FLP30-C. Do not use floating-point variables as loop counters

Because floating-point numbers represent real numbers, it is often mistakenly assumed that they can represent any simple fraction exactly. Floating-point numbers are subject to representational limitations just as integers are, and binary floating-point numbers cannot represent all real numbers exactly, even if they can be represented in a small number of decimal digits.

In addition, because floating-point numbers can represent large values, it is often mistakenly assumed that they can represent all significant digits of those values. To gain a large dynamic range, floating-point numbers maintain a fixed number of precision bits (also called the significand) and an exponent, which limit the number of significant digits they can represent.

Different implementations have different precision limitations, and to keep code portable, floating-point variables must not be used as the loop induction variable. See Goldberg's work for an introduction to this topic [Goldberg 1991].

For the purpose of this rule, a *loop counter* is an induction variable that is used as an operand of a comparison expression that is used as the controlling expression of a `do`, `while`, or `for` loop. An *induction variable* is a variable that gets increased or decreased by a fixed amount on every iteration of a loop [Aho 1986]. Furthermore, the change to the variable must occur directly in the loop body (rather than inside a function executed within the loop).

Noncompliant Code Example

In this noncompliant code example, a floating-point variable is used as a loop counter. The decimal number `0.1` is a repeating fraction in binary and cannot be exactly represented as a binary floating-point number. Depending on the implementation, the loop may iterate 9 or 10 times.

```
void func(void) {
  for (float x = 0.1f; x <= 1.0f; x += 0.1f) {
    /* Loop may iterate 9 or 10 times */
  }
}
```

For example, when compiled with GCC or Microsoft Visual Studio 2013 and executed on an x86 processor, the loop is evaluated only nine times.

Compliant Solution

In this compliant solution, the loop counter is an integer from which the floating-point value is derived:

```
#include <stddef.h>

void func(void) {
  for (size_t count = 1; count <= 10; ++count) {
    float x = count / 10.0f;
    /* Loop iterates exactly 10 times */
  }
}
```

Noncompliant Code Example

In this noncompliant code example, a floating-point loop counter is incremented by an amount that is too small to change its value given its precision:

```
void func(void) {
  for (float x = 100000001.0f; x <= 100000010.0f; x += 1.0f) {
    /* Loop may not terminate */
  }
}
```

On many implementations, this produces an infinite loop.

Compliant Solution

In this compliant solution, the loop counter is an integer from which the floating-point value is derived. The variable `x` is assigned a computed value to reduce compounded rounding errors that are present in the noncompliant code example.

```
void func(void) {
  for (size_t count = 1; count <= 10; ++count) {
    float x = 100000000.0f + (count * 1.0f);
    /* Loop iterates exactly 10 times */
  }
}
```

Risk Assessment

The use of floating-point variables as loop counters can result in unexpected behavior .

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FLP30-C	Low	Probable	Low	**P6**	**L2**

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**for-loop-float**	Fully checked
Axivion Bauhaus Suite	6.9.0	**CertC-FLP30**	Fully implemented
Clang	3.9	`cert-flp30-c`	Checked by `clang-tidy`
CodeSonar	5.2p0	**LANG.STRUCT.LOOP.FPC**	Float-typed loop counter
Compass/ROSE			
Coverity	2017.07	**MISRA C 2004 Rule 13.4** **MISRA C 2012 Rule 14.1**	Implemented
ECLAIR	1.2	**CC2.FLP30**	Fully implemented
Klocwork	2018	**MISRA.FOR.COND.FLT** **MISRA.FOR.COUNTER.FLT**	
LDRA tool suite	9.7.1	**39 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_C-FLP30-a**	Do not use floating point variables as loop counters
Polyspace Bug Finder	R2019b	CERT C: Rule FLP30-C	Checks for use of float variable as loop counter (rule fully covered)
PRQA QA-C	9.7	**3339, 3340, 3342**	Partially implemented
PRQA QA-C++	4.4	**4234**	
RuleChecker	19.04	**for-loop-float**	Fully checked
SonarQube C/C++ Plugin	3.11	**S2193**	Fully implemented
TrustInSoft Analyzer	1.38	**non-terminating**	Exhaustively detects non-terminating statements (see one compliant and one non-compliant example).

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
CERT C	FLP30-CPP. Do not use floating-point variables as loop counters	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT Oracle Secure Coding Standard for Java	NUM09-J. Do not use floating-point variables as loop counters	Prior to 2018-01-12: CERT: Unspecified Relationship

ISO/IEC TR 24772:2013	Floating-Point Arithmetic [PLF]	Prior to 2018-01-12: CERT: Unspecified Relationship
MISRA C:2012	Directive 1.1 (required)	Prior to 2018-01-12: CERT: Unspecified Relationship
MISRA C:2012	Rule 14.1 (required)	Prior to 2018-01-12: CERT: Unspecified Relationship

Bibliography

[Aho 1986]	
[Goldberg 1991]	
[Lockheed Martin 05]	AV Rule 197