

# Re: manipulating void\* in array

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2005-07/msg01690.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2005-07/msg01690.html)

---

- *From:* "Stijn van Dongen" <[gnn@xxxxxxxxxx](mailto:gnn@xxxxxxxxxx)>
  - *Date:* 18 Jul 2005 06:55:52 -0700
- 

Dave Thompson wrote:

> On 13 Jul 2005 15:51:04 -0700, "Stijn van Dongen" <[svd@xxxxxxxxxxxxx](mailto:svd@xxxxxxxxxxxxx)>  
> wrote:  
>  
>> A question about void\*. I have a hash library where the hash create  
>> function accepts functions  
>> unsigned (\*hash)(const void \*a)  
>> int (\*cmp) (const void \*a, const void \*b)  
>>  
>> The insert function accepts a void\* key argument, and uses the  
>> functions above to store this argument. It returns something (linked to  
>> the key) that the caller can store a value in. The actual key argument  
>> is always of the same pointer type (as seen in the caller, say foo\*).  
>>  
>> Now I would like to have a function, say hash\_keys, which returns all  
>> keys in the hash as a pointer to a malloced array of void\*. However,  
>> there is not really a type I can use. I see two possible solutions.  
>> The first is to typedef a struct containing a void pointer:  
>>  
>> typedef struct {  
>> void\* pp;  
>> } genpp;  
>>  
>> and have hash\_keys return genpp\* (the size could be written in an int\*  
>> argument  
>> to hash\_keys). This would make accessing the keys cumbersome.  
>>  
>> Solution 2) is illegal C (I think), but it's tempting. hash\_keys would  
>> construct an array of void\*-sized elements and copy its key pointers  
>> there using char\* arithmetic. It would return void\* and the caller  
>> would cast it to foo\*\*. For one thing, this assumes sizeof(void\*) is  
>> sizeof(foo\*) for all possible foo. For another, I see nothing in the  
>> standard on void\* that would support any of this. However, it feels as  
>> if approach 2) is conceptually extremely similar to the first and I  
>> suspect it would work on nearly all platforms/compilers.  
>>  
> An array of void\*, addressed as void\*\*, and an array of struct  
> containing only void\*, addressed as genpp\*, are indeed similar. And in  
> fact are very likely (though not guaranteed) to be laid out the same.

## Re: manipulating void\* in array

I think the array of void\* has to be addressed as void\*; AFAIK there is no type 'void\*\*' in C (as you cannot dereference a void\* pointer).

- > But the individual element pointers, of type void\* in either case, are
- > NOT guaranteed to be laid out the same as some other (data) pointer
- > type foo\*. So treating either of these as (casting to) foo\*\* and
- > trying to use it is not safe, and I know of systems, though only a
- > few, where it won't work. Note that even if different pointers are the
- > same size, their internal representations may be different, so
- > checking or verifying size is not enough to be safe.

ok, I suspected that much. Would still be interested to learn about problematic systems, but it's illegal C and off-topic, so I won't pursue it.

- > I'm not sure what you mean by "char\* arithmetic" — does the "hash"
- > (which sounds like it's actually a hash table) know about (remember?)
- > each thing stored in it, or is it packing (all or some of) them
- > (perhaps a buckets-worth) into a big array? I'm guessing the "hash"
- > knows only about void-pointers to things, and maybe their sizes, but
- > not their types?

The hash table (indeed) insert function gets a void\*, and nothing else — no size. It uses the cmp function given to it (when it was created) to traverse the linked lists attached to a bucket array, and it uses the hash function to compute the offset in that bucket array.

The thing that ties the key and value together is a struct looking like this:

```
typedef struct {  
void* key;  
void* val;  
} KV;
```

On my part, I'd prefer not to enter a discussion on the sanity of this setup. The idea behind the whole setup (most of which is omitted here) is that it provides a general setup for hash routines; the routines provided by this particular hash library often serve as a layer below another one that can implement the type of services and typechecking it chooses to.

- > If so, the logical thing to do is to create and
- > return an array of void\*, and the caller, who presumably knows the
- > type of what was (or should have been) stored, then converts that to a
- > foo\* before using it, either by explicitly casting or implicitly by
- > assigning to its own (copy) pointer.

The reason why I don't want to do that is because it implies I'll have a whole array of \*copies\* of my foo things. Given any one foo

## Re: manipulating void\* in array

(presumably some struct), I never want to duplicate it, and only pass pointers to it around. Having copies around would cause endless trouble with ownership issues. In the current setup returning foo\* is also impossible since the hash routines never (can) dereference their void\* key arguments. Which is a good thing. It could be achieved by passing another callback to the hash table creation routine of course. But I rather have a foo\*\* array or a genpp\* array as described above.

regards,  
Stijn

---

- *Follow-Ups:*

- ◆ [Re: manipulating void\\* in array](#)  
◇ From: Netocrat

- *References:*

- ◆ [manipulating void\\* in array](#)  
◇ From: Stijn van Dongen
- ◆ [Re: manipulating void\\* in array](#)  
◇ From: Dave Thompson

- Prev by Date: [Re: how to get the data type of a variable](#)
- Next by Date: [Re: Link list problem](#)
- Previous by thread: [Re: manipulating void\\* in array](#)
- Next by thread: [Re: manipulating void\\* in array](#)
- Index(es):
  - ◆ [Date](#)
  - ◆ [Thread](#)