rules.sonarsource.com

# C++ static code analysis: Floating point numbers should not be tested for equality

2-3 minutes

---

Floating point math is imprecise because of the challenges of storing such values in a binary representation. Even worse, floating point math is not associative; push a `float` or a `double` through a series of simple mathematical operations and the answer will be different based on the order of those operation because of the rounding that takes place at each step.

Even simple floating point assignments are not simple:

float f = 0.1; // 0.100000001490116119384765625
double d = 0.1; // 0.1000000000000000055511151231257827021181583404541015625

(Results will vary based on compiler and compiler settings.)

Therefore, the use of the equality (==) and inequality (`!=`) operators on `float` or `double` values is very often an error.

The accepted solution is to use or write a float comparison library:

- Either using a comparison taking into account the magnitude of the numbers being compared and an epsilon value (which may be based on the capability of the floating point epsilon (FLT_EPSILON)). This comparison will often be absolute for very small values, and relative for larger ones
- Or using the notion of units in the last place

This rule checks for the use of equality/inequality tests on `floats`, `doubles` and `long doubles`.

## Noncompliant Code Example

```
float myNumber = 3.146;
if ( myNumber == 3.146 ) {  //Noncompliant. Because of floating point imprecision, this will be false
  // ...
}

if (myNumber <= 3.146 && mNumber >= 3.146) { // Noncompliant indirect equality test
  // ...
}

if (myNumber < 4 || myNumber > 4) { // Noncompliant indirect inequality test
  // ...
}
```

## See

- MISRA C:2004, 13.3 - Floating-point expressions shall not be tested for equality or inequality.
- MISRA C++:2008, 6-2-2 - Floating-point expressions shall not be directly or indirectly tested for equality or inequality
- Comparing Floating Point Numbers, 2012 Edition