

Re: Increasing efficiency in C

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-03/0804.html

From: Mike Wahler (*mkwahler_at_mkwahler.net*)

Date: 03/05/04

Date: Fri, 05 Mar 2004 20:22:34 GMT

"jacob navia" <jacob@jacob.remcomp.fr> wrote in message
news:c2ajah\$8a\$1@news-reader3.wanadoo.fr...

>

> *"Mike Wahler" <mkwahler@mkwahler.net> a écrit dans le message de*

> *news:9G32c.22867\$aT1.20587@newsread1.news.pas.earthlink.net...*

>>

>

>> *Back [almost?] to topic: If I couldn't control my memory handling*

>> *and pointers, I wouldn't use C.*

>

> *You keep control of everything.*

That's right. That's why I like C. But with such control comes responsibility. I if lose control, it's my own fault.

> *But the system should*

> *have an automatic.*

C doesn't require support of a 'system', and in the presence of one, makes very few requirements of it. Some hardware and some operating systems do have 'safeties' built in. Some do not. C is designed, intentionally, to be able to be used with either. It does allow one to create 'software safeties' if desired, but does not impose such a burden in cases where it's not needed or desired. Such things belong in e.g. special-purpose libraries, not the language or its 'standard' (i.e. platform-neutral) library. C is a 'lightweight', *flexible* language. I like it that way.

I like Dan's 'sharp tool' metaphor.

If a soldier in combat shoots his foot, should we tell him to stop carrying his gun loaded, or train him to carry and handle it safely? If he cannot follow

such training, he does not belong on the battlefield.

- > *When you turn the wheel there is a point where the*
- > *machine refuses to turn more. It just refuses to*
- > *obey to a nonsense command. It is build like that.*

And the 'response' can vary widely amongst machines. 'Recovery' might or might not be possible. Careful coding is the way to prevent such responses.

- > *When you attempt to write beyond a pointer limits*
- > *the system doesn't do it.*

One cannot always assume existence of a 'system' (other than the program itself). If by 'system' you mean only e.g. a hardware platform, of course its capacity and possible behaviors must be considered (could be addressed with coding practice, or perhaps simply warnings in documentation).

- > *You still have an error, and you can fire an exception*
- > *or just return false, and leave it up to the calling*
- > *routine.*

Or you might have no recourse at all. The machine could 'freeze up'. The only solution is careful, thoughtful coding (or dependence upon such from others, via e.g. a library). Of course platform-specific issues are important, but they're not part of the standardized, 'general purpose' language, C. The design of C allows the creation of such platform-specific controls and checks, anyway.

- > *You keep control of the error since you are the*
- > *only one that can fix it. The machine will not fix*
- > *it.*

Depends upon the machine, but yes, imo one should code carefully and thoughtfully, making the application as 'self-responsible' as possible.

- >
- > *My first programming was with*
- > *'plug boards', I'm not sure if it was 'safe' or 'dangerous'.*
- > *I never got an electric shock, and didn't destroy any hardware,*
- > *anyway. :-)*
- >
- > *You should keep an eye into nostalgia.*
- >
- > *Yes, those were the times.*
- >
- > *I am too old to look back. I want to look forward.*

I look both directions. Prior knowledge and experience are as valuable resources as new knowledge. The latter often depends upon the former, as well.

> *If we program remembering the past we are doomed.*

So I should discard e.g. elementary logic, because I learned it a long time ago?

> *We have no future.*

>

> *Remember the future.*

Now that made me smile. I can **consider** the future, I find it difficult to **remember** it. :-)

> *Today's fast machines provide*

> *power for a new and widespread way of developing,*

> *in scales much bigger than before.*

But, (and I think this is essentially the crux of my objections to your ideas about 'safetifying' C strings):

Trying to "bulletproof" everything imo encourages mindlessness. **Not Good** for a programmer. This should be certainly implemented as much as possible at the end-user level, by programmers who **think**. Even the C++ 'std::string' (as well as the language itself) that I mentioned can be abused. It **is** imo 'safer' but not bulletproof, nor can it ever be.

It's this recent trend (that I see) in languages to obviate the need to think, that I find very disquieting.

> *Today's priorities aren't those of years ago.*

Some are, some are not. Priorities also depend upon an application's domain.

> *A good antidote against nostalgia is realizing that those circuit boards are gone for good.*

I was not promoting nostalgia, nor 'plug-board' programming today, only giving an example.

And, no 'circuit boards', imo will never go away. They're often implemented in silicon nowadays, and programmed at higher levels, that's all.

> *There are much more powerful circuit boards today.*

comp.lang.c: Re: Increasing efficiency in C

Yup. But the concept of a 'circuit' and its usefulness imo will not be going away any time soon. I don't do 'plug board' programming any more, but while learning it, I did learn about 'circuits', and find such learning still very useful today. Hey! Who turned out the lights!?! :-)

Anyway, if you see a need for your 'safer' strings, by all means make a library for such. But don't impose it upon the basic language and library.

-Mike