Subject: Re: Pointer Behavior Objects Vs Plain routines?
Posted by David Fanning on Wed, 11 Sep 2002 16:37:17 GMT
View Forum Message <> Reply to Message

savoie@nsidc.org (savoie@nsidc.org) writes:

>> Procedures can change things that are passed by reference.
>> They work on *copies* of things that are passed by value.
>
> O.k. I'll agree with that, I actually thought it was being passed by value.
> But thought, shouldn't a pointer and a copy of a pointer point to the same
> thing?

Yes, and I'm sure it does. The problem is not that the pointer
doesn't point to the same thing, the problem is that you can't
get the pointer that is pointing to the same thing back to the
calling program! You don't have a passing mechanism in the program
in the way you wrote it.

> *Morning coffee hits* Aha, but I'm doing is defining what it points to _the
> first time_ with a copy. This changes the copy, making it a valid pointer,
> leaving my original pointer alone. Duh.

You must be drinking the Arabian variety. What I'm drinking
isn't strong enough to figure out what this means. :-)

> But if the pointer is already valid, I should be able to dereference the copy
> and store whatever I want, blissfully ignorant of what it was pointing at
> thanks to the magic of IDL pointers.

I *think* this is right, although I'm not convinced I'm
following the logic every step of the way here.

>
>> Since your DOIT method is a self method, you can simple
>> change it like this:
>>
>> PRO WEIRD::DOIT
>> self.myptr = ptr_new('Why can not I change this?')
>> END
>>
>> Then, call it like this:
>>
>> self -> Doit
>
> If Doit didn't have to act on a whole bunch of different internal variables, I
> could do that.
>

>
> But it actually does a bunch of repetitive things and is called
>
>   self->DoIt, self.type1internalPointer, 'type1'
>   self->DoIt, self.type2internalPointer, 'type2'
>   self->DoIt, self.typeNinternalPointer, 'typeN'
>
> for several different types. I could redesign to an array of internal
> pointers and an array of types, but since It's already coded the other way....

Why can't you just pass in the "type" as a parameter? It already
knows about all the pointers. Those are obviously in the self structure.

> I can change this Weird::init function to initialize the pointer, and just
> dereference in the WEIRD::DoIt Function.
>
>
> PRO WEIRD::DOIT, ptrInside
>   *ptrInside = 'Look how I can change this?'
> END
>
> ;; Don't forget to make your INIT function , a member function
> FUNCTION WEIRD::INIT
>   self.myPtr = ptr_new('0')
>   return, 1
> END

If you just want to initialize the pointer (make it valid, but
not pointing to anything in particular), you can do this:

```
  self.myPtr = Ptr_New(/Allocate_Heap)
```

Now it is a valid pointer (it can be de-referenced) that points to
an undefined variable.

> And hope/trust that IDL is smart enought to not write over the end of the
> memory like C would. I vaguely remember a thread about growable arrays that
> says I can do this. Anyone think this is a /bad thing/? Rather than just inelegant?

As long as you know what you are doing, even bad things
are sometimes useful.

> Thanks again for such a fast answer!

My pleasure. Did you know you can get service that is twice
as fast for only a third more per month?

Cheers,

David

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