

Re: Is C99 the final C? (some suggestions)

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2003-12/0459.html

From: Sidney Cadot (sidney_at_jigsaw.nl)

Date: 12/02/03

Date: Tue, 02 Dec 2003 22:58:07 +0100

Paul Hsieh wrote:

> *Sidney Cadot <sidney@jigsaw.nl> wrote:*

>>*I think C99 has come a long way to fix the most obvious problems in
>>C89 (and its predecessors).*

> *It has? I can't think of a single feature in C99 that would come as
> anything relevant in any code I have ever written or will ever write
> in the C with the exception of "restrict" and //-style comments.*

For programming style, I think loop-scoped variable declarations are a big win. Then there is variable sized array, and complex numbers... I'd really use all this (and more) quite extensively in day-to-day work.

>>*[...] I for one would be happy if more compilers would
>>fully start to support C99, It will be a good day when I can actually
>>start to use many of the new features without having to worry about
>>portability too much, as is the current situation.*

> *I don't think that day will ever come. In its totality C99 is almost
> completely worthless in real world environments. Vendors will be
> smart to pick up restrict and few of the goodies in C99 and just stop
> there.*

Want to take a bet...?

>>** support for a "packed" attribute to structs, guaranteeing that no
>> padding occurs.*

>

> *Indeed, this is something I use on the x86 all the time. The problem
> is that on platforms like UltraSparc or Alpha, this will either
> inevitably lead to BUS errors, or extremely slow performing code.*

Preventing the former is the compiler's job; as for the latter, the alternative is to do struct unpacking/unpacking by hand. Did that, and didn't like it for one bit. And of course it's slow, but I need the semantics.

comp.lang.c: Re: Is C99 the final C? (some suggestions)

- > *If instead, the preprocessor were a lot more functional, then you*
- > *could simply extract packed offsets from a list of declarations and*
- > *literally plug them in as offsets into a char[] and do the slow memcpy*
- > *operations yourself.*

This would violate the division between preprocessor and compiler too much (the preprocessor would have to understand quite a lot of C semantics).

- >> ** upgraded status of enum types (they are currently quite*
- >> *interchangeable with ints); deprecation of implicit casts from*
- >> *int to enum (perhaps supported by a mandatory compiler warning).*
- >
- >
- > *I agree. Enums, as far as I can tell, are almost useless from a*
- > *compiler assisted code integrity point of view because of the*
- > *automatic coercion between ints and enums. Its almost not worth the*
- > *bothering to ever using an enum for any reason because of it.*

Yes.

- >> ** a clear statement concerning the minimal level of active function*
- >> *calls invocations that an implementation needs to support.*
- >> *Currently, recursive programs will stackfault at a certain point,*
- >> *and this situation is not handled satisfactorily in the standard*
- >> *(it is not adressed at all, that is), as far as I can tell.*
- > *That doesn't seem possible. The amount of "stack" that an*
- > *implementation might use for a given function is clearly not easy to*
- > *define. Better to just leave this loose.*

It's not easy to define, that's for sure. But to call into recollection a post from six weeks ago:

```
#include <stdio.h>

/* returns n! modulo 2^(number of bits in an unsigned long) */
unsigned long f(unsigned long n)
{
    return (n==0) ? 1 : f(n-1)*n;
}

int main(void)
{
    unsigned long z;
    for(z=1;z!=0;z*=2)
    {
        printf("%lu %lu\n", z, f(z));
        fflush(stdout);
    }
    return 0;
}
```

comp.lang.c: Re: Is C99 the final C? (some suggestions)

...This is legal C (as per the Standard), but it overflows the stack on any implementation (which is usually a symptom of UB). Why is there no statement in the standard that even so much as hints at this?

>> ** a library function that allows the retrieval of the size of a memory block previously allocated using "malloc"/"calloc"/"realloc" and friends.*
>
> *There's a lot more that you can do as well. Such as a tryexpand() function which works like realloc except that it performs no action except returning with some sort of error status if the block cannot be resized without moving its base pointer. Further, one would like to be able to manage *multiple* heaps, and have a freeall() function -- it would make the problem of memory leaks much more manageable for many applications. It would almost make some cases enormously faster.*

But this is perhaps territory that the Standard should steer clear of, more like something a well-written and dedicated third-party library could provide.

>> ** a #define'd constant in stdio.h that gives the maximal number of characters that a "%p" format specifier can emit. Likewise, for other format specifiers such as "%d" and the like.*
>>

>> ** a printf format specifier for printing numbers in base-2.*

> *Ah -- the kludge request.*

I'd rather see this as filling in a gaping hole.

> *Rather than adding format specifiers one at a time, why not instead add in a way of being able to plug in programmer-defined format specifiers?*

Because that's difficult to get right (unlike a proposed binary output form).

> *I think people in general would like to use printf for printing out more than just the base types in a collection of just a few formats defined at the whims of some 70s UNIX hackers. Why not be able to print out your data structures, or relevant parts of them as you see fit?*

The %x format specifier mechanism is perhaps not a good way to do this, if only because it would only allow something like 15 extra output formats.

>> ** I think I would like to see a real string-type as a first-class citizen in C, implemented as a native type. But this would open up too big a can of worms, I am afraid, and a good case can be made that this violates the principles of C too much (being a low-level language and all).*

Re: Is C99 the final C? (some suggestions)

comp.lang.c: Re: Is C99 the final C? (some suggestions)

- >
- > *The problem is that real string handling requires memory handling.*
- > *The other primitive types in C are flat structures that are fixed*
- > *width. You either need something like C++'s constructor/destructor*
- > *semantics or automatic garbage collection otherwise you're going to*
- > *have some trouble with memory leaking.*

A very simple reference-counting implementation would suffice. But yes, it would not rhyme well with the rest of C.

- > *With the restrictions of the C language, I think you are going to find*
- > *it hard to have even a language implemented primitive that takes you*
- > *anywhere beyond what I've done with the better string library, for*
- > *example (<http://bstring.sf.net>). But even with bstrlib, you need to*
- > *explicitly call bdestroy to clean up your bstrings.*
- >
- > *I'd be all for adding bstrlib to the C standard, but I'm not sure its*
- > *necessary. Its totally portable and freely downloadable, without much*
- > *prospect for compiler implementors to improve upon it with any native*
- > *implementations, so it might just not matter.*

>> ** Normative statements on the upper-bound worst-case asymptotic*
>> *behavior of things like qsort() and bsearch() would be nice.*

- >
- > *Yeah, it would be nice to catch up to where the C++ people have gone*
- > *some years ago.*

I don't think it is a silly idea to have some consideration for worst-case performance in the standard, especially for algorithmic functions (of which qsort and bsearch are the most prominent examples).

- >> *O(n*log(n)) for number-of-comparisons would be fine for qsort,*
- >> *although I believe that would actually preclude a qsort()*
- >> *implementation by means of the quicksort algorithm :-)*
- > *Anything that precludes the implementation of an actual quicksort*
- > *algorithm is a good thing. Saying Quicksort is O(n*log(n)) most of*
- > *the time is like saying Michael Jackson does not molest most of the*
- > *children in the US.*

>> ** a "reverse comma" type expression, for example denoted by*
>> *a reverse apostrophe, where the leftmost value is the value*
>> *of the entire expression, but the right-hand side is also*
>> *guaranteed to be executed.*

- >
- > *This seems too esoteric.*

Why is it any more esoteric than having a comma operator?

>> ** triple-`&&` and triple-`||` operators: `&&&` and `|||` with semantics*
>> *like the 'and' and 'or' operators in python:*
>>

comp.lang.c: Re: Is C99 the final C? (some suggestions)

>> *a &&& b ---> if (a) then b else a*
>> *a ||| b ---> if (a) then a else b*
>>
>> *(I think this is brilliant, and actually useful sometimes).*
>
> *Hmmm ... why not instead have ordinary operator overloading?*

I'll provide three reasons.

- 1) because it is something completely different
- 2) because it is quite unrelated (I don't get the 'instead')
- 3) because operator overloading is mostly a bad idea, IMHO

> *While*
> *this is sometimes a useful shorthand, I am sure that different*
> *applications have different list cutesy compactations that would be*
> *worth while instead of the one above.*

... I'd like to see them. &&& is a bit silly (it's fully equivalent to "a ? b : 0") but ||| (or ?: in gcc) is actually quite useful.

>> ** a way to "bitwise invert" a variable without actually*
>> *assigning, complementing "&=", "|=", and friends.*
>
> *Is a ~= a really that much of a burden to type?*

It's more a strain on the brain to me, why there are coupled assignment/operators for neigh all binary operators, but not for this unary one.

>> ** 'min' and 'max' operators (following gcc: ?< and ?>)*
>
> *As I mentioned above, you might as well have operator overloading instead.*

Now I would ask you: which existing operator would you like to overload for, say, integers, to mean "min" and "max" ?

>> ** a div and matching mod operator that round to -infinity,*
>> *to complement the current less useful semantics of rounding*
>> *towards zero.*

> *Well ... but this is the very least of the kinds of arithmetic operator*
> *extensions that one would want. A widening multiply operation is*
> *almost *imperative*. It always floors me that other languages are not*
> *picking this up. Nearly every modern microprocessor in existence has*
> *a widening multiply operation -- because the CPU manufacturer *KNOW**
> *its necessary. And yet its not accessible from any language.*

...It already is available in C, given a good-enough compiler. Look at the code gcc spits out when you do:

comp.lang.c: Re: Is C99 the final C? (some suggestions)

```
unsigned long a = rand();  
unsigned long b = rand();
```

```
unsigned long long c = (unsigned long long)a * b;
```

- > *Probably because most languages have been written on top of C or C++.*
- > *And what about a simple carry capturing addition?*

Many languages exist where this is possible, they are called "assembly". There is no way that you could come up with a well-defined semantics for this.

Did you know that a PowerPC processor doesn't have a shift-right where you can capture the carry bit in one instruction? Silly but no less true.

>>*Personally, I don't think it would be a good idea to have templates in C, not even simple ones. This is bound to have quite complicated semantics that I would not like to internalize.*

- > *Right -- this would just be making C into C++. Why not instead dramatically improve the functionality of the preprocessor so that the macro-like cobblings we put together in place of templates are actually good for something? I've posted elsewhere about this, so I won't go into details.*

This would intertwine the preprocessor and the compiler; the preprocessor would have to understand a great deal more about C semantics than it currently does (almost nothing).

Best regards,

Sidney