Facts

Here are some facts regarding embedded programming with C++, as well as MISRA C++ and AUTOSAR C++14 guidelines.

WE'LL COVER THE FOLLOWING

MISRA C++

The current MISRA C++:2008 guidelines were published by the **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation. They are based on the MISRA C guidelines from 1998. Originally designed for the automotive industry, MISRA C++ became the standard for the implementation of critical software in the aviation, military, and medical sector. Just like MISRA C, MISRA C++ also describes guidelines for a safe subset of C++.

This subset consists of more than 200 rules classified as a document, required, or advisory.

- Document:
  - Mandatory requirements on the developer
  - Derivations are not permitted
- Required:
  - Mandatory requirements for the developer
  - Formal derivation must be raised
- Advisory:
  - Should be followed as closely as possible

    - Formal derivation is not necessary but may be considered

MISRA C++ Rules

Lets's look at some of the important rules regarding the C++ core language and libraries. To make it clearer, we will present a few rules from MISRA C++.

- Unnecessary construct
    - The project shall not contain unreachable code. (required)
    - The project shall not contain unused variables. (required)
- Assembler
    - All usage of assembler shall be documented. (document)
- Arithmetic
    - Use of floating-point arithmetic shall be documented. (document)
- Language
    - The code shall conform to the C++03 standard (Remark: Small addition to C++98). (required)
- Comments
    - No C comments shall be used to "comment out" code. (required)
    - No C++ comments shall be used to "comment out" code. (advisory)
- Pointer conversions
    - NULL shall not be used as an integer value. (required)
- Multiple base classes
    - Classes should not be derived from virtual bases. (advisory)
- Virtual functions
    - Each overriding virtual function shall be declared with the virtual keyword. (required)
- Exception handling
    - Exceptions shall only be used for error handling. (document)
- Templates
    - All partial and explicit specializations for a template shall be declared in the same file as the declarations of their primary template. (required)
- Macro replacements
    - The # and ## operators should not be used. (advisory)
- Library

    - The C library shall not be used. (required)
    - All library code shall conform to MISRA C++.(document)

You can verify these and all the other MISRA C++ rules with static code analysis tools.

Conclusion

Which conclusions can we draw from the MISRA C++ rules for the usage of C++ in critical systems? Neither one feature nor the whole language is excluded by MISRA C++.

MISRA C++ also emphasizes why C++ in critical systems becomes more important. (1.1 The use of C++ in critical systems):

- C++ offers support for high-speed, low-level, input/output operations, which are essential to many embedded systems.
- The increased complexity of applications makes the use of a high-level language more appropriate than assembly language.
- C++ compilers generate code with similar size and RAM requirements to those of C.

One small issue remains, however. MISRA C++ is based on classical C++, while Modern C++ has more to offer for embedded systems. Sadly, MISRA C++ cannot keep in lockstep with the C++ standardization but there are efforts being made to fill the gap.

AUTOSAR C++14 Guideline

AUTOSAR C++14 (**AUT**omotive **O**pen **S**ystem **AR**chitecture) consists of guidelines for the usage of C++ in safety-critical systems. AUTOSAR C++14 is based on a more updated MISRA C++ ,and it is used more frequently in the automotive domain. AUTOSAR C++14 is under active development.

In contrast to MISRA C++, AUTOSAR C++14 allows the usage of dynamic memory allocation and operator overloading.

AUTOSAR C++14 disallows the following points:

- `malloc`, `free`, and C-casts
- `const_cast`, `dynamic_cast`, and `reinterpret_cast`

- `typedef` name
- Unions
- Multiple inheritances
- `friend` declarations
- Dynamic exception specifications (`throw`)

AUTOSAR C++14 overcomes the biggest weakness of MISRA C++. In contrast to MISRA C++14, AUTOSAR C++14 is based on a more updated C+14. However, it shares one significant weakness with MISRA C++: both are the result of formal standardization, meaning it is challenging to keep it standardized with C++.

The driving force behind MISRA C++ and AUTOSAR C++14 is the automotive industry. This does not hold for the C++ core guidelines, which are a C++ community-driven project.

C++ Core Guidelines

The C++ core guidelines are a community-driven project, which is hosted on Github. The editors are Bjarne Stroustrup and Herb Sutter.

The abstract to the C++ core guidelines describe their goals:

> "This document is a set of guidelines for using C++ well. The aim of this document is to help people to use modern C++ effectively. By "Modern C++" we mean C++11 and C++14 (and soon C++17)."

The C++ core guidelines have a similar scope as MISRA C++ and AUTOSAR C++14. The additional guideline support library (GSL) helps to verify the rules of this set of guidelines.

The rules of the C++ core guidelines deal primarily with the following concerns.

- Interfaces
- Functions
- Classes and class hierarchies
- Enumerations
- Resource management

- Expressions and statements
- Error handling
- Constants and immutability
- Templates and generic programming
- Concurrency
- The Standard library
- Source files
- C-style programming

More posts to the C++ core guidelines can be found here: Modernes C++.

You can find more info regarding the C++ Core Guidelines here:

- https://www.modernescpp.com/index.php/what-is-modern-c
- https://www.modernescpp.com/index.php/c-core-guidelines-the-philosophy

---

In the next lesson, we will discuss the Technical Report on C++ Performance.