MISRA-C

Alena Tesařová, Stylianos Tsagkarakis, Vasileios Konstantaras

Faculty of Engineering of the University of Porto

May 6, 2020

Outline

What is MISRA-C?

MISRA-C is a set of software development guidelines for the programming language C in critical systems, developed by **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation.

- 1st Edition 1998 (127 coding rules)
- 2nd Edition 2004 (142 coding rules)
- 3rd Edition 2012 (143 coding rules)
- 2016 & 2020 Amendment

Use of C in Embedded Systems

+ easy access to hardware
+ efficient run-time performance
+ low memory requirements

– has limited run-time checking
– a programmer can easily make a mistake
– compilers might contain errors

Today, MISRA is used in fields such as:

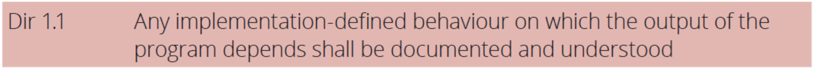
- Automotive Industry
- Railway Systems
- Aerospace Industry
- Telecommunications
- Medical Devices

Using MISRA standards companies ensure their code is:

- Safe
- Secure
- Reliable
- Portable

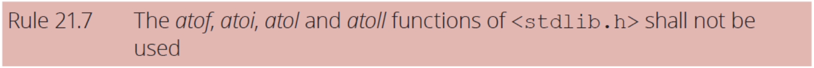
Guideline types

– Directives
- does not have to be well defined
- often address *process* or *documentation* requirements



Dir 1.1 Any implementation-defined behaviour on which the output of the program depends shall be documented and understood

– Rules
- formally and well defined
- compliance is depended entirely on the source code

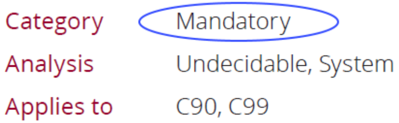


Rule 21.7 The *atof*, *atoi*, *atol* and *atoll* functions of `<stdlib.h>` shall not be used

Guideline Classification

Guideline categories

- Mandatory
  - deviations and violation are not permitted
- Required
  - mandatory requirements
  - can be treated as Mandatory
  - deviation allowed, but must follow some formalization
- Advisory
  - recommendations, can be treated as Mandatory or Required



Category	Mandatory
Analysis	Undecidable, System
Applies to	C90, C99

Guideline Classification

Decidability of rules

- Decidable – can be verified by a program
- Undecidable – if it is not decidable

Example of a decidable rule

Rule 5.2: identifiers declared in the same scope and namespace shall be distinct

Example of a undecidable rule

Rule 2.1: A project shall not contain unreachable code

Organization of the rules

- 1. Environment
- 2. Language Extensions
- 3. Character set
- 4. Documentation
- 5. Identifiers
- 6. Types
- 7. Constants
- 8. Declarations & Definitions
- 9. Initialization
- 10. Arithmetic Type Conversions
- 11. Pointer Type Conversions
- 12. Expressions
- 13. Control Statements
- 14. Control Flow
- 15. Switch statements
- 16. Functions
- 17. Pointers & Arrays
- 18. Structures & Unions
- 19. Preprocessing directives
- 20. Standard Libraries
- 21. Run-time failures

8.3 Comments

Rule 3.2: Line-splicing shall not be used in // comments

```
extern bool_t b;
void f ( void )
{
    uint16_t x = 0; // comment \
    if ( b )
    {
        ++x; /* This is always executed */
    }
}
```

- required
- the new line becomes a part of the comment

8.5 Identifiers

Rule 5.3: A typedef name shall be a unique identifier.

The following example is NOT acceptable:

```
{ typedef unsigned char uint8_t;}
{ typedef unsigned char uint8_t;} //NOT compliant
    - redefinition
{ unsigned char uint8_t;} //NOT compliant - reuse
    of int8_t
```

- required
- redefinition of typedef is not allowed

8.13 Side Effects

Rule 13.3: A full expression containing an increment(++) or decrement(–) operator should have no other potential side effects other than that caused by the increment or decrement operator

```
u8a = ++u8b + u8c--;
// is clearer when written as the following
    sequence:
++u8b;
u8a = u8b + u8c;
u8c--;
```

- advisory

8.15 Control Flow

Rule 15.1: Advisory: The goto statement **should not** be used.

Rule 15.2: Required: The goto statement **shall** jump to a label declared later in the same function.

```
L1: ++i;
    if i>10
        goto L2; // compliant
L2: ++j;
    if j>20
        goto L1; // NOT-compliant
```

Using MISRA-C (1)

Things to take into consideration to comply with MISRA.

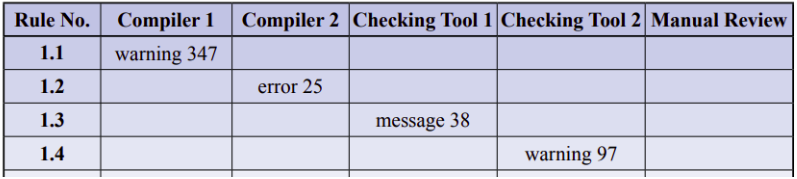
1. Know the Rules
2. Check Your Code Constantly
3. Set Baselines
4. Prioritize Violations Based on Risk
5. Document Your Deviations
6. Monitor Your MISRA Compliance
7. Choose the Right Static Code Analyzer

Using MISRA-C (2)

Compliance Matrix

1. Cross - compiler
2. Different Tools
3. Manual Inspection

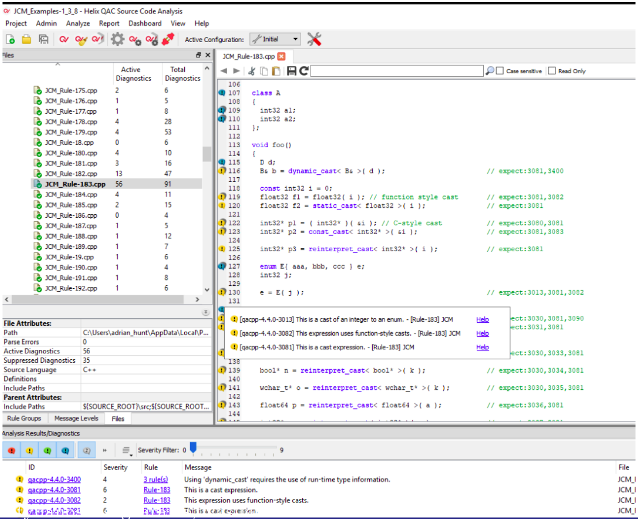
If any specific restrictions are omitted there should be full justification.



Rule No.	Compiler 1	Compiler 2	Checking Tool 1	Checking Tool 2	Manual Review
1.1	warning 347				
1.2		error 25			
1.3			message 38		
1.4				warning 97	

Figure 1: Compliance Matrix

Helix static analysis tool


JCM_Examples-1_3_8 - Helix QAC Source Code Analysis
Project Admin Analyze Report Dashboard View Help
Active Configuration: Initial
Files
JCM_Rule-183.cpp
Case sensitive
Read Only
Rule Groups Message Levels Files
Analysis Results/Diagnostics
Severity Filter: 0 9

	Active Diagnostics	Total Diagnostics
JCM_Rule-175.cpp	2	6
JCM_Rule-176.cpp	1	5
JCM_Rule-177.cpp	1	8
JCM_Rule-178.cpp	4	28
JCM_Rule-179.cpp	4	53
JCM_Rule-18.cpp	0	6
JCM_Rule-180.cpp	4	10
JCM_Rule-181.cpp	3	16
JCM_Rule-182.cpp	13	47
JCM_Rule-183.cpp	56	91
JCM_Rule-184.cpp	4	11
JCM_Rule-185.cpp	2	15
JCM_Rule-186.cpp	0	4
JCM_Rule-187.cpp	1	5
JCM_Rule-188.cpp	1	12
JCM_Rule-189.cpp	1	7
JCM_Rule-19.cpp	1	6
JCM_Rule-190.cpp	1	4
JCM_Rule-191.cpp	1	8
JCM_Rule-192.cpp	1	6

File Attributes:	
Path	C:\Users\adrian_hunt\AppData\Local\P...
Parse Errors	0
Active Diagnostics	56
Suppressed Diagnostics	35
Source Language	C++
Definitions	
Include Paths	
Parent Attributes:	
Include Paths	${SOURCE_ROOT}\src;${SOURCE_ROOT...

```

class A
{
  int32 a1;
  int32 a2;
};

void foo()
{
  D d;
  B& b = dynamic_cast< B& >( d );                        // expect:3081,3400

  const int32 i = 0;
  float32 f1 = float32( i ); // function style cast      // expect:3081,3082
  float32 f2 = static_cast< float32 >( i );              // expect:3081

  int32* p1 = ( int32* )( &i ); // C-style cast          // expect:3080,3081
  int32* p2 = const_cast< int32* >( &i );                // expect:3081,3083

  int32* p3 = reinterpret_cast< int32* >( i );           // expect:3081

  enum E{ aaa, bbb, ccc } e;
  int32 j;

  e = E( j );                                            // expect:3013,3081,3082

                                                         ect:3030,3081,3090
                                                         ect:3031,3081

                                                         ect:3030,3033,3081

  bool* n = reinterpret_cast< bool* >( k );              // expect:3030,3034,3081

  wchar_t* o = reinterpret_cast< wchar_t* >( k );        // expect:3030,3035,3081

  float64 p = reinterpret_cast< float64 >( a );          // expect:3036,3081

```

```
[qacpp-4.4.0-3013] This is a cast of an integer to an enum. - [Rule-183] JCM     Help
[qacpp-4.4.0-3082] This expression uses function-style casts. - [Rule-183] JCM   Help
[qacpp-4.4.0-3081] This is a cast expression. - [Rule-183] JCM                   Help
```

	ID	Severity	Rule	Message	File
●	qacpp-4.4.0-3400	4	3 rule(s)	Using 'dynamic_cast' requires the use of run-time type information.	JCM_I
●	qacpp-4.4.0-3081	6	Rule-183	This is a cast expression.	JCM_I
●	qacpp-4.4.0-3082	2	Rule-183	This expression uses function-style casts.	JCM_I
●	qacpp-4.4.0-3081	6	Rule-183	This is a cast expression.	JCM_I

Conclusion

Why MISRA C?

- Maximizes effectiveness
- Documenting the disadvantages and limitations
- Guiding developers to use them in their advantage

" Using C weaknesses to your own gain. "

References

- *MISRA C:2012 - Addendum 2: Coverage of MISRA C:2012 against ISO/IEC TS 17961:2013 "C Secure", ISBN 978-906400-18-7 (PDF), Second Edition, January 2018.*
- *MISRA C:2004 Permits: Deviation permits for MISRA compliance, ISBN 978-906400-14-9 (PDF), Edition 1, April 2016.*
- *Introduction to MISRA C. [Online, last update 1.7.2002]. URL* *https://www.embedded.com/introduction-to-misra-c/*
- *3 Examples of Better Embedded Coding with MISRA [Online, last update 6.1.2020]* *https://www.perforce.com/blog/qac/3-examples-better-embedded-coding-misra*
- *MISRA C and MISRA C++* *https://www.perforce.com/resources/qac/misra-c-cpp*

" MISRA-C will make your code not MISRAble "