WG 21-SG 12/MISRA C++ Meeting

6 Nov 2019

Attendees

Chris Tapp Chair MISRA C++, LDRA

Michael Wong SG 12

Lisa Lippincott SG 12 Tanium (cyber security)

John MacFarlane

Clive Pygott LDRA, WG 14 MISRA

Peter Sommerlad SG 12

Stephen Michell WG 23

Billy Baker Aviation

Richard Corden - Perforce

Mateusz Pusz Bosch

John MacFarlane Land Rover software Centre

Andrez Kremenski

Andrea Weis BMW and AutoSAR, now MISRA

aaa Rose (?)

Michael Spurtis Semantic (now Norton Lifelock)

Bob Seigal Safety medical imaging

Daniel Wright PhD student

Detlef Vollman

Charles

Mike

Frank Birbacher

Erhard Ploedereder

Introduction – Michael Wong

MISRA

Machine-checkable rules

Mandatory

Required

Advisory

Directives – not completely machine checkable

Michael has a password protected PDF that is sharable with the team

Goal of the meeting is to review MISRA C++ guidelines to reflect C++ 2011, 2014 or 2017 capabilities.

Chris – rationale for MISRA C++

Approach from US DoD that would allow them to use C++ in high integrity projects.

Noted that difficult to itemize undefined behaviours in code.

Many objections to MISRA is that guidance such as “do not use exceptions” is problematic for the community. MISRA really says only use it if you can show that it can be used safely.

Discussions of ways to improve sharing without opening up the document to public access.

Examining specific guidelines

Example of single exit

Example of dead code vs unused code

Essential types

Mechanism to make code behave predictably. Casting and promotions interfere with the the correctness of calcualtions.

Look at section 6.29

Rule 000331 : Every static function and private member function of all classes in a project shall be used.

Want to permit unused functions in library code.

What does “used” mean? - called?

Use “member function” instead of “method”

Want to use more specific C++ terms such as “ODR-used” vs “used”

The following appear to be new rules from AutoSAR that need review.

rule 29.1 The identifier name of a non-member object or function with internal linkage should not be reused

What about name space?

John – this has a scaling issue. Auto systems are now large code basis.

Rule will be dropped in favour of a document that addresses name-hiding. Reviewing all naming rules.

Q – is there a set of higher level goals “

A – yes, no undefined behaviour, no unspecified behaviour, , improve code readability, improve maintainability

Rule 29.2 Type wchar\_t shall not be used

General agreement

rule 29.3 Return type of a value returning lambda expression should be ???

Likely to be deleted.

Rule 29.4 Enumeration types should be declared as scoped enum classes.

These enum's cannot be used in the fundamental array types. (i.e. you cannot write x[enum\_literal])

You can cast to more traditional types, but more guidance would be needed for that.

It is “advisory” so likely the right level.

Discussion of using unscoped enums to declare named integers. Other C++ methods such as named constants have different rules. Section 7.1.1 and 7.1.2. constexpr x should not have run-time linkage but many implementations put the value in the runtime code. Recommend using constexpr in place of untyped enums.

Side discussion:

Write own classes vs using std library – MISRA almost never recommends the use of std library.

Exceptions, memory allocator can lead to unsatisfactory behaviour.

Erhard proposes looking at Microsoft internal guidelines, Suggest mining those rules.

2.5 Braced-initialization {} shall be used ...

Drop rule.

Preference for {{ values }}

Note that in templates, one should not use {}. Maybe this is the rule.

Michael proposes a group discusses the variations

Andre, Mateusz, Michael, to develop a new rule and forward to MISRA

Rule 29.6 Variable of type auto shall be initialized using braced initialization

AUTOSAR says opposite.

N3.861 has an explnation.

Initializers that use initializer and = initializer

“auto” appears often on the defect list

Significant discussion about alternate ways to use constructors, braces, etc.

Do not use foo = {list} construct. Assignment never happens in declaration statements.

Will be rewritten by Andre and Mateusz

Discuss having multiple versions of the rules. Acknowledged that this will likely be necessary.

29.7 Give to Andre and Mateusz

29.8 A non-virtual member function shall not be defined in a derived class

Update to address name hiding and class hierarchy – screws up polymorphism

Lisa proposes rewrite

 A member declaration shall not hide any member of the base class.

General acceptance.

Notes: overriding is not hiding.

 Hiding can be prevented with the “using” declaration.

29.9

After lunch, discussions that MISRA want guidance about:

two rules that conflict:

 Every switch will have a default

 No unreachable code

Propose a single rule that says that every value of an enum must be covered by a switch alternative (no default)

Leave ban on unreachable code.

Discussions on how a program can fail. Discussion on how to add “unreachable” defensive code.

Reasonable to have a rule that says not to use “volatile” (except in specified situations)

Conclusion:

 Research uses of volatile.

Back to Essential Types

Look at C promotions and implicit conversions.

Annex D of MISRA C edition 3

Not written for C++ yet.

Areas of concern:

 const and const expressions and their representation

More review of rules

Lifetimes and temporaries reviewing P1906R0 from Peter Sommerlad

 Peter presents alternatives to using pointers

 use of unique\_ptr<T> and optional<T>

 notion of “ownership” - essentially taking responsibility for ending T

 Significant disagreement with some of the premises.

 for nullable referenced T propose use

 object\_ptr<T> const (from jss) or

 optional<T&> const (from boost) by A. Williams

Example from Frank to show how to employ a using declaration to avoid hiding names from a base class.

#include <cmath>

struct Base {

 int value;

 void from(int v) { value = v; }

};

struct Dev : Base {

 using Base::from; // avoid hiding

 void from(float f) { value = std::ceil(f); }

};

int main() {

 Dev d;

 d.from(3); // calls from(int)

 d.from(3.75f); // calls from(float)

 return d.value;

}

Example from Peter Sommerlad

//============================================================================
// Name : assign.cpp
// Author :
// Version :
// Copyright : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <memory>
#include <cassert>
using namespace std;

struct X{
 X(int i):i{i}{}
 X& operator = (X const &) & = default;
 X operator = (X const &x) && { i = x.i; return \*this;}
 int i;
 ~X() { i = 42;}
};

void foo(X& x){
 x = X(42);
}

auto& bar(unique\_ptr<int> &pi){
 return pi = make\_unique<int>(42);
}

int main() {
 auto&& x = (X{1} = X{2});
 assert(2 == x.i);
 using upi = unique\_ptr<int>;
 auto && ppp = bar(upi{} = upi{});
 cout << "!!!Hello World!!!"

 << ppp.get()
 << endl; // prints !!!Hello World!!!
 return 0;
}

Paper has recommendations fro SG 12 and WG 23. For discussion at WG 23.

No vote on paper. Author has sufficient feedback.