



Go =GO

5 HTML

JavaScript

Java

Kotlin

Kubernetes

Objective C

PHP

PL/I

PL/SQL

Python

RPG

Ruby

Scala

Swift

Terraform

Text

TypeScript

T-SQL

VB.NET

VB6

XML



C++ static code analysis

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

• Security ⊗ Code (436) Quick 68 Fix ΑII 578 **R** Bug (111) 6 Vulnerability 13 Hotspot rules

Tags

"memset" should not be used to delete sensitive data Vulnerability POSIX functions should not be called with arguments that trigger buffer overflows ■ Vulnerability XML parsers should not be vulnerable to XXE attacks ■ Vulnerability Function-like macros should not be

📆 Bug

invoked without all of their arguments

The address of an automatic object should not be assigned to another object that may persist after the first

🖷 Bug

Assigning to an optional should directly target the optional

object has ceased to exist

🖷 Bug

Result of the standard remove algorithms should not be ignored

📆 Bug

"std::scoped_lock" should be created with constructor arguments

📆 Bug

Objects should not be sliced

📆 Bug

Immediately dangling references should not be created

📆 Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

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

Analyze your code

Code Smell

cppcoreguidelines performance since-c++11 clumsy

Search by name...

All special member functions (default constructor, copy and move constructors, copy and move assignment operators, destructor) can be automatically generated by the compiler if you don't prevent it (for many classes, it is good practice to organize your code so that you can use these default versions, see {rule:cpp:S4963}).

There are cases where it's still useful to manually write such a function, because the default implementation is not doing what you need. But if the manually written function is equivalent to the default implementation, this is an issue:

- · It's more code to write, test and maintain for no good reason
- Writing the code of those functions correctly is surprisingly difficult
- Once you write one such function, you will typically have to write several (see {rule:cpp:S3624})
- If you want your class to be trivial or to be an aggregate, those functions cannot be user-provided anyways

In most cases, you should just remove the code of the redundant function. In some cases, the compiler will not automatically generate the default version of the function, but you can force it to do so by using the = default syntax.

For default constructors, you will often be able to use the default version if you use in-class initialization instead of the initializer list (see {rule:cpp:S5424}). You will have to make it explicitly defaulted if your class has any other constructor.

For destructors, you may want to use the =default syntax to be able to declare it as virtual (see {rule:cpp:S1235}).

This rule raises an issue when any of the following is implemented in a way equivalent to the default implementation:

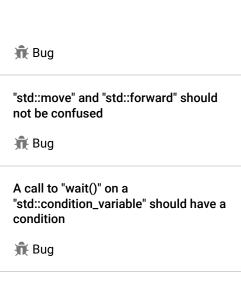
- · default constructor
- destructor
- · move constructor
- · move-assignment operator
- · copy constructor
- · copy-assignment operator

Noncompliant Code Example

```
struct Book {
  string Name;
  Book() { } // Noncompliant
  Book(const Book &Other) : Name(Other.Name) { } // Noncompli
 Book & operator = (const Book &);
};
Book &Book::operator=(const Book &Other) { // Noncompliant
  Name = Other.Name;
  return *this;
```

Compliant Solution

```
struct Book {
 string Name;
 Book() = default; // Restores generation of default
```



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

📆 Bug

Functions with "noreturn" attribute should not return

📆 Bug

RAII objects should not be temporary

📆 Bug

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

📆 Bug

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

Rug Bug

"std::auto_ptr" should not be used

📆 Bug

Destructors should be "noexcept"

👬 Bug

```
Book(const Book &Other) = default;
Book &operator=(const Book &) = default;
};

// Or, more common:
struct Book {
   string Name;
};
```

See

• <u>C++ Core Guidelines C.30</u> - Define a destructor if a class needs an explicit action at object destruction

Available In:

sonarlint in sonarcloud on sonarqube Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy