Subject: Re: Object Data and pointer assignments
Posted by John-David T. Smith on Thu, 09 Mar 2000 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> J.D. Smith (jdsmith@astro.cornell.edu) writes:
>
>> Just to be clear... you are free to free self.inarray, and point it somewhere
>> else, at any time. This can be useful if you have a list which is either empty
>> (NULL pointer a.k.a. a dangling reference), or not (pointer to a list of finite
>> size). If the list changes size, and becomes empty again, you can simply free
>> it, which indicates its emptiness. If it then grows again, simply use ptr_new()
>> to get another heap variable for it. So, while it might be easiest in some
>> cases only to call ptr_new() once, in other cases it is useful to let a single
>> member variable like self.inarray point to different heap variables over its
>> life.
>
> Lord knows I need more excitement in my life if I'm quibbling with
> quibbles, but let me make one suggestion:
>
> If I want to point to an "empty" variable, I prefer to
> use a pointer to an undefined variable. The advantage
> to me is that this is a VALID pointer, in contrast
> to the NULL pointer, which is an invalid pointer.
>
> Note:
>
> IDL> a = Ptr_New()
> IDL> Print, Ptr_Valid(a)
> 0
> IDL> *a = 5
> % Unable to dereference NULL pointer: A.
>
> IDL> b = Ptr_New(/Allocate_Heap)
> IDL> Print, Ptr_Valid(b)
> 1
> IDL> *b = 5
>
> I like this because it fits into the programming style
> I've developed. For example:
>
> IF N_Elements(color) EQ 0 THEN color = 5
> IF N_Elements(*b) EQ 0 THEN *b = 5
>
> But again, you must *initialize* this pointer to an
> undefined variable in the INIT method, NOT in the __DEFINE
> module.

>

That's a nice idea. I hadn't thought of doing it that way. In my method, the validity of the pointer is what indicates an empty vs. non-empty list. In your method, whether the variable pointed to by the pointer is defined provides the same distinction. With your method, you save yourself tests like:

```
 if ptr_valid(ptr) n_elem=0 else n_elem=n_elements(ptr)
```

(of which I have *many*) in favor of:

```
 n_elem=n_elements(*ptr)
```

This is very clean. To pay for that, though, each time your list (or whatever) reaches 0 size, you must do a:

```
ptr_free,ptr
ptr=ptr_new(/ALLOC)
```

the latter line not being required in my method (a consequence of the indistinguishability of null pointers and dangling pointers). I think this trade is well worth it, though, and I will consider using your method in the future.

Thanks for the tip!

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 |*|