Subject: Re: pointer to structures
Posted by John-David T. Smith on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"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?).

Maybe we need a lexical parser like cdecl, to check on these for you? A fine
programming project for an aspiring IDL programmer out there.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: pointer to structures
Posted by Liam E. Gumley on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"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.

Subject: Re: pointer to structures
Posted by davidf on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith (jdsmith@astro.cornell.edu) writes:

> Uh oh... forgot to test your code I'm afraid. You have the precendence correct,
> but the solution reversed. You must force the pointer dereference to occur
> *before* structure dereference... see the other posts.
>
> If you had something like this:
>
> filter={points:ptr_new(['a','b']),pt1_value:200, pt2_value:'X_WHYLOG'}
>
> You can use the precendence to your advantage, visa vis:
>
> print,*filter.points
>
> Keeping the precendence in mind can eliminate extraneous parentheses, which, for
> everyone except lisp programmers, often add confusion.

Sorry. I'm getting the house ready to sell and I should
know better than to check the newsgroup while I'm waiting
for the trim to dry. :-(

Cheers,

David

P.S. I'll check in next week. :-(

--
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 Ben Tupper on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

> eeeyler@my-deja.com (eeeyler@my-deja.com) writes:
>
>> 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
>
> The problem here is that a pointer dereference has the lowest
> order of precedence. Lower, even, than a structure dereference.
> So, you must first dereference the structure (by using
> parentheses), and *then* dereference the pointer, like this:
>
>   filter=ptr_new({points:['a','b'],pt1_value:200, pt2_value:'X_WHYLOG'})
>   Print, *(filter.points)
>

David,

Caution! Bikes all over the road ahead!

I have always hoped that I'm not the only one who has driven a perfectly
nice bike into a parked car! Ouch!

Ben

--

Ben Tupper

Bigelow Laboratory for Ocean Science
tupper@seadas.bigelow.org

pemaquidriver@tidewater.net

Subject: Re: pointer to structures
Posted by John-David T. Smith on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> eeeyler@my-deja.com (eeeyler@my-deja.com) writes:
>
>> 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
>
> The problem here is that a pointer dereference has the lowest
> order of precedence. Lower, even, than a structure dereference.
> So, you must first dereference the structure (by using
> parentheses), and *then* dereference the pointer, like this:
>
>   filter=ptr_new({points:['a','b'],pt1_value:200, pt2_value:'X_WHYLOG'})
>   Print, *(filter.points)
>

Uh oh... forgot to test your code I'm afraid. You have the precendence correct,
but the solution reversed. You must force the pointer dereference to occur
before structure dereference... see the other posts.

If you had something like this:

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

You can use the precendence to your advantage, visa vis:

print,*filter.points

Keeping the precendence in mind can eliminate extraneous parentheses, which, for
everyone except lisp programmers, often add confusion.

JD

```
--
 J.D. Smith                             |*|      WORK: (607) 255-5842
 Cornell University Dept. of Astronomy  |*|            (607) 255-6263
 304 Space Sciences Bldg.               |*|       FAX: (607) 255-5875
 Ithaca, NY 14853                       |*|
```

Subject: Re: pointer to structures
Posted by John-David T. Smith on Tue, 04 Apr 2000 07:00:00 GMT

View Forum Message <> Reply to Message

"Liam E.Gumley" wrote:

```
>
> eeeyler@my-deja.com wrote:
>> 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
>
> ptr = ptr_new({test:indgen(10)})
> print, (*ptr).test
>       0       1       2       3       4       5       6       7
> 8       9
>
> If you want to keep it really simple and clean, separate the pointer
> de-reference and the structure reference:
>
> struct = *ptr
> print, struct.test
>
> This can make your code much more understandable when multiple levels of
> de-referencing are required (say if the structure contains a pointer to
> an array).
```

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)
```

but you eventually get the hang of it.

JD

--
```
 J.D. Smith                             |*|      WORK: (607) 255-5842
 Cornell University Dept. of Astronomy  |*|            (607) 255-6263
 304 Space Sciences Bldg.               |*|       FAX: (607) 255-5875
 Ithaca, NY 14853                       |*|
```

Subject: Re: pointer to structures
Posted by davidf on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

eeeyler@my-deja.com (eeeyler@my-deja.com) writes:

> 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

The problem here is that a pointer dereference has the lowest
order of precedence. Lower, even, than a structure dereference.
So, you must first dereference the structure (by using
parentheses), and *then* dereference the pointer, like this:

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

Cheers,

David
--
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 wbiagiot on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

In article <8cd1er$2pp$1@nnrp1.deja.com>,
eeeyler@my-deja.com wrote:

> 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!
>
From the IDL help:
print, (*filter).points
> Sent via Deja.com http://www.deja.com/
> Before you buy.
>
--
"They don't think it be like it is, but it do."
Oscar Gamble, NY Yankees

Sent via Deja.com http://www.deja.com/
Before you buy.

Subject: Re: pointer to structures
Posted by Liam E. Gumley on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

eeeyler@my-deja.com wrote:
> 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

```
ptr = ptr_new({test:indgen(10)})
print, (*ptr).test
     0      1      2      3      4      5      6      7
8      9
```

If you want to keep it really simple and clean, separate the pointer
de-reference and the structure reference:

```
struct = *ptr
print, struct.test
```

This can make your code much more understandable when multiple levels of

de-referencing are required (say if the structure contains a pointer to
an array).

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: pointer to structures
Posted by Ben Tupper on Tue, 04 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

eeeyler@my-deja.com wrote:

> 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!
>
>

Hello,

You must first derefence the pointer before derefencing the structure.

```
IDL> print,(*filter).points
a b
```

This is the exact problem I bump into all the time when I have structures
of pointers or pointers of structures of pointers. I always have to slow
down and noodle it out. Unfortunately, for me, it has been just like
riding a bike... I seem to always forget.

Good luck,

Ben

--
Ben Tupper

Bigelow Laboratory for Ocean Science
tupper@seadas.bigelow.org

pemaquidriver@tidewater.net

Subject: Re: pointer to structures
Posted by John-David T. Smith on Wed, 05 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

```
"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.

JD

--
 J.D. Smith |*| WORK: (607) 255-5842
 Cornell University Dept. of Astronomy |*| (607) 255-6263
 304 Space Sciences Bldg. |*| FAX: (607) 255-5875
 Ithaca, NY 14853 |*|

Subject: Re: pointer to structures
Posted by Liam E. Gumley on Wed, 05 Apr 2000 07:00:00 GMT

View Forum Message <> Reply to Message

"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).

Cheers,
Liam.