The Embedded Muse 52

Editor: Jack Ganssle (jack@ganssle.com) October 4, 2000

MISRA Review

Frequent contributors to the comp.arch.embedded newsgroup sometimes refer to the MISRA (Motor Industry Software Reliability Association) publication "Guidelines For the Use of The C Language in Vehicle Based Software". As one interested in the firmware reliability (is that an oxymoron?) I wanted to check out this publication, but was frustrated by its unavailability on the net. So I ordered a copy from England (35 pounds for overseas shipments) through the web site (http://www.misra.org.uk).

In just a few weeks the 70 page bound booklet arrived. It's emphatically NOT a software standard; rather, the authors define safe ways to use some C constructs and identify others that must be avoided. Use these guidelines in concert with a real standard, one that defines coding styles, commenting conventions, and the like (you're welcome to download the one I use from http://www.ganssle.com/misc/fsm.doc).

While C is indeed a very powerful language, it should come with a warning label: "danger: experts only". It's so easy to create programs that leak memory, run pointers wildly all over memory, or create other difficult-to-find havoc.

The MISRA standard, a collection of 127 coding rules, tries to prevent problems by limiting the types of C constructs we use, and defining safe ways to use others.

Quite a few of the MISRA rules make tremendous sense: don't redefine reserved words and standard library function names. Document and explain all uses of #pragma. When a function may return an error, always test for that error. Functions should have a single exit point.

Some are interesting: never use recursion. Keep pointer indirection to no more than two levels.

A couple are hard but possibly quite valuable: check every value passed to every library routine. Avoid many common library functions.

Other are trivial: only use characters defined by the ISO C standard. Don't nest comments. Write code conforming to ANSI C. Don't confuse logical and bitwise operators. Don't have unreachable code.

Some of the requirements I find disturbing. For instance, rule 118 prohibits the use of dynamic memory allocation. Not a bad idea, due to problems associated with fragmentation. But there are alternatives to malloc/free that still give us the benefits of dynamic memory allocation without the pitfalls. More problematic, this rule tells us not to use library functions which employ dynamic memory, specifically mentioning string.h. This seems awfully restrictive to me… I sure don't want to write my own string handlers… and further, how is one to identify the suspect libraries?

Rule 122 prohibits the use of setjmp and longjmp. These are worse than gotos, of course, in that they let us branch to specific memory addresses. Yet in a few cases longjmp is almost unavoidable.

I think there's much value to the document, but as a stand-alone set of rules it's incomplete. Better, incorporate the rules into your in-house software standard. It's just too hard to conform to two sets of rules living in two different documents.

If MISRA published the rules on-line, they'd be more accessible to the embedded community, hopefully improving the quality of code everywhere. Without such an electronic copy, I doubt if many will ever incorporate these rules into their own standards.

Thought for the Week

In the 80s:
Recruiter: "Tell me the meanings of all the extensions to file names"
Candidate: "VMS or MS-DOS?"
Recruiter: "You got the job!"

In the 90s:
Recruiter: "Tell me all the options used by /bin/ls"
Candidate: "BSD options or System V?"
Recruiter: "You got the job!"

In the 00s:
Candidate: "Tell me all the options"
Recruiter: "20000, 1 year vesting, $1 strike price, IPO next month"
Candidate: "I'll take the job!"

About The Embedded Muse

The Embedded Muse is an occasional newsletter sent via email by Jack Ganssle. Send complaints, comments, and contributions to him at jack@ganssle.com.

To subscribe, send a message to majordomo@ganssle.com, with the words "subscribe embedded *your-email-address*" in the body. To unsubscribe, change the message to "unsubscribe embedded *your-email-address*".

The Embedded Muse is supported by The Ganssle Group, whose mission is to help embedded folks get better products to market faster. We offer seminars at your site offering hard-hitting ideas - and action - you can take now to ***improve firmware quality and decrease development time***. Contact us at info@ganssle.com for more information.