

Re: Spirit rover OS problems (a reliable language)

Source: <http://coding.derkeiler.com/Archive/General/comp.arch.embedded/2004-02/1182.html>

From: Scott Moore (samiam_at_moorecad.com)

Date: 02/13/04

Date: Fri, 13 Feb 2004 09:09:21 GMT

"Dave Hansen" <iddw@hotmail.com> wrote in message news:402b8f85.86573082@News.CIS.DFN.DE...

> *On Thu, 12 Feb 2004 07:51:29 GMT, "Scott Moore" <samiam@moorecad.com>*

> *wrote:*

>

>>

> > *"Andy Sinclair" <me@privacy.net> wrote in message*
> *news:nc9h20h6qi5qe44gqah6hs3e2qnrkd65b@4ax.com...*

> >> *Scott Moore wrote:*

> *[...]*

> >> > *void x(void)*

> >> >

> >> > {

> >> >

> >> > *int a;*

> >> > *int *b;*

> >> >

> >> > *b = &a;*

> >> > *b *= 10;*

> >> >

> *[...]*

> >> *That doesn't work. If you change the last line to `b+=10` it will*

> >> *compile, but a good lint tool will not let it pass.*

>>

> > *Lint, which nowadays is built into the better compilers (gcc, etc),*

>

> *Lint is *not* built into gcc. Gcc is *far* better at reporting*

> *problems than any C compiler I used 20 years ago, but good lints are*

> *also much better at `_their_job`.*

>

> > *is a band-aid. Certainly you can catch some problems, but others not.*

>

> > *It's more like water when you're running a marathon: necessary but not*

> *sufficient.*

>

> > *For example, the routine:*

>>

> > *void a(int *b)*

```
> >
> >{
> >
> > b *= 100;
> > *b = 5;
>
> No need to make it so complicated (and make the same mistake you made
> earlier: any compiler that doesn't detect a problem in that code is
> *broken*).
>
> Here's an actual example:
>
> --- begin included file ---
> C:\Dave>type vlint.c
> void set(int *b)
> {
> b[100] = 5;
> }
>
> int array[100];
>
> void test()
> {
> set(array);
> }
>
> C:\Dave>lint-nt -u -passes(2) vlint.c
> PC-lint for C/C++ (NT) Ver. 8.00n, Copyright Gimpel Software 1985-2003
>
> --- Module: vlint.c
>
> /// Start of Pass 2 ///
>
> --- Module: vlint.c
>
> During Specific Walk:
> File vlint.c line 10: set([100])
> vlint.c 3 Warning 415: Likely access of out-of-bounds pointer (1
> beyond end
> of data) by operator '[' [Reference: file vlint.c: lines 3, 10]
> vlint.c 3 Info 831: Reference cited in prior message
> vlint.c 10 Info 831: Reference cited in prior message
>
> C:\Dave>
>
> --- end included file ---
>
> >
> > Cannot be caught because the compiler has no idea where b came from
> > or what it points to. There is no underlying model of safety for
> > lint to use.
```

- >
- > *Impressive, eh? PC-lint does try to find out where b comes from, and*
- > *is usually quite successful.*

No, it cannot find this out all the time. It is not a %100 solution.
Type safe languages are a %100 percent solution.

- >
- > *Note: the "-u" option disables warnings like "test defined but not*
- > *called," i.e., it does a "unit" lint rather than a "whole program"*
- > *lint. The "-passes(2)" option tells lint to make a second pass to*
- > *specifically look for errors like this. The "831" info messages are*
- > *references for editors that automatically parse error messages and*
- > *locate the cursor on the specified line.*
- >
- >>
- >> *Lets fast forward. There are a lot of tools that do "super lint",*
- >> *and even try to simulate limited runs of the C code to determine*
- >> *if problem areas exist. I unfortunately got the job of administering*
- >> *such a tool set for a large company. I say unfortunately, because*
- >> *I got the assignment due to the fact it was a sh*t job that nobody*
- >> *wanted, and all anyone remembered about me was that I seemed to*
- >> *complain about code quality a lot.*
- >>
- >> *What occurs in real life is that you get massive amounts of warnings.*
- >
- > *Lint early. Lint often. Lint is your *friend*.*

Again, all this accomplishes is loads of (possibly irrelevant) warnings that if you correct, you might have better code. No matter how good it is, it cannot be %100 complete, because the information to do that does not exist in the C language. Other languages do not have that restriction, and don't need to issue warnings for everything, and don't need to use words such as "likely" and "possible" in their output. If you have misused an array reference, that is known, it's an error, not a warning. If you have misused a pointer, that is an error, not a warning.

- >
- > *One (complete, buggy, never-before-linted) project I linted generated*
- > *an error file three times the size of the source code.*
- >
- >> *If the code had not ever been compiled with something like*
- >> *-Wall (gcc for "warn all on"), this would be a massive clean up*
- >> *project which most companies would balk at. This company had been*
- >> *running -Wall tho, and the warnings were considered manageable.*
- >> *However, there was a multimonth cleanup project that would have to*
- >> *occur. To give you an idea of it, the tools were complaining that*
- >> *"printf" returned a value that was ignored. When was the last time*
- >> *you saw the return value of printf used for anything ?*
- >

- > *PC-lint has a very flexible means of disabling unnecessary warnings.*
- > *In the case of this specific example, the default configuration for*
- > *all the supported compilers includes "-esym(534,printf)" which says*
- > *"don't report error 534 for the symbol printf."*
- >

I don't doubt that your lint program is very impressive to you.
To me, its a very complex program that is completely unnecessary
in other, type safe, languages.

- >>
- >>*Anyways, the conversion project was in fact completed. THings like*
- >>*"=" used inside expressions were banned. Although everyone knew there*
- >>
- >>*Flatly banning things replaces thinking with rules. Rules alone will*
- >>*never solve all your problems. There is no silver bullet.*
- >>
- >>*It's better to allow such things, but require them to be justified.*
- >>
- >>*was no way to catch every wrong pointer reference in C, the theory was*
- >>*that making the code as clean as possible would make it as stable*
- >>*as possible.*
- >>
- >>*What you obtain with C in conjunction with such testing is C whose*
- >>*usage is fairly restricted. The toolset, I think properly, was*
- >>*complaining about initalizations that were not obvious, for instance,*
- >>*if the initalization was done in a remote function, or had a condition*
- >>*attached to it, it was flagged, even if the programmer knew that*
- >>*it would get initalized despite the conditional.*
- >>
- >>*Again, with PC-lint it's quite easy to suppress a specific message*
- >>*with a comment in the code. A code review can verify the*
- >>*justification in the maintenance phase.*
- >>
- >>
- >>*So you get C with a truckload of restrictions, and you still have*
- >>*the possibility of wild pointers. Any computer scientist worth his*
- >>*salt will tell you that C is not going to be %100 verifiable no matter*
- >>*what you do.*
- >>
- >>*True. But even languages to which formal proofs can be applied are*
- >>*vulnerable to bugs in specification. TANSTAAFL.*
- >>
- >>
- >>*If you live with this, you will have better quality certainly. But*
- >>*my question is, if you live with all those restrictions, and give*
- >>*up the wild west style of C,*
- >>
- >>*Why again is C a better idea than a language that had type safety*
- >>*built in from day one ?*
- >>

comp.arch.embedded: Re: Spirit rover OS problems (a reliable language)

- > *C never competed with Pascal or Ada or any other HLL (with or without*
- > *type safety). C was competing (and in many cases is still competing)*
- > *with assembly. It's popular because of:*
- >
- > *1) Target platform support. The only microprocessor/microcontrollers*
- > *I can name that don't have a supporting C compiler are 4-biters.*

Every pasture has sh*t laying on it. That does not make same a good product.

- >
- > *2) Serendipity. C was becoming popular just about the time embedded*
- > *microcontrollers were getting large enough to support HLLs and*
- > *compiler technology was getting to the point where the generated code*
- > *was nearly as good as hand-crafted assembly.*

This is the same as point (1), ie., "there seem to be more C compilers in the world". There is no particular efficiency advantage to C.

- >
- > *3) Perceived efficiency. C is a lower-level HLL. Especially for*
- > *embedded work, the bitwise operators were a great boon.*

"perceived". Well put. The efficiency advantage of C lies between the programmers ears.

- >
- > *4) Inertia. Most embedded programmers know C. Most embedded jobs*
- > *involve programming in C. Each feeds of the other. Embedded*
- > *programmers, as a rule, are a conservative bunch. They stick to what*
- > *works. They are suspicious of abstraction.*

Fortran had this advantage as well. Is fortran popular now ?

- >
- > *Summary: I'm not really arguing with you here. My points are 1) Lint*
- > *is better than you think,*

Ive done more work with "super lints" than anyone here, and most likely you as well. Dumping lots of "possibles" and "likelys" on the programmer is not productive programming, it is programming by paranoia.

If the compiler is worth a damm, it can check absolutely if the program is right or wrong. C compilers cannot perform this act because the language does not let the compiler do it.

Now notice the subject of this thread. A \$800 Million dollar Mars lander was sent across space to be felled by a 2 bit software error. C and "super lint" have no place in the scope of \$800 Million of our tax dollars.

comp.arch.embedded: Re: Spirit rover OS problems (a reliable language)

- > 2) without C (or something like it) we (this
- > *is* comp.arch.embedded) would still be mired in assembly. Which is
- > wilder and wester than C. ;-)

Not correct. The language used for most embedded is pure happenstance. I don't use C in my embedded work unless the customer requests it. It delivers a less reliable product for more work, and it is not the appropriate language for any serious work, IMHO.

It CERTAINLY is not appropriate for life support applications or serious money work. In fact, it is probably a worse choice than assembly language, which is at least transparent. C removes transparency from the code without removing assembly's bug potential, a disastrous combination.

- >
- > *Regards,*
- >
- > --*Dave*
- > --
- > *Change is inevitable, progress is not.*