Subject: Re: Storing !NULL in struct
Posted by Michael Galloy on Mon, 18 Mar 2013 16:48:54 GMT
View Forum Message <> Reply to Message

On 3/18/13 6:40 AM, Tom Grydeland wrote:
> On Monday, March 18, 2013 12:20:51 PM UTC, alx wrote:
>
>>> Do you also understand why I referred to this as defeating the purpose of high-level data structures?
>
>> No, I do'nt. I do not understand what you mean by "high-level" data structures.
>
> Basically, LISTs and HASHes. These are capable of storing arbitrary values in each value slot, thereby giving you the opportunity to create arbitrarily complex nested structures. The problem is that values stored in these structures cannot easily be accessed and modified, which defeats the purpose of having these structures in the first place.
>
>> As previously said, IDL structures are defined like C structures: each field being defined "by value". If you want more flexibility, you can (and you must) use pointers (each field is then defined "by reference").
>
> This point has been made several times already, and in my reply to Chris Torrence, I granted that the reasons behind this decision is valid, even if it has a consequence that seems silly by itself.
>
> What I was demonstrating in the message you replied to, however, is a different point. When a struct is stored in a LIST or a HASH, you cannot change its values, while a struct referred to by a pointer _can_ be modified.
>
> Consider this simple example (and before you suggest workarounds, please consider that I already said I understand how to work around these issues, _AND_ that I'm deliberately constructing _simple_ examples, so that the points stand out more clearly, but that the cases I intend to deal with are going to be much more involved, and nested much more deeply than what I'm showing you here):
>
> IDL> .run
> - pro modify, s
> - s.foo += 1
> - end
> % Compiled module: MODIFY.
> IDL> s = {foo:0}
> IDL> modify, s
> IDL> help, s
> ** Structure <23d4ed8>, 1 tags, length=4, data length=4, refs=1:
> FOO LONG 1
>
> ;; So a structure stored in a simple variable can be MODIFY'ed
>

> IDL> p = ptr_new({foo:0})
> IDL> modify, *p
> IDL> help, *p
> ** Structure <23d4b08>, 1 tags, length=4, data length=4, refs=1:
> FOO LONG 1
>
> ;; Similarly, a structure stored behind a pointer can be MODIFY'ed
>
> IDL> L = list({foo:0})
> IDL> help, L[0]
> ** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
> FOO LONG 0
> IDL> modify, L[0]
> IDL> help, L[0]
> ** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
> FOO LONG 0
>
> ;; while a structure stored in a LIST _cannot_ be MODIFY'ed, and my example fails _without warning_.
>
> Do you see _now_ what I mean by defeating the purpose?
>
>> alx.
>
> --T
>

I think what you are showing here is that any variable passed by value
does not end up modified at the calling level. It doesn't work for
arrays of structures either:

```
IDL> sarr = replicate({ foo: 0 }, 10)
IDL> modify, sarr[0]
IDL> print, sarr[0]
{       0}
```

s and *p are named variables, but L[0] and sarr[0] are not. s and *p are
passed by reference, L[0] and sarr[0] have pass by value semantics.

My hash suggestion was to replace your structure with a hash:

```
IDL> c = list(hash('t', 0))
IDL> print, (c[0])['t']
       0
IDL> (c[0])['t'] = 1
IDL> print, (c[0])['t']
       1
```

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation
