Subject: Re: Creating pointer in structure
Posted by John-David T. Smith on Thu, 01 Nov 2001 20:57:20 GMT
View Forum Message <> Reply to Message

"K. Bowman" wrote:
>
> If I need to define a structure containing a pointer before I know the
> characteristics of the associated heap variable, which of the following
> is preferable? Does it make any difference, or is it simply a matter
> of programming taste?
>
> For example:
>
> a = {point: PTR_NEW()} ;Create struct w/ null pointer
> ... figure out what n is
> a.point = PTR_NEW(FINDGEN(n)) ;Replace null pointer
>
> or
>
> b = {point: PTR_NEW(/ALLOCATE_HEAP)} ;Create struct w/ pointer->undef
> ... figure out what n is
> *b.point = FINDGEN(n) ;Define heap var
>
> Thanks, Ken

I use both methods. When I'd like ptr_valid() to be the ultimate test
of whether anything is in the slot, I use the former. For lists of
variable length, including 0, this is an excellent test, since it
doesn't rely on what precisely is in the slot. It does, however, force
you to check before dereferencing your pointer.

Even an undefined variable could get stuck the slot to indicate
something is there. It's unusual, but you could end up with:

foo=1 & boo=temporary(foo)
a.point=ptr_new(foo)

which is the same really as a.point=ptr_new(/ALLOCATE_HEAP). In this
case:

```
IDL> print,n_elements(*a.point)
           0
IDL> print,ptr_valid(a.point)
  1
```

On the other hand, when I have a list which will always have at least
one element, I often use the latter method, to remind me of the fact.
Then you can dereference freely, and use n_elements() to see just what

is there.

As long as you remember the difference, you'll be fine.

One other case where having true null pointers is useful: when saving objects or structures, it's sometimes useful to kull out useless bits (like widget states and id's), which aren't really appropriate for keeping on disk. If you've stuck them behind a pointer, you can just do a:

state=self.myuselesswidgetstate
self.myuselesswidgetstate=ptr_new()

before saving and restore it afterwards.

JD
