

Embedded Systems: C and C++ Tools Reduce Code Errors

Jon Titus, Senior Technical Editor

[C and C++ Tools Reduce Code Errors](#)

Tools find problems before you compile code.

by Jon Titus, Senior Technical Editor



Programmers now have many tools that help reduce or eliminate problems. Unfortunately, they might not know these tools exist. "In 1998, the UK's Motor Industry Software Reliability Association (MISRA) published their standard for the C language to promote 'safe C' in the UK automotive industry," explained Chris Tapp, a field-applications engineer at LDRA. "The software industry has seen MISRA-C as a way to encourage good programming practice, focus on coding rules, and ensure well designed and tested safe code."

David Ward, MISRA project manager at MIRA, Ltd. explained, "Soon, programmers in other industries realized how MISRA-C would benefit them and it seems fair to say MISRA C has become the de facto standard for writing embedded C code for safety-critical applications." MISRA released a new version of its guidelines in 2004 to taking into account its use outside the automotive industry.

"The MISRA-C rules don't exist only to impose pass/fail criteria on code," noted Ward. "They make programmers aware of C-language issues and help them write code that avoids problems from the start. Tools aren't perfect and tool vendors implement MISRA-C checking in various ways, so you must understand what a tool tells you so you can spot code violations. We want programmers to write code that avoids problems rather than rely on a tool to see if code passes the MISRA checks."

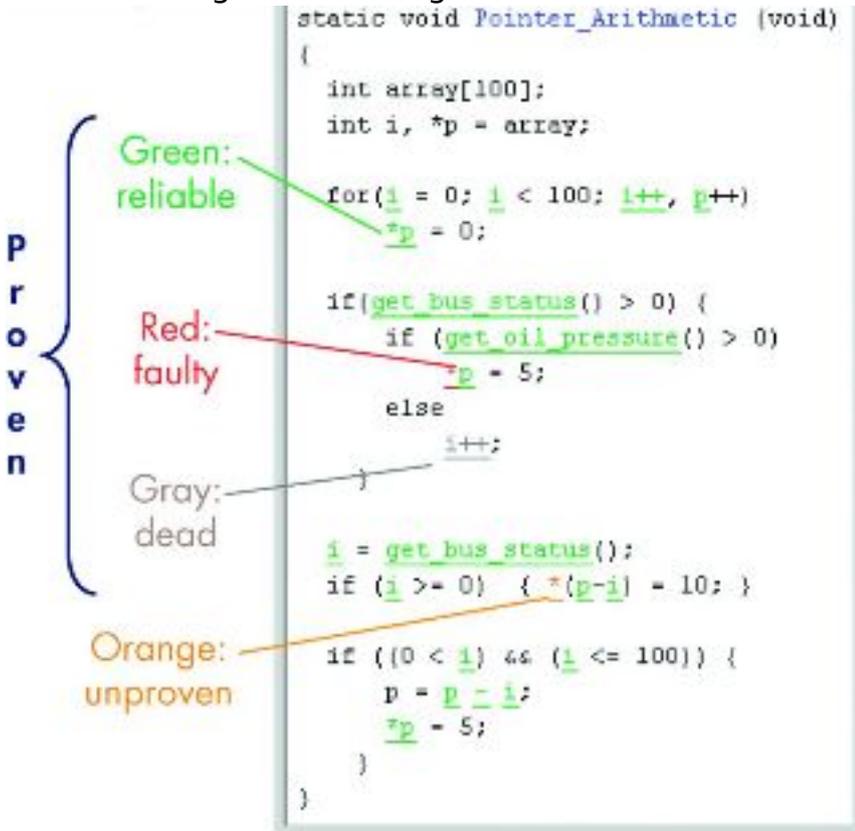
The MISRA-C standard does not comment on the suitability of C++ in safety-related systems. "That presented a problem for Lockheed Martin when Joint Strike Fighter (JSF) contractors standardized on both C and C++," said LDRA's Tapp. "So Lockheed

Embedded Systems: C and C++ Tools Reduce Code Errors

Published on Electronic Component News (<http://www.ecnmag.com>)

Martin took the MISRA-C guidelines and added its own set of rules specific to C++. But the industry preferred a more generic standard, so MISRA developed a C++ standard, released in June 2008."

Programmers unfamiliar with MISRA-C might have heard of LINT, a program that checks code for a variety of problems. The name explains itself--bits of faulty or poorly written code. LINT software exists in many forms. PC-lint, available from Gimpel Software, will create over 600 warning-like messages that indicate problems in code. "Programmers might not be aware of some C



```
static void Pointer_Arithmetic (void)
{
    int array[100];
    int i, *p = array;
    for(i = 0; i < 100; i++, p++)
        *p = 0;
    if(get_bus_status() > 0) {
        if (get_oil_pressure() > 0)
            p = 5;
        else
            i++;
    }
    i = get_bus_status();
    if (i >= 0) { *(p-i) = 10; }
    if ((0 < i) && (i <= 100)) {
        p = p - i;
        *p = 5;
    }
}
```

problems, such as order of evaluation," said Jim Gimpel, the company's owner. "An expression can be evaluated differently depending upon the compiler, the operating system, or whatever. Someone told me a Microsoft compiler would evaluate the same expression differently depending upon whether optimization was turned on or off. That's the type of thing PC-lint will pick up."

"We also have inter-function value tracking in which actual argument values initialize parameters, compute return values, go through multi-pass operations and look deep into function behavior," said Gimpel. "PC-lint lets us focus on an algorithm and pick up small problems later. As an example, we report if a variable, function, or macro doesn't get used. If you see an unused macro, maybe you forgot to write a section of code to use it. We can't guarantee code will work the way you want it to, but PC-lint lets you work at a higher level."

Besides locating problem code, you may need to prove your code will not cause run-time errors. "A typical run-time error might be a division by zero," said Paul Barnard, marketing director for design automation at The MathWorks. "Our PolySpace series of tools use formal mathematics to analyze the code and look for problematic patterns such as divide by zero, accessing something beyond the end of an array,

Embedded Systems: C and C++ Tools Reduce Code Errors

Published on Electronic Component News (<http://www.ecnmag.com>)

an overflow, an underflow, and so on. We use formal mathematics to detect run-time errors before you compile and execute code. What's powerful about this technique is that it goes beyond looking for bugs and running static checks and proves the absence of run-time and other errors in your code."

The PolySpace tools employ abstract interpretation to build the structure of the code and then look for places where things can go wrong, say, a pointer that references nonexistent memory or an uninitialized variable. "These are things that can 'break' when you run code," noted Barnard. "These types of errors prove difficult to catch in simulations or unit tests of the code because you would have to build an enormous number of test cases to check all possible paths through the code and all possible values for each variable."

A formal method, however, can mathematically prove that a variable will always be initialized or that for the operation X/Y, the Y value will never be zero. "The tools look at the code in a holistic way to determine the mathematical properties of the variable Y and to prove or disprove that it will always evaluate to a nonzero value," said Barnard. "We use a similar process to check other critical run-time errors such as the occurrence of overflow, underflow, etc."

The PolySpace tools also can analyze code automatically generated by other software products such as Simulink in combination with the MathWorks Real-Time Workshop. You build a Simulink model, generate the C code and analyze it with PolySpace. "Then you can examine problems and then directly link back into the Simulink model to identify the particular spot in the design that might have caused a problem," explained Barnard. "You would fix the model and regenerate the code rather than attempt to fix problems right in the generated C code."

For further reading

Jones, Nigel, "Introduction to MISRA C," Embedded Systems Design, July 2002. www.embedded.com [1].

Selwood, Dick, "Taming C?" Embedded Technology Journal, October 7, 2008.

--"Tutorial on LINT," Center for Parallel Computers, KTH. www.pdc.kth.se/training/Tutor/Basics/lint/index-frame.html [2].

The MISRA standards are available from: www.misra-c.com/ [3]

Source URL (retrieved on 11/27/2014 - 3:17pm):

<http://www.ecnmag.com/articles/2008/11/embedded-systems-c-and-c-tools-reduce-code-errors>

Links:

[1] <http://www.embedded.com/>

[2] <http://www.pdc.kth.se/training/Tutor/Basics/lint/index-frame.html>

[3] <http://www.misra-c.com/>

