not be performed from pointer to void into pointer to object
MISRA C:2012 Rule 11.6	A cast shall not be performed between pointer to void and an arithmetic type
MISRA C:2012 Rule 11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type
MISRA C:2012 Rule 11.8	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer
MISRA C:2012 Rule 11.9	The macro NULL shall be the only permitted form of integer null pointer constant
MISRA C:2012 Rule 12.1	The precedence of operators within expressions should be made explicit
MISRA C:2012 Rule 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
MISRA C:2012 Rule 12.3	The comma operator should not be used
MISRA C:2012 Rule 12.4	Evaluation of constant expressions should not lead to unsigned integer wrap-around
MISRA C:2012 Rule 12.5	The sizeof operator shall not have an operand which is a function parameter declared as “array of type”

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 13.1	Initializer lists shall not contain persistent side effects
MISRA C:2012 Rule 13.2	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders
MISRA C:2012 Rule 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
MISRA C:2012 Rule 13.4	The result of an assignment operator should not be used
MISRA C:2012 Rule 13.5	The right hand operand of a logical && or || operator shall not contain persistent side effects
MISRA C:2012 Rule 13.6	The operand of the sizeof operator shall not contain any expression which has potential side effects
MISRA C:2012 Rule 14.1	A loop counter shall not have essentially floating type
MISRA C:2012 Rule 14.2	A for loop shall be well-formed
MISRA C:2012 Rule 14.3	Controlling expressions shall not be invariant
MISRA C:2012 Rule 14.4	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type
MISRA C:2012 Rule 15.1	The goto statement should not be used
MISRA C:2012 Rule 15.2	The goto statement shall jump to a label declared later in the same function
MISRA C:2012 Rule 15.3	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement
MISRA C:2012 Rule 15.4	There should be no more than one break or goto statement used to terminate any iteration statement
MISRA C:2012 Rule 15.5	A function should have a single point of exit at the end
MISRA C:2012 Rule 15.6	The body of an iteration-statement or a selection-statement shall be a compound statement
MISRA C:2012 Rule 15.7	All if … else if constructs shall be terminated with an else statement
MISRA C:2012 Rule 16.1	All switch statements shall be well-formed
MISRA C:2012 Rule 16.2	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement
MISRA C:2012 Rule 16.3	An unconditional break statement shall terminate every switch-clause
MISRA C:2012 Rule 16.4	Every switch statement shall have a default label
MISRA C:2012 Rule 16.5	A default label shall appear as either the first or the last switch label of a switch statement
MISRA C:2012 Rule 16.6	Every switch statement shall have at least two switch-clauses

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 16.6	Every switch statement shall have at least two switch-clauses
MISRA C:2012 Rule 16.7	A switch-expression shall not have essentially Boolean type
MISRA C:2012 Rule 17.1	The features of <starg.h> shall not be used
MISRA C:2012 Rule 17.2	Functions shall not call themselves, either directly or indirectly
MISRA C:2012 Rule 17.3	A function shall not be declared implicitly
MISRA C:2012 Rule 17.4	All exit paths from a function with non-void return type shall have an explicit return statement with an expression
MISRA C:2012 Rule 17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements
MISRA C:2012 Rule 17.6	The declaration of an array parameter shall not contain the static keyword between the [ ]
MISRA C:2012 Rule 17.7	The value returned by a function having non-void return type shall be used
MISRA C:2012 Rule 17.8	A function parameter should not be modified
MISRA C:2012 Rule 18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand
MISRA C:2012 Rule 18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array
MISRA C:2012 Rule 18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object
MISRA C:2012 Rule 18.4	The +, -, += and -= operators should not be applied to an expression of pointer type
MISRA C:2012 Rule 18.5	Declarations should contain no more than two levels of pointer nesting
MISRA C:2012 Rule 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
MISRA C:2012 Rule 18.7	Flexible array members shall not be declared
MISRA C:2012 Rule 18.8	Variable-length array types shall not be used
MISRA C:2012 Rule 19.1	An object shall not be assigned or copied to an overlapping object
MISRA C:2012 Rule 19.2	The union keyword should not be used
MISRA C:2012 Rule 20.1	#include directives should only be preceded by preprocessor directives or comments
MISRA C:2012 Rule 20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 20.3	The #include directive shall be followed by either a <filename> or \"filename\" sequence
MISRA C:2012 Rule 20.4	A macro shall not be defined with the same name as a keyword
MISRA C:2012 Rule 20.5	#undef should not be used
MISRA C:2012 Rule 20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument
MISRA C:2012 Rule 20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses
MISRA C:2012 Rule 20.8	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1
MISRA C:2012 Rule 20.9	All identifiers used in the controlling expression of #if or #elif pre-processing directives shall be #define'd before evaluation
MISRA C:2012 Rule 20.10	The # and ## preprocessor operators should not be used
MISRA C:2012 Rule 20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator
MISRA C:2012 Rule 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
MISRA C:2012 Rule 20.13	A line whose first token is # shall be a valid preprocessing directive
MISRA C:2012 Rule 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
MISRA C:2012 Rule 21.1	#define and #undef shall not be used on a reserved identifier or reserved macro name
MISRA C:2012 Rule 21.2	A reserved identifier or macro name shall not be declared
MISRA C:2012 Rule 21.3	The memory allocation and deallocation functions of <stdlib.h> shall not be used
MISRA C:2012 Rule 21.4	The standard header file <setjmp.h> shall not be used
MISRA C:2012 Rule 21.5	The standard header file <signal.h> shall not be used
MISRA C:2012 Rule 21.6	The Standard Library input/output functions shall not be used
MISRA C:2012 Rule 21.7	The atof, atoi, atol, and atoll functions of <stdlib.h> shall not be used
MISRA C:2012 Rule 21.8	The library functions abort, exit, getenv and system of <stdlib.h> shall not be used
MISRA C:2012 Rule 21.9	The library functions bsearch and qsort of <stdlib.h> shall not be used
MISRA C:2012 Rule 21.10	The Standard Library time and date functions shall not be used
MISRA C:2012 Rule 21.11	The standard header file <tgmath.h> shall not be used

MISRA C:2012 Directives and Rules - continued	
MISRA C:2012 Rule 21.12	The exception handling features of <fenv.h> should not be used
MISRA C:2012 Rule 21.13	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF
MISRA C:2012 Rule 21.14	The Standard Library function memcmp shall not be used to compare null terminated strings
MISRA C:2012 Rule 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
MISRA C:2012 Rule 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
MISRA C:2012 Rule 21.17	Use of the string handling function from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters
MISRA C:2012 Rule 21.18	The size_t argument passed to any function in <string.h> shall have an appropriate value
MISRA C:2012 Rule 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
MISRA C:2012 Rule 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
MISRA C:2012 Rule 22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released
MISRA C:2012 Rule 22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function
MISRA C:2012 Rule 22.3	The same file shall not be open for read and write access at the same time on different streams
MISRA C:2012 Rule 22.4	There shall be no attempt to write to a stream which has been opened as read-only
MISRA C:2012 Rule 22.5	A pointer to a FILE object shall not be dereferenced
MISRA C:2012 Rule 22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed
MISRA C:2012 Rule 22.7	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF
MISRA C:2012 Rule 22.8	The value of errno shall be set to zero prior to a call to an errno-setting-function
MISRA C:2012 Rule 22.9	The value of errno shall be tested against zero after calling an errno-setting function
MISRA C:2012 Rule 22.10	The value of errno shall only be tested when the last function to be called was an errno-setting function

Code Metrics	
`Number of Direct Recursions`	Number of instances of a function calling itself directly
`Number of Header Files`	Number of included header files
`Number of Files`	Number of source files
`Number of Recursions`	Number of call graph cycles over one or more functions
`Comment Density`	Ratio of number of comments to number of statements
`Estimated Function Coupling`	Measure of complexity between levels of call tree
`Number of Lines`	Total number of lines in a file
`Number of Lines Without Comment`	Number of lines of code excluding comments
`Cyclomatic Complexity`	Number of linearly independent paths in function body
`Higher Estimate of Local Variable Size`	Total size of all local variables in function
`Language Scope`	Language scope
`Lower Estimate of Local Variable Size`	Total size of local variables in function taking nested scopes into account
`Number of Call Levels`	Maximum depth of nesting of control flow structures
`Number of Call Occurrences`	Number of calls in function body
`Number of Called Functions`	Number of callees of a function
`Number of Calling Functions`	Number of distinct callers of a function
`Number of Executable Lines`	Number of executable lines in function body
`Number of Function Parameters`	Number of function arguments
`Number of Goto Statements`	Number of goto statements
`Number of Instructions`	Number of instructions per function
`Number of Lines Within Body`	Number of lines in function body
`Number of Local Non-Static Variables`	Total number of local variables in function
`Number of Local Static Variables`	Total number of local static variables in function
`Number of Paths`	Estimated static path count
`Number of Return Statements`	Number of return statements in a function