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:
>
> Ben Tupper (tupper@seadas.bigelow.org) writes:
>
>> I am in the middle of wrtting my first object from scratch. Scratch is
>> a good word since I'm doing a lot of that on my head. I'm hoping to get
>> some advice on organization of data. I need 4 pieces of data (one 2d
>> arrays and two structures that vary in size according to the size of the
>> arrays) plus six keywords that I need to get/set. Currently, I have
>> defined each of the 3 bits of data as null pointers in the BLAH__DEFINE
>> procedure.
>>
>> In the BLAH::INIT function, the user passes one of the two arrays as an
>> argument. At that point I reassign one of the pointers to...
>>
>> Self.InArray = Ptr_New(InArray).
>>
>> I think I understand why I can reassign the structure field when going
>> from a null pointer to a filled pointer. On second thought, I don't
>> understand it but I can accept that it works. It's the next step I need
>> help on.
>
> The reason you need to use an actual pointer (Ptr_New) here,
> is that you *don't* have a pointer from the BLAH__DEFINE
> module. What you have done in that module is said that the
> *definition* of the InArray field *will be* a pointer. In other
> words, the BLAH__DEFINE module only *defines* the object and
> its fields, it doesn't assign anything to the self object. This
> is what must be done by the INIT method.

This is true. But keep in mind that all structure defines (e.g. a={BLAH_STRUCT}) zero the structure members upon creation. In the case of arrays, it fills them with zeroes. In the case of strings, it makes them zero length. In the case of pointers and objects, it makes them null pointers/objects. These *are* real pointers, but ones which cannot be dereferenced, as they point nowhere! They are "dangling references" at birth. Note that you can say:

```
IDL> a=ptr_new(1)
IDL> b=a
IDL> ptr_free(a)
```

and b will be, for all practical purposes, a NULL pointer! It points to a heap variable which does not exist. It has the same ptr_valid() properties, etc.

Only printing it can reveal the difference.

The only reason I quibble is to dispel the notion that the __define in class definitions merely suggests or defines the class data members, as it does for fields of structures, and that you must "fill out" the skeleton of class data in the Init method. In the context of object creation, obj_new *does* in fact implicitly assign values to them all: namely null (0,'',null pointer, etc.). You can think of the first step of obj_new being something like self={MY_CLASS}. So by the time you get to the Init Method, you do have a *real* pointer, namely a null pointer.

The moral is: if null class data is what you want, you can skip the Init (though of course there are other advantages to Init'ing), or skip irrelevant assignments in Init (e.g. self.var=0).

One curious by-product of structure/class definition/instantiation peculiarities is shown in the following example:

```
pro foo__define
  foo={FOO, $
       ptr: ptr_new(fltarr(100))}
end


IDL>  a={FOO}
% Compiled module: FOO__DEFINE.
IDL> print,a
{<NullPointer>}
IDL> help,/heap
Heap Variables:
   # Pointer: 1
   # Object : 0

<PtrHeapVar1>   FLOAT     = Array[100]
```

An IDL-provided memory leak! Yay. The moral of that is, *never* use anything but the bare ptr_new() and obj_new() inside of your __defines.

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 |*|