

## Re: Simple C containers, std::vector analog

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2005-03/0763.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2005-03/0763.html)

---

*From:* Ross A. Finlayson ([raf\\_at\\_tiki-lounge.com](mailto:raf_at_tiki-lounge.com))

*Date:* 03/07/05

Date: 7 Mar 2005 08:48:44 -0800

Stop intimidating me!

Yeah, there are bugs in that. The macro has a block which declares its own variable, and there is a missing closing brace on the block. All those ##'s are not needed.

Today the compiler will readily tell you exactly where the bug is, and is that close to correcting it, steps toward automatic computing or rather automatic programming. Thirty years ago, the compiler will readily give you the object file to execute it on the thirty million dollar supercomputing mainframe. Today, the computing power of thirty years ago's thirty million dollar supercomputing mainframe is fit into an MP3 player that plays MP3 files into headphones and straps to your arm. Thirty years ago it was 1975 and computers ran in Kilohertz (kilohertz, kiloHertz, KiloHertz). Today they run in Gigahertz, as Moore approaches physical silicon limits, leading to reversible processors that run cold on a trickle of picoamps. Having no truck with school, the computer still runs a million times faster.

The realloc function doesn't always deallocate the previous block of memory and copy the contents, er... sometimes there is still room on the heap contiguous to the originally allocated block. It still returns NULL on failure. It's at the implementation's discretion how to do that. I'm concerned that some C implementations don't support nested blocks.

It's probably safer to use `malloc(sizeof(word_t))` in the initializer than `malloc(0)`, or rather:

```
#define vector_init(vector, T) vector.data =  
s_cast(T)(malloc(sizeof(T))
```

and some implementations do choke on realloc'ing a null pointer.

I think RAI is a decent concept, a good concept, but like everything else, good in moderation.

(Some things are good in lots of moderation.)

## comp.lang.c: Re: Simple C containers, std::vector analog

That C vector analog is kind of a castrated C++ standard library container, that's because in this case for its use there is no need for many of the functions except those listed. It can return begin and end pointers for use with std::algorithm, obviously enough implemented with std::vector it does. There's no implementation of push\_front or pop\_front, but it could pop\_front by incrementing the pointer, storing the original pointer for allocation, and stuff. There no insertion into the array, only push\_back, or vector push\_back(vector, value).

Basically the idea here is to design the functions as good ISO C, and ANSI C, yet have them essentially be template functions for use with C++, with old or new style I/O and containers.

```
#ifdef __cplusplus
template f <voidpT> void f(voidpT buffer){
#elif
void f(void* buffer){
#endif
```

For example, they work on buffers, and these vector type dealies, but again I am working on simple software where the full complexity of C++ is underappreciated. The #elif thing there is not universally supported. So, I'm trying to figure out how to work in basically istream and ostream, but only as they look like a file-mapped memory buffer, just readT and writeT, \* and ++. I'm out of my depth here.

Thank you,

Ross F.

--

"It's the smallest infinitesimal, Russell,  
there are smaller infinitesimals."