Subject: Re: Pointer Behavior Objects Vs Plain routines?
Posted by savoie on Wed, 11 Sep 2002 16:02:50 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> writes:

> savoie@nsidc.org (savoie@nsidc.org) writes:
>
>> O.k. I'm looking at some pointer weridness. Well, I'm calling it weirdness
>> because I obviously don't understand something that is happening. There are
>> two examples below.
>> <snip/>
>
> The problem here has nothing to do with either pointers
> or objects. The problem is that structure dereferences
> (I.e., self.myptr) are passed by value, whereas passing
> the pointer itself (I.e., myptr) is passed by reference.
> 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?

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.

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.

> 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....

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
```

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?

Thanks again for such a fast answer!

Matt Savoie
National Snow and Ice Data Center, Boulder, CO
