

Re: Increasing efficiency in C

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-03/0667.html

From: jacob navia (jacob_at_jacob.remcomp.fr)

Date: 03/05/04

Date: Fri, 5 Mar 2004 00:02:56 +0100

"Dan Pop" <Dan.Pop@cern.ch> a écrit dans le message de news:c27gec\$4n5\$1@sunnews.cern.ch...
> In <[c27e5q\\$c72\\$1@news-reader1.wanadoo.fr](mailto:c27e5q$c72$1@news-reader1.wanadoo.fr)> "jacob navia" <jacob@jacob.remcomp.fr> writes:
>
>
> > "Dan Pop" <Dan.Pop@cern.ch> a écrit dans le message de > >news:c27a00\$7hh\$4@sunnews.cern.ch...
> >> In <[c25kuu\\$rae\\$1@news-reader4.wanadoo.fr](mailto:c25kuurae1@news-reader4.wanadoo.fr)> "jacob navia" > ><jacob@jacob.remcomp.fr> writes:
> >>I wanted to emphasize "unbounded" because there is no way to know if > >the zero is not there where the pointer will end pointing to...
>
> You don't know where the pointer will end pointing to. Your wording > simply didn't make any sense to anyone but you.
>
> The representation of a string in C is the sequence of characters, up to > and including the null terminator. No kind of pointer is involved in the > representation of a C string.
>

Wow Dan, this is news for me. No kind of pointer?

Not even a char * as it seems?

Strange. Are all those prototypes in string.h wrong?

I would fill a defect report.

Just do not exaggerate Dan. Let's keep cool ok?

I am speaking about a naked char * that points to the start of a sequence of bytes that should end with a terminating zero.

By definition of the data structure, its length is not known, and the same scan must be repeated

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

> *defined goals, rather than as universal replacements for the C strings.*
>

Safety was one of the more widespread goals. I am trying to build checked strings into lcc-win32. I think that a more debuggable environment is easier to work with.

> *And the very existence of these libraries proves that the C language DOES support alternate schemes. So, your point is moot.*
>

The language doesn't support it.

I repeat that length prefixed strings should be easy to use: name[2] should do what is supposed to.

My whole point is that data structure development should be opened up to the C user that should be able to specify data types that follow special rules he/she defines.

For instance you could add a "flags" field to the standard length prefixed string, and implement read only strings, or time stamp based data, or whatever.

The language should allow people defining programs that handle the data structures in a way it suits them the best.

C is not object oriented but we all use lists, stacks, hash tables in our everyday programming.

> >> *Since C programmers aren't the last people to care about efficiency, what conclusion can you draw?*
> >
> > *Since language support doesn't encourage the use of bounded pointers*
> > *C string handling is much more error prone than it should be.*
>
> *1. This is not a performance issue.*

No, this is a human performance issue. People get bored of details. Computers do not.

People use computers to make repetitive work. Why can't we use the computer to check for mistakes?

Your answer is:

>
> *2. This is a *general* problem of C: most C features are error prone in the hands of the incompetent.*
>

Your are competent Dan. Surely more than me.

Re: Increasing efficiency in C

I belong to the other ones.

The ones that make mistakes. I am not afraid of saying this, maybe because I think knowing this is the start of knowledge.

When you realize your mistakes you can start learning.
Only then.

> >Never had the traps because of the missing zero?
>
> Nope.
>

:-)

Of course not Dan. Sure. I believe you that 100%

> >The failure modes of the string functions in the library like strcpy
> >are just horrible. Memory corruption is guaranteed unless a zero
> >is found...
>
> Dynamic memory allocation has exactly the same problems: write beyond
> a dynamically allocated memory bolck (in either direction) and memory
> corruption will (most likely) bite you, sooner or later. What is your
> better replacement for malloc and friends?
>

The garbage collector. I wrote one for my Lisp interpreter in the 90ties, and I have adapted Mt Boehm's work to lcc-win32.

The GC is much better than malloc/free. But I know, that's another discussion ...

> C is a sharp tool *by design*. People who can't use sharp tools or are
> afraid of them, should not use C. There are plenty of other programming
> languages designed for them so there is no need to turn C into a less
> sharp tool (and, therefore less effective in the hands of the competent
> programmers) and annoy C's *intended* user base.
>

I want it to be sharper Dan. C is not sharp enough with all those bugs that creep the programs. You can't be sure of a tool if it is not designed to be sharp and safe.

You take the knife not at the edge?

A knife is a sharp tool by its very nature.

But it can only be used because you do not touch at the edge isn't it?

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

That blunt side, that provides safety for your hand makes for a usable knife. Without it, using a knife is cutting yourself in the fingers :-)

- > *There are many ways in which C needs to be extended, but adding more*
- > *string formats is not one of them. You're wasting your time trying to*
- > *fix something that isn't broken.*
- >

That is the start. A better string library would be an achievement.

Nothing spectacular, and very simple.