
Subject: pointer to structures

Posted by [eeeyler](#) on Tue, 04 Apr 2000 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

suppose I wish to create a structure and wish to reference that structure and its contents via a pointer:

```
filter=ptr_new({points:['a','b'],pt1_value:200, pt2_value:'X_WHYLOG'})
```

how do I reference the points array? I thought it would be as:

```
print, *filter.points
```

but I get the message

%Expression must be a structure in this context: Filter

Thank you for your help!

Sent via Deja.com <http://www.deja.com/>

Before you buy.

Subject: Re: pointer to structures

Posted by [davidf](#) on Thu, 06 Apr 2000 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Liam E.Gumley (Liam.Gumley@ssec.wisc.edu) writes:

- > A very reasonable argument. I strive for readability and consistency
- > before flexibility, because it allows me to come back to my source code
- > after a weeks vacation and figure out what the heck I was doing before I
- > left.

One of the reasons I always give my wife for not taking vacations is that not only do I lose a week of income while I'm traveling, but I lose three weeks of income when I get back trying to re-learn everything I used to know about IDL. :-(

Cheers,

David

P.S. Let's just say those poor soul's signed up for the next Advanced IDL Programming course in the UK in June are in for it. That course falls smack in the middle of a tour of German beer gardens and a trip to Wembleton. They don't stand a chance. "Uh, IDL!? Something to do with Cobra, I think." :-(

On the other hand, that might be one of the most entertaining courses I've taught in a long time. :-)

--

David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: <http://www.dfanning.com/>
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: pointer to structures
Posted by [Liam E. Gumley](#) on Thu, 06 Apr 2000 07:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

"J.D. Smith" wrote:
>
> "Liam E.Gumley" wrote:
>>
>> "J.D. Smith" wrote:
>>>
>>> "Liam E.Gumley" wrote:
>>>>
>>>> "J.D. Smith" wrote:
>>>> > With time, you will get used to these semantics. They seem arcane, but
>>>> > eventually it becomes somewhat readable to the experienced eye. Of course, I've
>>>> > struggled with statements like:
>>>> >
>>>> > HEADER=*(*(self.DR)[sel[i]].HEADER)
>>>>
>>>> I neglected to provide an example of why simplified pointer and
>>>> structure referencing is desirable. Thanks for the help JD!
>>>>
>>>> ;-)
>>>>
>>>> Cheers,
>>>> Liam.
>>>
>>> But then you have to ask yourself which is worse, the confusing string above, or
>>> the explicit:
>>>
>>> drs_ptr=self.DR
>>> drs=*drs_ptr
>>> this=drs[sel[i]]
>>> hd_arr_ptr=*this
>>> hd=*hd_arr_ptr
>>>
>>> repeat this about 5000 times throughout your application, and you begin to
>>> appreciate the terse form above. Especially if you're passing some part of the
>>> nested data to a routine by reference... intermediate variables require you to

```

>>> remember to assign them after use (everybody remember
>>> widget_control,stash,set_uvalue=state,/NO_COPY?).
>>
>> I would not repeat this code 5000 times. I'd find a way to encapsulate
>> it in a function where I can include comments and error checking (e.g.
>> Is this a valid pointer? Does it point to a defined variable?). In these
>> cases I find it much better to create a 'put' and 'get' function pair
>> where all the de-referencing is handled inside the function. That way I
>> can use the 'put' and 'get' modules all over the place, and if I change
>> the way the pointers/structures are nested, I only have to change the
>> code in two places (inside the functions).
>
> The problem with this is code inflation. If you want to manipulate parts of
> your data structure in place, you need direct access to a pointer or some other
> by reference value. If you choose to pass pointer values to all intermediate
> routines, you are in a sense compromising the very data structure encapsulation
> you are attempting to achieve. What if later it became a list of pointers?
> With the put/set paradigm, you are limited in the ways helper functions can
> interact with your data structure, and you are forced to wrap each call:
>
> get,My_Var=mv
> do_something,mv
> put,My_Var=mv
>
> reminiscent of the example stash variable I gave. This is not necessarily a bad
> idea. Especially now that we have _REF_EXTRA so that incorporating overloaded
> get/put methods in an object hierarchy is possible. But it yields consistency
> at the price of flexibility. Sometimes this is a good tradeoff, perhaps even
> more times than most people would be inclined to think. In other situations, a
> more carefully designed data structure can give you the procedural flexibility
> you need without compromising future design revisions. There is room for both
> styles of design in your toolchest.

```

A very reasonable argument. I strive for readability and consistency before flexibility, because it allows me to come back to my source code after a weeks vacation and figure out what the heck I was doing before I left.

Cheers,
Liam.
