Subject: Re: Operator precedence
Posted by davidf on Mon, 10 Jul 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Harvey Rarback (rarback@ssrl.slac.stanford.edu) writes:

> Thanks for confirming this. Would someone who has clout with RSI get this
> tidbit into the documentation on Operator Precedence?

I nominate JD. Second? Done.

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: Operator precedence
Posted by Harvey Rarback on Mon, 10 Jul 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Harvey Rarback (rarback@ssrl.slac.stanford.edu) writes:
>
>> Structure field extraction and array indexing have equal precedence, higher
>> than pointer dereference but lower than parentheses to group expressions.
>>
>> Is this statement true?
>
> True enough, I think. :-)

Thanks for confirming this. Would someone who has clout with RSI get this
tidbit into the documentation on Operator Precedence?

> I'm in the midst of a dozen things, Harvey, and I have
> to teach courses the next couple of weeks. And JD is going
> to give us the definitive answer anyway. But here is a quick
> stab at this.
>
> Your problem lies here:

>
>> pro obj1__define
>> obj1 = {obj1, obj2:obj_new()}
>> end
>
> Things would behave very much as you expect them to if
> you had only *inherited* obj2 instead of putting it into
> this structure as an object reference:

I use object inheritance when it seems appropriate, but not in the instance I was complaining about. Let me explain. obj1 is a complicated widget application. obj2 is an object to replace widget_table. I would have logic and implementation issues with subclassing obj2:

logic: obj2 is not an "is a" to obj1. obj1 "has a" obj2.

implementation: making sure that the obj1 and obj2 have unique tags takes away some of the flexibility of having a standalone widget_table.

what if obj1 has multiple obj2's?

>> ; next line prints data but produces
>> ; % Temporary variables are still checked out - cleaning up...
>> print, (obj1.obj2).data
>
> Yeah, I don't have a clue what this is doing, but anytime
> you get that error message it sure as hell isn't what you
> *want* to be doing. It's probably going crazy trying to
> figure out what you had in mind. I think "Temporary variables
> still checked out" is the IDL equivalent of throwing up your
> hands and going to lunch in the real world. :-)

I love your anthropomorphizing. It's what makes this newsgroup so much fun to read.

"J.D. Smith" wrote:
>
> You have found a bug in IDL's object data encapsulation code, which likely
> arises from the internal data-storage equivalence of structures and object
> heap data.

Ah, the predicted "definitive answer".

> I've found you can access the full data without error or warning if you use

> two steps:
>
> o=obj1.obj2
> print,o.data

I have discovered this too, but didn't want to let on in order to hear about other suggestions.

Thanks much to the two of you.

--Harvey

Subject: Re: Operator precedence
Posted by John-David T. Smith on Mon, 10 Jul 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Harvey Rarback wrote:
>
> Folks,
>
> I have a couple of questions regarding operator precedence. From this newsgroup
> and some experimentation I believe the following statement is true:
>
> Structure field extraction and array indexing have equal precedence, higher than
> pointer dereference but lower than parentheses to group expressions.
>
> Is this statement true?
>
> So for nested structures struct1.struct2.data produces the same result as
> (struct1.struct2).data as expected. However, for nested objects (example code
> appended) these rules don't seem to apply:
>
> obj1.obj2.data produces an error
> (obj1.obj2).data produces the expected result, along with the infamous
> % Temporary variables are still checked out - cleaning up...
>
> Can some kind soul enlighten me about this behavior?
>

You have found a bug in IDL's object data encapsulation code, which likely arises from the internal data-storage equivalence of structures and object heap data.

As David points out, RSI made the (dubious) choice of enforcing full encapsulation... object instance data is available directly only within the methods of its class. So by all rights, this should not work at all, and you should get the error you would have if tried from the command line: "% Object

instance data is not visible outside class methods."

I've found you can access the full data without error or warning if you use two steps:

o=obj1.obj2
print,o.data

Presumably IDL is getting confused about which type of object it is dealing with in relation to the method affiliation. Since objects and structures share so much in common, RSI likely had to append encapsulation functionality with no doubt some interesting hackery. I strongly recommend against relying on this feature, since they are quite clear about: "only allowing access to an object's instance data via that object's methods," and will likely fix it in future releases. So, like the rest of us, you'll be stuck writing "GetProperty" methods ... but at least we now have "_REF_EXTRA" for implementing these correctly.

JD

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

Subject: Re: Operator precedence
Posted by davidf on Mon, 10 Jul 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Harvey Rarback (rarback@ssrl.slac.stanford.edu) writes:

> I have a couple of questions regarding operator precedence. From this newsgroup
> and some experimentation I believe the following statement is true:
>
> Structure field extraction and array indexing have equal precedence, higher than
> pointer dereference but lower than parentheses to group expressions.
>
> Is this statement true?

True enough, I think. :-)

> So for nested structures struct1.struct2.data produces the same result as
> (struct1.struct2).data as expected. However, for nested objects (example code
> appended) these rules don't seem to apply:
>
> obj1.obj2.data produces an error

> (obj1.obj2).data produces the expected result, along with the infamous
> % Temporary variables are still checked out - cleaning up...
>
> Can some kind soul enlighten me about this behavior?

Oh, dear. :-(

I'm in the midst of a dozen things, Harvey, and I have
to teach courses the next couple of weeks. And JD is going
to give us the definitive answer anyway. But here is a quick
stab at this.

Your problem lies here:

> pro obj1__define
> obj1 = {obj1, obj2:obj_new()}
> end

Things would behave very much as you expect them to if
you had only *inherited* obj2 instead of putting it into
this structure as an object reference:

```
pro obj1__define
obj1 = {obj1, INHERTS obj2}
end
```

Now a statement like:

```
print, obj1.data
```

makes sense, since the data field is part of obj1 (via
the field that was inherited from obj2). But you
made a field in obj1 that is an object reference:

> obj1 = {obj1, obj2:obj_new()}

Hence, the only way to "see" that data field is to write
a method, since an object can only be dereferenced as a structure
in its own methods. That is why this statement:

```
Print, obj1.obj2.data
```

causes a problem. The obj2.data part is illegal. In fact, obj1.obj2 would
need to be a structure for the statement to be legal, and it is not.
It is an object. :-)

What would work is something like this (assuming you had written the
obj2__getdata method, of course):

Print, obj1.obj2->GetData()

> ; next line prints data but produces
> ; % Temporary variables are still checked out - cleaning up...
> print, (obj1.obj2).data

Yeah, I don't have a clue what this is doing, but anytime
you get that error message it sure as hell isn't what you
want to be doing. It's probably going crazy trying to
figure out what you had in mind. I think "Temporary variables
still checked out" is the IDL equivalent of throwing up your
hands and going to lunch in the real world. :-)

Hope this puts you on the right page, anyway.

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