An Introduction to MISRA C:2012

Getting familiar with the new MISRA C language subset

www.ldra.com

Introduction

MISRA C is a language subset of the C programming language (often referred to colloquially as a “coding standard”) that is developed and maintained by the Motor Industry Software Reliability Association (MISRA). Originally designed to promote the use of the C language in safety-critical embedded applications within the motor industry, the original version was released to target C90 in 1998 (MISRA C:1998).

Over the years, the language subset has gained widespread acceptance for safety-, life-, and mission-critical applications in aerospace, telecom, medical devices, defence, railway, and other industries. The 2004 version (MISRA-C:2004) was renamed to reflect that more widespread use, and included a host of extensions and improvements to the original version.

The latest update, MISRA C:2012, is being released in early 2013 and will provide support for ISO 9899:1999 (C99) while retaining support for C90. The new language subset will help mitigate software-related risks for safety-critical applications, while allowing programmers to spend more time coding and less time on compliance efforts. In the updated language subset, rules have been made more precise so that the language subset will not prevent reasonable uses or behaviours that have no undesirable consequences. In addition developers will now have better guidance on rules enforcement, such as whether a rule defines a general behaviour across the project or only specific cases.

A design objective for the 2012 version was to ensure that, wherever possible, the thrust of each rule would allow it to be “decidable”—so that an analysis tool can always determine compliance or non-compliance, as opposed to “undecidable” - generally due to pointer or data values affecting control flow – Figure 1. Undecidable rules can result in false-positive or false-negative test results because the tool has inadequate information available to it during analysis. This improvement significantly reduces manual code-review requirements.

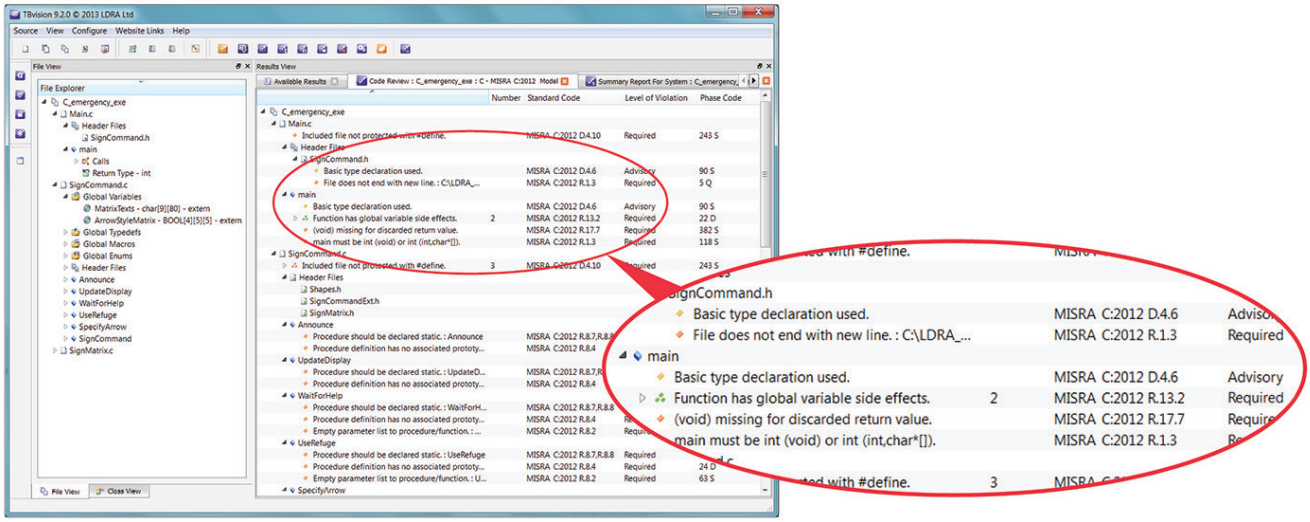

Figure 1: TBvision can identify “decidable” MISRA C:2012 rule violations and help in the checking of some “undecidable” rules. Hyperlinks both to the offending code and to descriptions of the violations help in their resolution. Such convenience complements the more user friendly nature of this latest MISRA standard.

Whether programmers are moving to C99 to take advantage of the latest language functionality or have on-going C90-based projects, MISRA C:2012 will help them meet stringent industry certification requirements for a wide range of safety-critical applications.

Developing a new version

There were some disadvantages to consider before deciding to develop a third MISRA C version. It would mean, for example, that developers would be required to learn a new version, and that tool vendors would need to develop supporting tools for it.

Ultimately, these disadvantages were minimised through a process of refinement in the face of feedback and experience from the earlier revisions. However, although the requirement to support C99 was the key motivator in the decision to develop a third version of the MISRA C standard, the revisions go much further than simply adding new rules to cover any new C99 functionality.

In evolving MISRA-C:2004 into MISRA C:2012, a review of the rules followed some suitability "rules of thumb":

- It was to be made clear which versions of the C standard each rule applies to - C90, C99, or both.
- All instances of Undefined and Unspecified C language behaviour would be covered by a rule.
- All rules would be categorised as "Decidable" or "Undecidable". "Undecidable" rules were to be assessed further to see whether they could be repositioned a little to become "Decidable". (Most remaining "Undecidable" rules deal with pointer or data values which affect control flow).
- Rule enforcement procedure would be described by a single line "Headline text", complemented in more detail by means of "normative text".
- Rule precision was to be such that only problem uses were to be restricted, and hence that reasonable uses were not.

In general, the net result is a document which is much more educational and informative than its predecessors, which were more prescriptive in nature.

Examples of specific beneficial changes

This evolution of the MISRA C standard into a more user-friendly document can be seen in a number of different ways.

Introduction of "mandatory" rules

In addition to existing "Required" and "Advisory" rules, the new "Mandatory" category lists rules which must NEVER be broken (Figure 2). The other categories can be broken with varying degrees of justification required, so that "advisory" might be at a programmer's discretion while "required" might require the approval of his/her superior.

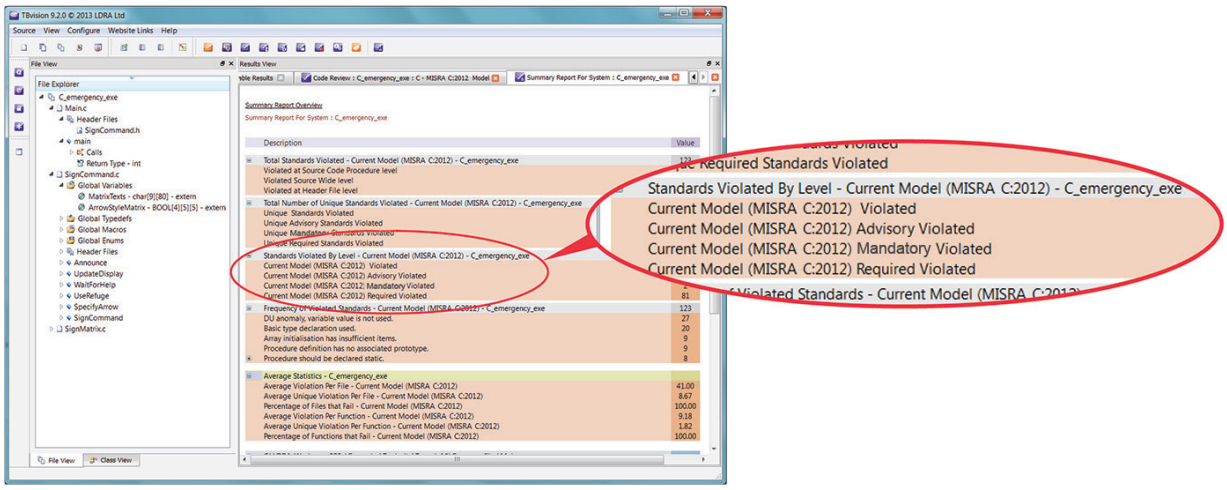

Figure 2: MISRA C:2012 introduces a "Mandatory" category for rules which must never be broken to complement the existing "Required" and "Advisory" categories. As illustrated, TBvision can summarise information on any identified violations.

For example:

1. *Rule 22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function.*

   There have been instances of developers freeing memory automatically allocated to variables for use elsewhere. This remains possible and is legitimate C syntax, but is dangerous and unnecessary. The rule is designed to prevent developers being "too clever for their own good."

   ```
   void fn ( void )
   {
   int32_t a;
   free ( &a ); /* Non-compliant - a does not point to allocated storage */
   }
   ```

2. *Rule 22.4 There shall be no attempt to write to a stream which has been opened as read-only.*

   ISO/IEC 9899 (i.e., the C language standard) does not specify the behaviour if an attempt is made to write to a read-only stream. For this reason it is considered unsafe to write to a read-only stream.

   This rule is designed to stop mistakes. There is no benefit from attempting to do this and so it is unlikely to be deliberate.

Variation of existing rules

Existing rules have been revisited, refined, adjusted and justified. MISRA C:2012 enhances the established concept of "rationale" descriptions of why each rule is a good idea. This approach is beneficial both to those looking to implement MISRA C:2012 in its entirety, and those looking to use it as the basis for in-house standards (Figure 3).

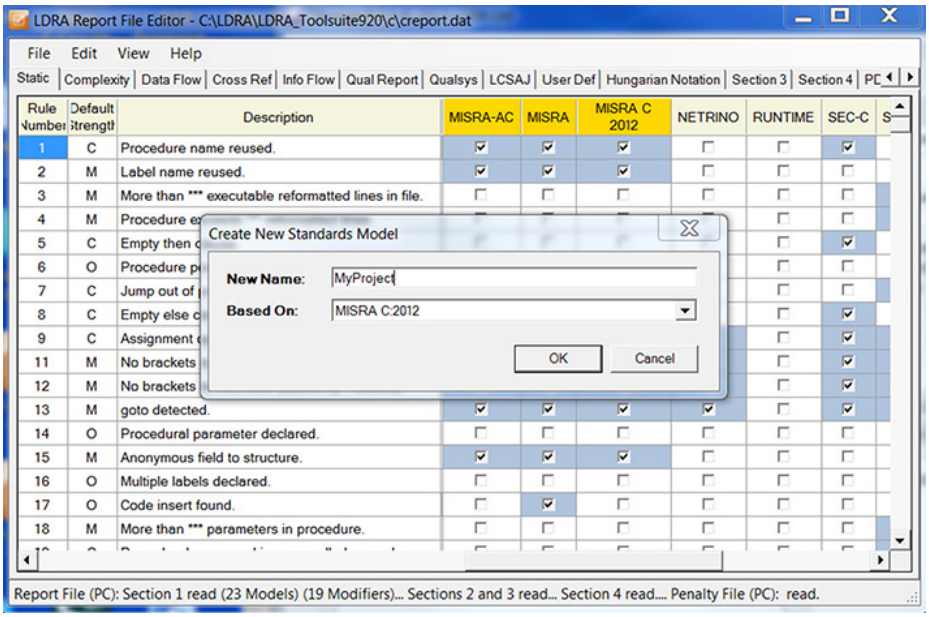

Figure 3: TBvision and TBrules both allow MISRA standards to be used as the basis for company or project specific rule sets, so that in-house rules can be added and selected MISRA rules disabled.

For example:

1. There are places where the use of the "goto" statement is justified, but all too often in the past it has been used to patch up woolly thinking or an ill-defined algorithm.

   Suppose there is an emergency situation in a process control application. Is it really better to set a flag and check it later in the algorithm than to take a direct route via a goto?

   Rule 15.1 The goto statement should not be used is now advisory rather than required, and there are an additional two required rules to narrow down the circumstances under which it is acceptable, vis.

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

   Rule 15.3 Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.

2. *Rule 12.1 The precedence of operators within expressions should be made explicit* replaces a rule from the 2004 language subset vis. "Limited dependence should be placed on C's operator precedence rules in expressions". The improved precision both in the wording and in the form of a decision table provided with the new language subset will lead to both users and (importantly) tool developers having a more consistent interpretation.

Refinement and clarification of an approach to implement specific behaviour

From the perspective of individuals involved with the hardware/software interface there are several revisions which make the new language subset better defined.

For example:

- The directive on the use of typedefs has been revised to “Advisory” from “Required”.
  Dir 4.6 typedefs that indicate size and signedness should be used in place of the basic numerical types. For example, on a 32-bit C90 implementation the following definitions might be suitable:

  typedef signed char int8_t;
  typedef signed short int16_t;
  typedef signed int int32_t;
  typedef signed long int64_t;

  From the perspective of portability, the rationale debunks the possible false perception that adherence to this guideline guarantees portability because the size of the *int* type may determine whether or not an expression is subject to integral promotion. For example, an expression with type *int16_t* will not be promoted if *int* is implemented using 16 bits but will be promoted if *int* is implemented using 32 bits.

Distinction of “System” wide versus “Single Translation Unit” analysis

Although all “Decidable” rules are theoretically decidable, it does not follow that all compliance checkers can check all “Decidable” rules. For example, the more sophisticated tool suites are capable of system wide analysis (analogous to the work of a linker) whereas inferior tools offer only translation unit analysis (analogous to the compiler).

Rules are further categorised as requiring “System” wide versus “Single Translation Unit” analysis, identifying those rules which will require an alternative approach to checking where a tool is in use which is capable of only the latter.

For example:

- *“Rule 22.3 The same file shall not be open for read and write access at the same time on different streams.”* provides a good example of a rule requiring system wide analysis. Like rule 22.4 above, it is unlikely that anyone would do this deliberately but a raised violation will help to prevent mistakes. A tool can only help confirm or deny a violation if it can reference all source code in that system through System Wide Analysis.

Summary

MISRA C:2012 is an evolution of the earlier MISRA C standards, ensuring that its contents will feel familiar for existing MISRA users while providing additional benefits for newcomers too. It benefits not only from the addition of rules to accommodate C99 functionality but also from improved rule precision, better rule categorisation, and more comprehensive explanation to educate as well as instruct.

With the aid of appropriate checking tools, it provides invaluable assistance to any organisation looking to protect themselves from the problems inherent in the inadvertent or deliberate misuse of the C language.

www.ldra.com

LDRA

LDRA UK & Worldwide
Portside, Monks Ferry, Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology, Inc.
2540 King Arthur Blvd, Suite #228 Lewisville Texas 75056
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd
#2989/1B, 3rd Floor, 12th Main, 80 Feet Road,
HAL II Stage, Bangalore- 560008. Near BSNL Building
Tel: +91 80 4080 8707
e-mail: india@ldra.com