C++ static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C++ code

1. "memset" should not be used to delete sensitive data
   Vulnerability
2. POSIX functions should not be called with arguments that trigger buffer overflows
   Vulnerability
3. XML parsers should not be vulnerable to XXE attacks
   Vulnerability
4. Function-like macros should not be invoked without all of their arguments
   Bug
5. The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist
   Bug
6. Assigning to an optional should directly target the optional
   Bug
7. Result of the standard remove algorithms should not be ignored
   Bug
8. "std::scoped_lock" should be created with constructor arguments
   Bug
9. Objects should not be sliced
   Bug
10. Immediately dangling references should not be created
    Bug
11. "pthread_mutex_t" should be unlocked in the reverse order they were locked
    Bug
12. "pthread_mutex_t" should be properly initialized and destroyed
    Bug
13. "pthread_mutex_t" should not be consecutively locked or unlocked twice
    Bug
14. "std::move" and "std::forward" should not be confused
    Bug
15. A call to "wait()" on a "std::condition_variable" should have a condition
    Bug

16.

A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast
Bug

17.

Functions with "noreturn" attribute should not return
Bug

18.

RAII objects should not be temporary
Bug

19.

"memcmp" should only be called with pointers to trivially copyable types with no padding
Bug

20.

"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types
Bug

21.

"std::auto_ptr" should not be used
Bug

22.

Destructors should be "noexcept"
Bug

23.

Stack allocated memory and non-owned memory should not be freed
Bug

24.

Closed resources should not be accessed
Bug

25.

Dynamically allocated memory should be released
Bug

26.

Freed memory should not be used
Bug

27.

Memory locations should not be released more than once
Bug

28.

Memory access should be explicitly bounded to prevent buffer overflows
Bug

29.

Printf-style format strings should not lead to unexpected behavior at runtime
Bug

30.

Recursion should not be infinite
Bug

31.

Resources should be closed
Bug

32.

Appropriate memory de-allocation should be used
Bug

33.

Hard-coded credentials are security-sensitive
Security Hotspot

34.

"goto" should jump to labels declared later in the same function
Code Smell

35.

The name "main" should not be used for any function other than the global "main" function
Code Smell

36.

Only standard forms of the "defined" directive should be used
Code Smell

37.

Switch labels should not be nested inside non-switch blocks
Code Smell

38.

The right-hand operands of && and || should not contain side effects
Code Smell

39.

Digraphs should not be used
Code Smell

40.

Trigraphs should not be used
Code Smell

41.

Use "std::variant" instead of unions with non-trivial types.
Code Smell

42.

A single statement should not have more than one resource allocation
Code Smell

43.

Facilities in <random> should be used instead of "srand", "rand" and "random_shuffle"
Code Smell

44.

Move and swap operations should be "noexcept"
Code Smell

45.

"case" ranges should cover multiple values
Code Smell

46.

Array indices should be placed between brackets
Code Smell

47.

Comparison operators should not be virtual
Code Smell

48.

Assignment operators should not be "virtual"
Code Smell

49.

Redundant pointer operator sequences should be removed
Code Smell

50.

Child class fields should not shadow parent class fields
Code Smell

51.

Non-reentrant POSIX functions should be replaced with their reentrant versions
Code Smell

52.

"goto" statements should not be used to jump into blocks
Code Smell

53.

Keywords introduced in later specifications should not be used as identifiers
Code Smell

54.

Context-sensitive keywords should not be used as identifiers
Code Smell

55.

Switch cases should end with an unconditional "break" statement
Code Smell

56.

"switch" statements should not contain non-case labels
Code Smell

57.

Control should not be transferred into a complex logic block using a "goto" or a "switch" statement
Code Smell

58.

Accessing files should not introduce TOCTOU vulnerabilities
Vulnerability

59.

Cipher algorithms should be robust
Vulnerability

60.

Encryption algorithms should be used with secure mode and padding scheme
Vulnerability

61.

Server hostnames should be verified during SSL/TLS connections
Vulnerability

62.

Server certificates should be verified during SSL/TLS connections
Vulnerability

63.

Cryptographic keys should be robust
Vulnerability

64.

Weak SSL/TLS protocols should not be used
Vulnerability

65.

Insecure functions should not be used

Vulnerability

66.

"scanf()" and "fscanf()" format strings should specify a field width for the "%s" string placeholder
Vulnerability

67.

Function exit paths should have appropriate return values
Bug

68.

Coroutine should have co_return on each execution path or provide return_void
Bug

69.

"volatile" should not be used to qualify objects for which the meaning is not defined
Bug

70.

"volatile" types should not be used in compound operations
Bug

71.

Values returned from string find-related methods should not be treated as boolean
Bug

72.

Relational and subtraction operators should not be used with pointers to different arrays
Bug

73.

Arguments evaluation order should not be relied on
Bug

74.

"reinterpret_cast" should be used carefully
Bug

75.

Parameter values should be appropriate
Bug

76.

Zero should not be a possible denominator
Bug

77.

Line-splicing should not be used in "//" comments
Bug

78.

Member variables should be initialized
Bug

79.

Pointers should not be cast to integral types
Bug

80.

"operator delete" should be written along with "operator new"
Bug

81.

Destructors should not throw exceptions
Bug

82.

Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases
Bug

83.

Empty throws ("throw;") should only be used in the compound statements of catch handlers
Bug

84.

An exception object should not have pointer type
Bug

85.

"sprintf" should not be used
Security Hotspot

86.

Changing working directories without verifying the success is security-sensitive
Security Hotspot

87.

Using "tmpnam", "tmpnam_s" or "tmpnam_r" is security-sensitive
Security Hotspot

88.

Changing directories improperly when using "chroot" is security-sensitive
Security Hotspot

89.

Using publicly writable directories is security-sensitive
Security Hotspot

90.

Using clear-text protocols is security-sensitive
Security Hotspot

91.

Expanding archive files without controlling resource consumption is security-sensitive
Security Hotspot

92.

Using weak hashing algorithms is security-sensitive
Security Hotspot

93.

Using pseudorandom number generators (PRNGs) is security-sensitive
Security Hotspot

94.

"#undef" should be used with caution
Code Smell

95.

Function names should be used either as a call with a parameter list or with the "&" operator
Code Smell

96.

Functions should not be defined with a variable number of arguments
Code Smell

97.

The comma operator, "&&", and "||" should not be overloaded
Code Smell

98.

A cast shall not remove any const or volatile qualification from the type of a pointer or reference
Code Smell

99.
The return value of "std::move" should be used in a function
Code Smell

100.
Cognitive Complexity of coroutines should not be too high
Code Smell

101.
Use discriminated unions or "std::variant"
Code Smell

102.
Multiple mutexes should not be acquired with individual locks
Code Smell

103.
Pointers or references obtained from aliased smart pointers should not be used as function parameters
Code Smell

104.
"try_lock", "lock" and "unlock" should not be directly used for mutexes
Code Smell

105.
Appropriate arguments should be passed to UNIX/POSIX functions
Code Smell

106.
Appropriate arguments should be passed to stream functions
Code Smell

107.
"Forwarding references" parameters should be used only to forward parameters
Code Smell

108.
Non-const global variables should not be used
Code Smell

109.
Functions that throw exceptions should not be used as hash functions
Code Smell

110.
Blocking functions should not be called inside critical sections
Code Smell

111.
Return value of "setuid" family of functions should always be checked
Code Smell

112.
Size of variable length arrays should be positive
Code Smell

113.
Argument of "printf" should be a format string
Code Smell

114.
"mktemp" family of functions templates should have at least six trailing "X"s
Code Smell

115.

Logical operators should not be confused with bitwise operators
Code Smell

116.

Header guards should be followed by according "#define" macro
Code Smell

117.

Template parameters should be preferred to "std::function" when configuring behavior at compile time
Code Smell

118.

The addresses of standard library functions should not be taken
Code Smell

119.

Macros should not be used to define constants
Code Smell

120.

Memory should not be managed manually
Code Smell

121.

Lambdas that capture "this" should capture everything explicitly
Code Smell

122.

"void *" should not be used in typedefs, member variables, function parameters or return type
Code Smell

123.

The "Rule-of-Zero" should be followed
Code Smell

124.

"nullptr" should be used to denote the null pointer
Code Smell

125.

"default" clauses should be first or last
Code Smell

126.

A conditionally executed single line should be denoted by indentation
Code Smell

127.

Conditionals should start on new lines
Code Smell

128.

Cognitive Complexity of functions should not be too high
Code Smell

129.

Exceptions should not be thrown in "noexcept" functions
Code Smell

130.

Member variables should not be "protected"
Code Smell

131.

When the "Rule-of-Zero" is not applicable, the "Rule-of-Five" should be followed
Code Smell

132.

Default capture should not be used
Code Smell

133.

Standard groupings should be used with digit separators
Code Smell

134.

Special member function should not be defined unless a non standard behavior is required
Code Smell

135.

Standard namespaces should not be modified
Code Smell

136.

Destructors should not be called explicitly
Code Smell

137.

"static" base class members should not be accessed via derived types
Code Smell

138.

Control characters should not be used in literals
Code Smell

139.

Exception specifications should not be used
Code Smell

140.

The sign of an unsigned variable should not be tested
Code Smell

141.

Pre-defined macros should not be defined, redefined or undefined
Code Smell

142.

"explicit" should be used on single-parameter constructors and conversion operators
Code Smell

143.

Constructors and destructors should only use defined methods and fields
Code Smell

144.

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Code Smell

145.

Inherited functions should not be hidden
Code Smell

146.

C-style memory allocation routines should not be used
Code Smell

147.

Methods should not be empty
Code Smell

148.

Pure "virtual" functions should not override non-pure "virtual" functions
Code Smell

149.

"using namespace" directives should not be used in header files
Code Smell

150.

Account validity should be verified when authenticating users with PAM
Vulnerability

151.

Lines starting with "#" should contain valid preprocessing directives
Bug

152.

"#include" directives should be followed by either <filename> or "filename" sequences
Bug

153.

Non-standard characters should not occur in header file names in "#include" directives
Bug

154.

Non-empty statements should change control flow or have at least one side-effect
Bug

155.

Unary minus should not be applied to an unsigned expression
Bug

156.

Objects with integer type should not be converted to objects with pointer type
Bug

157.

Variables should be initialized before use
Bug

158.

String literals with different prefixes should not be concatenated
Bug

159.

Only escape sequences defined in the ISO C standard should be used
Bug

160.

"std::bit_cast" should be used instead of union type-punning
Bug

161.

"std::cmp_*" functions should be used to compare unsigned values with negative values
Bug

162.

Call to "std::is_constant_evaluated" should not be gratuitous
Bug

163.

Heterogeneous sorted containers should only be used with types that support heterogeneous comparison
Bug

164.

"#pragma pack" should be used correctly

Bug

165.

Enums should be consistent with the bit fields they initialize
Bug

166.

Class members should not be initialized with dangling references
Bug

167.

Array values should not be replaced unconditionally
Bug

168.

Integral operations should not overflow
Bug

169.

"case" ranges should not be empty
Bug

170.

All branches in a conditional structure should not have exactly the same implementation
Bug

171.

"extern" shouldn't be used on member definitions
Bug

172.

Declaration specifiers should not be redundant
Bug

173.

Function declarations that look like variable declarations should not be used
Bug

174.

"sizeof" should not be called on pointers
Bug

175.

"const" references to numbers should not be made
Bug

176.

Unary prefix operators should not be repeated
Bug

177.

"=+" should not be used instead of "+="
Bug

178.

Values of different "enum" types should not be compared
Bug

179.

Conditionally executed code should be reachable
Bug

180.

Null pointers should not be dereferenced
Bug

181.

Single-bit named bit fields should not be of a signed type

Bug

182.

Values should not be uselessly incremented
Bug

183.

"sizeof(sizeof(...))" should not be used
Bug

184.

Related "if/else if" statements should not have the same condition
Bug

185.

Identical expressions should not be used on both sides of a binary operator
Bug

186.

All code should be reachable
Bug

187.

Loops with at most one iteration should be refactored
Bug

188.

The original exception object should be rethrown
Bug

189.

Variables should not be self-assigned
Bug

190.

Condition-specific "catch" handlers should not be used after the ellipsis (catch-all) handler
Bug

191.

Handlers in a single try-catch or function-try-block for a derived class and some or all of its bases should be ordered most-derived-first
Bug

192.

Exception classes should be caught by reference
Bug

193.

Setting capabilities is security-sensitive
Security Hotspot

194.

Using "strncpy" or "wcsncpy" is security-sensitive
Security Hotspot

195.

Using "strncat" or "wcsncat" is security-sensitive
Security Hotspot

196.

Using "strcat" or "wcscat" is security-sensitive
Security Hotspot

197.

Using "strlen" or "wcslen" is security-sensitive
Security Hotspot

198.

Using "strcpy" or "wcscpy" is security-sensitive
Security Hotspot

199.

Setting loose POSIX file permissions is security-sensitive
Security Hotspot

200.

#include directives in a file should only be preceded by other preprocessor directives or comments
Code Smell

201.

Loops should not have more than one "break" or "goto" statement
Code Smell

202.

Unused type declarations should be removed
Code Smell

203.

Comma operator should not be used
Code Smell

204.

The unary "&" operator should not be overloaded
Code Smell

205.

"bool" expressions should not be used as operands to built-in operators other than =, &&, ||, !, ==, !=, unary &, and the conditional operator
Code Smell

206.

"enum" members other than the first one should not be explicitly initialized unless all members are explicitly initialized
Code Smell

207.

If a function has internal linkage then all re-declarations shall include the static storage class specifer
Code Smell

208.

Functions should not be declared at block scope
Code Smell

209.

Bit fields should be declared with appropriate types
Code Smell

210.

Coroutines should not take const references as parameters
Code Smell

211.

Thread local variables should not be used in coroutines
Code Smell

212.

Use symmetric transfer to switch execution between coroutines
Code Smell

213.

rvalue reference members should not be copied accidentally
Code Smell

214.

"std::string_view" and "std::span" parameters should be directly constructed from sequences
Code Smell

215.

Comparision operators ("<=>", "==") should be defaulted unless non-default behavior is required
Code Smell

216.

"std::chrono" components should be used to operate on time
Code Smell

217.

"std::enable_if" should not be used
Code Smell

218.

"std::source_location" should be used instead of "__FILE__", "__LINE__", and "__func__" macros
Code Smell

219.

Function template parameters should be named if reused
Code Smell

220.

Redundant comparison operators should not be defined
Code Smell

221.

"std::bit_cast" should be used to reinterpret binary representation instead of "std::memcpy"
Code Smell

222.

"[[likely]]" and "[[unlikely]]" should be used instead of compiler built-ins
Code Smell

223.

"starts_with" and "ends_with" should be used for prefix and postfix checks
Code Smell

224.

Designated initializers should be used in their C++ compliant form
Code Smell

225.

"std::jthread" should be used instead of "std::thread"
Code Smell

226.

Elements in a container should be erased with "std::erase" or "std::erase_if"
Code Smell

227.

Mathematical constants should not be hardcoded
Code Smell

228.

Transparent comparator should be used with associative "std::string" containers
Code Smell

229.

"emplace" should be prefered over "insert" with "std::set" and "std::unordered_set"
Code Smell

230.
Unnecessary expensive copy should be avoided when using auto as a placeholder type
Code Smell

231.
The right template argument should be specified for std::forward
Code Smell

232.
"try_emplace" should be used with "std::map" and "std::unordered_map"
Code Smell

233.
Exception specifications should be treated as part of the type
Code Smell

234.
"auto" should be used for non-type template parameter
Code Smell

235.
"std::optional" member function "value_or" should be used
Code Smell

236.
"std::byte" should be used when you need byte-oriented memory access
Code Smell

237.
Inline variables should be used to declare global variables in header files
Code Smell

238.
"[*this]" should be used to capture the current object by copy
Code Smell

239.
"std::uncaught_exception" should not be used
Code Smell

240.
Objects should not be created solely to be passed as arguments to functions that perform delegated object creation
Code Smell

241.
"std::filesystem::path" should be used to represent a file path
Code Smell

242.
Fold expressions should be used instead of recursive template instantiations
Code Smell

243.
"as_const" should be used to make a value constant
Code Smell

244.
Structured binding should be used
Code Smell

245.
Emplacement should be prefered when insertion creates a temporary with sequence containers
Code Smell

246.

"std::visit" should be used to switch on the type of the current value in a "std::variant"
Code Smell

247.

"bind" should not be used
Code Smell

248.

Use "make_unique" and "make_shared" to construct "unique_ptr" and "shared_ptr"
Code Smell

249.

C-style array should not be used
Code Smell

250.

"auto" should be used to avoid repetition of types
Code Smell

251.

Integer literals should not be cast to bool
Code Smell

252.

Member functions that don't mutate their objects should be declared "const"
Code Smell

253.

Functions having rvalue reference arguments should "std::move" those arguments
Code Smell

254.

Capture by reference in lambdas used locally
Code Smell

255.

Size of bit fields should not exceed the size of their types
Code Smell

256.

"std::move" should only be used where moving can happen
Code Smell

257.

Classes should not contain both public and private data members
Code Smell

258.

GNU attributes should be used correctly
Code Smell

259.

Unevaluated operands should not have side effects
Code Smell

260.

Size argument of memory functions should be consistent
Code Smell

261.

Return value of "nodiscard" functions should not be ignored
Code Smell

262.

Implicit casts should not lower precision
Code Smell

263.

"std::move" should only be added when necessary
Code Smell

264.
Appropriate size arguments should be passed to "strncat" and "strlcpy"
Code Smell

265.
Moved-from objects should not be relied upon
Code Smell

266.
Keywords shall not be used as macros identifiers
Code Smell

267.
Incomplete types should not be deleted
Code Smell

268.
Dereferenced null pointers should not be bound to references
Code Smell

269.
"else" statements should be clearly matched with an "if"
Code Smell

270.
Function pointers should not be used as function parameters
Code Smell

271.
Function parameters should not be of type "std::unique_ptr<T> const &"
Code Smell

272.
Include directives should not rely on non-portable search strategy
Code Smell

273.
Methods should not have identical implementations
Code Smell

274.
"#include" paths should be portable
Code Smell

275.
"#import" should not be used
Code Smell

276.
Atomic types should be used instead of "volatile" types
Code Smell

277.
String literals should not be immediately followed by macros
Code Smell

278.
"reinterpret_cast" should not be used
Code Smell

279.
"switch" statements should cover all cases
Code Smell

280.

Methods returns should not be invariant
Code Smell

281.

Printf-style format strings should be used correctly
Code Smell

282.

Conditional operators should not be nested
Code Smell

283.

Member data should be initialized in-class or in a constructor initialization list
Code Smell

284.

"this" should not be compared with null
Code Smell

285.

The "delete" operator should only be used for pointers
Code Smell

286.

Multiline blocks should be enclosed in curly braces
Code Smell

287.

Increment should not be used to set boolean variables to 'true'
Code Smell

288.

Boolean expressions should not be gratuitous
Code Smell

289.

Standard C++ headers should be used
Code Smell

290.

Parameters should be passed in the correct order
Code Smell

291.

"static" members should be accessed statically
Code Smell

292.

Obsolete POSIX functions should not be used
Code Smell

293.

Two branches in a conditional structure should not have exactly the same implementation
Code Smell

294.

Unused assignments should be removed
Code Smell

295.

Structures should not have too many fields
Code Smell

296.

"switch" statements should not have too many "case" clauses
Code Smell

297.
Classes should not have too many methods
Code Smell

298.
Sections of code should not be commented out
Code Smell

299.
Pass by reference to const should be used for large input parameters
Code Smell

300.
Assignment operators should return non-"const" references
Code Smell

301.
Polymorphic base class destructor should be either public virtual or protected non-virtual
Code Smell

302.
Lambdas should not have too many lines
Code Smell

303.
Generic exceptions should not be caught
Code Smell

304.
Unused function parameters should be removed
Code Smell

305.
Unused functions and methods should be removed
Code Smell

306.
Try-catch blocks should not be nested
Code Smell

307.
Track uses of "FIXME" tags
Code Smell

308.
Deprecated attributes should include explanations
Code Smell

309.
Assignments should not be made from within sub-expressions
Code Smell

310.
Generic exceptions should never be thrown
Code Smell

311.
Variables should not be shadowed
Code Smell

312.
Redundant pairs of parentheses should be removed
Code Smell

313.
Inheritance tree of classes should not be too deep
Code Smell

314.

Nested blocks of code should not be left empty
Code Smell

315.

Functions should not have too many parameters
Code Smell

316.

Unused "private" fields should be removed
Code Smell

317.

Collapsible "if" statements should be merged
Code Smell

318.

Unused labels should be removed
Code Smell

319.

Virtual functions should be declared with the "virtual" keyword
Code Smell

320.

Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments
Code Smell

321.

Header files should not contain unnamed namespaces
Code Smell

322.

The "sizeof" and "alignof" operator should not be used with operands of a "void" type
Bug

323.

"nonnull" pointers should not be set to null
Bug

324.

"for" loop counters should not have essentially floating type
Bug

325.

Line continuation characters '\' should not be followed by trailing whitespace
Bug

326.

Using hardcoded IP addresses is security-sensitive
Security Hotspot

327.

Pointer and reference parameters should be "const" if the corresponding object is not modified
Code Smell

328.

The three expressions of a "for" statement should only be concerned with loop control
Code Smell

329.

Literal suffix "L" for long integers shall be upper case
Code Smell

330.

Use type-erased "coroutine_handle" when applicable
Code Smell

331.

Use conditional suspension to resume current coroutine
Code Smell

332.

"auto" should be used to store a result of functions that conventionally return an iterator or a range
Code Smell

333.

"std::has_single_bit" should be used to test if an integer is a power of two
Code Smell

334.

Empty class members should be marked as "[[no_unique_address]]"
Code Smell

335.

"std::to_address" should be used to convert iterators to raw pointers
Code Smell

336.

"[[nodiscard]]" attributes on types should include explanations
Code Smell

337.

STL constrained algorithms with range parameter should be used when iterating over the entire range
Code Smell

338.

"std::span" should be used for a uniform sequence of elements contiguous in memory
Code Smell

339.

Operator spaceship "<=>" should be used to define comparable types
Code Smell

340.

"std::midpoint" and "std::lerp" should be used for midpoint computation and linear interpolation
Code Smell

341.

"contains" should be used to check if a key exists in a container
Code Smell

342.

Free functions should be preferred to member functions when accessing a container in a generic context
Code Smell

343.

The "_t" and "_v" version of type traits should be used instead of "::type" and "::value"
Code Smell

344.

"if constexpr" should be preferred to overloading for metaprogramming
Code Smell

345.

"static_assert" with no message should be used over "static_assert" with empty or redundant message
Code Smell

346.
Redundant class template arguments should not be used
Code Smell

347.
"std::string_view" should be used to pass a read-only string to a function
Code Smell

348.
"if","switch", and range-based for loop initializer should be used to reduce scope of variables
Code Smell

349.
"std::scoped_lock" should be used instead of "std::lock_guard"
Code Smell

350.
Multicharacter literals should not be used
Code Smell

351.
Classes should explicitly specify the access level when specifying base classes
Code Smell

352.
"std::initializer_list" constructor should not overlap with other constructors
Code Smell

353.
Threads should not be detached
Code Smell

354.
Loop variables should be declared in the minimal possible scope
Code Smell

355.
"shared_ptr" should not be taken by rvalue reference
Code Smell

356.
Inheriting constructors should be used
Code Smell

357.
Return type of functions shouldn't be const qualified value
Code Smell

358.
Macros should not be used as replacement to "typdef" and "using"
Code Smell

359.
Concise syntax should be used for concatenatable namespaces
Code Smell

360.
STL algorithms and range-based for loops should be preferred to traditional for loops
Code Smell

361.
"using" should be preferred for type aliasing
Code Smell

362.
"constexpr" functions should not be declared "inline"

Code Smell

363.

"^" should not be confused with exponentiation
Code Smell

364.

Pointer and reference local variables should be "const" if the corresponding object is not modified
Code Smell

365.

Format strings should comply with ISO standards
Code Smell

366.

Functions which do not return should be declared as "noreturn"
Code Smell

367.

Macros should not be redefined
Code Smell

368.

'extern "C"' should not be used with namespaces
Code Smell

369.

"auto" should not be used as a storage class specifier
Code Smell

370.

"#include_next" should not be used
Code Smell

371.

String literals should not be concatenated implicitly
Code Smell

372.

Reference types should not be qualified with "const" or "volatile"
Code Smell

373.

Partial specialization syntax should not be used for function templates
Code Smell

374.

Alternative operators should not be used
Code Smell

375.

Types and variables should be declared in separate statements
Code Smell

376.

Scoped enumerations should be used
Code Smell

377.

"const" and "volatile" should not be used in "enum" declarations
Code Smell

378.

Jump statements should not be redundant
Code Smell

379.

"static" should not be used in unnamed namespaces
Code Smell

380.

"final" classes should not have "virtual" functions
Code Smell

381.

Redundant lambda return types should be omitted
Code Smell

382.

Declarations of functions defined outside of the class should not be marked as "inline"
Code Smell

383.

Allocation and deallocation functions should not be explicitly declared "static"
Code Smell

384.

Access specifiers should not be redundant
Code Smell

385.

The "register" storage class specifier should not be used
Code Smell

386.

"override" or "final" should be used instead of "virtual"
Code Smell

387.

Empty "case" clauses that fall through to the "default" should be omitted
Code Smell

388.

Namespaces should not be empty
Code Smell

389.

Forward declarations should not be redundant
Code Smell

390.

Members should be initialized in the order they are declared
Code Smell

391.

Declarations should not be empty
Code Smell

392.

General "catch" clauses should not be used
Code Smell

393.

"catch" clauses should do more than rethrow
Code Smell

394.

Exceptions should not be ignored
Code Smell

395.

"final" classes should not have "protected" members
Code Smell

396.

"final" should not be used redundantly
Code Smell

397.
Redundant casts should not be used
Code Smell

398.
Code annotated as deprecated should not be used
Code Smell

399.
"#pragma warning (default: ...)" should not be used
Code Smell

400.
Init-declarator-lists and member-declarator-lists should consist of single init-declarators and member-declarators respectively
Code Smell

401.
Unused local variables should be removed
Code Smell

402.
"switch" statements should have at least 3 "case" clauses
Code Smell

403.
A "while" loop should be used instead of a "for" loop
Code Smell

404.
Nested code blocks should not be used
Code Smell

405.
Overriding member functions should do more than simply call the same member in the base class
Code Smell

406.
Do not check emptiness with a size method when a dedicated function exists
Code Smell

407.
Empty statements should be removed
Code Smell

408.
"/*" and "//" should not be used within comments
Code Smell

409.
Classes should not be derived from virtual bases
Code Smell

410.
Track uses of "TODO" tags
Code Smell

411.
Deprecated code should be removed
Code Smell

412.
Reserved identifiers and functions in the C standard library should not be defined or declared

Code Smell

413.

In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##
Code Smell

414.

Bit fields should not be used
Code Smell

415.

Track lack of copyright and license headers
Code Smell

416.

Octal values should not be used
Code Smell

417.

Function templates should not be specialized
Code Smell

418.

"abort", "exit", "getenv" and "system" from <stdlib.h> should not be used
Bug

419.

"atof", "atoi" and "atol" from <stdlib.h> should not be used
Bug

420.

"<signal.h>" should not be used
Bug

421.

Dynamic heap memory allocation should not be used
Bug

422.

The global namespace should only contain "main", namespace declarations, and "extern" C declarations
Code Smell

423.

"<time.h>" should not be used
Code Smell

424.

"<stdio.h>" should not be used in production code
Code Smell

425.

"offsetof" macro from <stddef.h> should not be used
Code Smell

426.

"errno" should not be used
Code Smell

427.

"setjmp" and "longjmp" should not be used
Code Smell

428.

Function-like macros should not be used
Code Smell

429.
Macros should not be #define'd or #undef'd within a block
Code Smell
430.
Unions should not be used
Code Smell
431.
Array type function arguments should not decay to pointers
Code Smell
432.
Object declarations should contain no more than 2 levels of pointer indirection
Code Smell
433.
Recursion should not be used
Code Smell
434.
Constants of unsigned type should have a "U" suffix
Code Smell
435.
Cyclomatic Complexity of coroutines should not be too high
Code Smell
436.
Functions should not have more than one argument of type "bool"
Code Smell
437.
using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files
Code Smell
438.
Virtual functions should not have default arguments
Code Smell
439.
Octal and hexadecimal escape sequences should be terminated
Code Smell
440.
Flexible array members should not be declared
Code Smell
441.
Preprocessor directives should not be indented
Code Smell
442.
"switch" statements should not be nested
Code Smell
443.
Lambdas should not be used
Code Smell
444.
Cyclomatic Complexity of functions should not be too high
Code Smell
445.
Cyclomatic Complexity of classes should not be too high

Code Smell

446.
"switch" statements should have "default" clauses
Code Smell

447.
"if ... else if" constructs should end with "else" clauses
Code Smell

448.
"typedef" should be used for function pointers
Code Smell

449.
Control structures should use curly braces
Code Smell

450.
Expressions should not be too complex
Code Smell

451.
"<cstdio>" should not be used
Code Smell

452.
"<ctime>" should not be used
Code Smell

453.
C libraries should not be used
Code Smell

454.
Macros used in preprocessor directives should be defined before use
Bug

455.
Evaluation of the operand to the sizeof operator shall not contain side effects
Bug

456.
Bitwise operators should not be applied to signed operands
Bug

457.
Boolean operations should not have numeric operands, and vice versa
Bug

458.
Pointer conversions should be restricted to a safe subset
Bug

459.
Function pointers should not be converted to any other type
Bug

460.
Results of ~ and << operations on operands of underlying types unsigned char and unsigned short should immediately be cast to the operand's underlying type
Bug

461.
Each operand of the ! operator, the logical && or the logical || operators shall have type bool
Bug

462.

When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization
Bug

463.

User-defined types should not be passed as variadic arguments
Bug

464.

Floating point numbers should not be tested for equality
Bug

465.

Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier
Bug

466.

There shall be at most one occurrence of the # or ## operators in a single macro definition
Code Smell

467.

Parameters in a function prototype should be named
Code Smell

468.

"goto" statement should not be used
Code Smell

469.

A loop-control-variable other than the loop-counter which is modified in statement shall have type bool
Code Smell

470.

Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression
Code Smell

471.

"enum" values should not be used as operands to built-in operators other than [ ], =, ==, !=, unary &, and the relational operators <, <=, >, >=
Code Smell

472.

C-style and functional notation casts should not be used
Code Smell

473.

Operands of "&&" and "||" should be primary (C) or postfix (C++) expressions
Code Smell

474.

Limited dependence should be placed on operator precedence
Code Smell

475.

Braces should be used to indicate and match the structure in the non-zero initialization of arrays and structures
Code Smell

476.

Array declarations should include an explicit size specification
Code Smell

477.

Objects or functions with external linkage shall be declared in a header file
Code Smell

478.

"typedef" names should be unique identifiers
Code Smell

479.

Identifiers should not be longer than 31 characters
Code Smell

480.

All uses of the #pragma directive should be documented
Code Smell

481.

Assembly language should be encapsulated and isolated
Code Smell

482.

Coroutines should not have too many lines of code
Code Smell

483.

[[nodiscard]] should be used when the return value of a function should not be ignored
Code Smell

484.

Functions that are not used in a project should be removed
Code Smell

485.

Local variables should be initialized immediately
Code Smell

486.

The order for arguments of the same type in a function call should be obvious
Code Smell

487.

A cast should not convert a pointer type to an integral type
Code Smell

488.

An object with integral type or pointer to void type shall not be converted to an object with pointer type
Code Smell

489.

An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly
Code Smell

490.

Non-exception types should not be caught
Code Smell

491.

Non-exception types should not be thrown
Code Smell

492.

Binary operators should be overloaded as "friend" functions
Code Smell

493.

Track parsing failures
Code Smell

494\.

Files should not be too complex
Code Smell

495\.

The ternary operator should not be used
Code Smell

496\.

A "struct" should not have member functions
Code Smell

497\.

Default parameters should not be defined
Code Smell

498\.

Exceptions should not be used
Code Smell

499\.

Rvalue references should not be used
Code Smell

500\.

Functions/methods should not have too many lines
Code Smell

501\.

Track uses of "NOSONAR" comments
Code Smell

502\.

"::" operator should be used to access global variables and functions
Code Smell

503\.

"for" loop stop conditions should be invariant
Code Smell

504\.

Statements should be on separate lines
Code Smell

505\.

"switch case" clauses should not have too many lines of code
Code Smell

506\.

Functions should not contain too many return statements
Code Smell

507\.

Magic numbers should not be used
Code Smell

508\.

Standard outputs should not be used directly to log anything
Code Smell

509\.

Files should not have too many lines of code
Code Smell

510\.

Lines should not be too long
Code Smell

511.
"operator=" should check for assignment to self
Bug

512.
Accessible base classes should not be both "virtual" and non-virtual in the same hierarchy
Bug

513.
A variable which is not modified shall be const qualified
Code Smell

514.
Preprocessor operators "#" and "##" should not be used
Code Smell

515.
Switch statement conditions should not have essentially boolean type
Code Smell

516.
"continue" should not be used
Code Smell

517.
The loop-counter should be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop
Code Smell

518.
Signed and unsigned types should not be mixed in expressions
Code Smell

519.
typedefs that indicate size and signedness should be used in place of the basic types
Code Smell

520.
The first operand of a conditional operator should have type bool
Code Smell

521.
The condition of an if-statement and the condition of an iteration-statement shall have type bool
Code Smell

522.
Appropriate char types should be used for character and integer values
Code Smell

523.
Source code should only use /* ... */ style comments
Code Smell

524.
Concept names should comply with a naming convention
Code Smell

525.
Coroutine names should comply with a naming convention
Code Smell

526.
"std::cmp_*" functions should be used to compare signed and unsigned values
Code Smell

527.

"nodiscard" attributes on functions should include explanations
Code Smell

528.

"dynamic_cast" should be used for downcasting
Code Smell

529.

Struct should explicitly specify the access level when specifying base classes
Code Smell

530.

"std::endl" should not be used
Code Smell

531.

The identifiers used for the parameters in a re-declaration or override of a function shall be identical to those in the declaration
Code Smell

532.

A loop-control-variable other than the loop-counter shall not be modified within condition or expression
Code Smell

533.

The loop-counter shall not be modified within condition or statement
Code Smell

534.

If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=
Code Smell

535.

A for loop shall contain a single loop-counter which shall not have floating type
Code Smell

536.

Every switch statement shall have at least one case-clause
Code Smell

537.

All "if ... else if" constructs shall be terminated with an "else "clause
Code Smell

538.

An `if ( condition )` construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement
Code Smell

539.

The statement forming the body of a "switch", "while", "do {...} while" or "for" statement shall be a compound statement
Code Smell

540.

C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used
Code Smell

541.

"auto" should not be used to deduce raw pointers
Code Smell

542.

Method overloads should be grouped together in the interface
Code Smell

543.

GNU extensions should not be used
Code Smell

544.

Raw string literals should be used
Code Smell

545.

"inline" should not be used redundantly
Code Smell

546.

Digit separators should be used
Code Smell

547.

Base class access specifiers should not be redundant
Code Smell

548.

Inheritance should be "public"
Code Smell

549.

Methods should not return constants
Code Smell

550.

Label names should comply with a naming convention
Code Smell

551.

Enumeration values should comply with a naming convention
Code Smell

552.

Enumeration names should comply with a naming convention
Code Smell

553.

Namespace names should comply with a naming convention
Code Smell

554.

Comment styles "//" and "/* ... */" should not be mixed within a file
Code Smell

555.

"union" names should comply with a naming convention
Code Smell

556.

"public", "protected" and "private" sections of a class should be declared in that order
Code Smell

557.

Constants should come first in equality tests
Code Smell

558.

Type specifiers should be listed in a standard order
Code Smell

559.

C++ comments should be used
Code Smell

560.
Track "TODO" and "FIXME" comments that do not contain a reference to a person
Code Smell

561.
The prefix increment/decrement form should be used
Code Smell

562.
"struct" names should comply with a naming convention
Code Smell

563.
File names should comply with a naming convention
Code Smell

564.
Macro names should comply with a naming convention
Code Smell

565.
Comments should not be located at the end of lines of code
Code Smell

566.
Functions without parameters should not use "(void)"
Code Smell

567.
break statements should not be used except for switch cases
Code Smell

568.
Local variable and function parameter names should comply with a naming convention
Code Smell

569.
Field names should comply with a naming convention
Code Smell

570.
Lines should not end with trailing whitespaces
Code Smell

571.
Files should contain an empty newline at the end
Code Smell

572.
Tabulation characters should not be used
Code Smell

573.
Class names should comply with a naming convention
Code Smell

574.
A function should have a single point of exit at the end of the function
Code Smell

575.
"using-directives" should not be used
Code Smell

576.

Function names should comply with a naming convention
Code Smell

577.

Track comments matching a regular expression
Code Smell

578.

Track instances of the "#error" preprocessor directive being reached
Code Smell