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

Ben Tupper wrote:
>
> Hello,
>
> 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.

A pointer is a pointer is a pointer, whether to nothing (a NULL pointer), or to an array of 1 million images. Think of a pointer as just some number, like <1185>, and it won't be as confusiong. In your case the structure field just contains one of these special "numbers". In a machine-level language, the number would be a hardware address. In IDL, the number is some arbitrary lookup in an internal table of pointer heap data.

>
> I would like to change the contents of this field later to some other
> value (a differently sized array.) Here's where the ice under me gets
> very very thin and my eyes get misty. In the BLAH::SETPROPERTY method,
> I don't know if I should free this pointer before reassigning (and does
> that leave the structure field undefined?), or if I should simply
> overwrite it as I did in the INIT function. If I reassign the filed
> to a new pointer, what happens to the previously occupied heap space?
> Have I sprung a leak?

Two possibilities:

*ptr=newarr

or

ptr_free,ptr
ptr=ptr_new(newarr) ; with optional /NO_COPY keyword -- faster but makes newarr undefined

In the first case, think of it just like reassigning a regular variable. If you say:

```
IDL> a=[1,2,3]
IDL> a=[4,5,6,7,8,9]
```

you don't worry that the space allocated for the [1,2,3] is lost... no memory leak occurs here. IDL makes sure of that (or they are supposed to!). Likewise, when changing the value of the pointer heap data (which in essence is just a specially accessed and globally persistent form of a regular variable like the "a" above), the exact same rules apply.

The only way you will have memory leaks is if you reassign your ptr elsewhere without freeing the heap data. E.g.

```
IDL> a=ptr_new(findgen(100,100))
IDL> help,/heap
Heap Variables:
   # Pointer: 1
   # Object : 0

<PtrHeapVar1>   FLOAT     = Array[100, 100]
IDL> a=ptr_new(5)
IDL> help,/heap
Heap Variables:
   # Pointer: 2
   # Object : 0

<PtrHeapVar1>   FLOAT     = Array[100, 100]
<PtrHeapVar2>   INT       =        5
IDL> help
% At  $MAIN$
A               POINTER   = <PtrHeapVar2>
```

You see we have two "heap" variables, PtrHeapVar1 and 2, and and only variable ("a") pointing to #2. The data in PtrHeapVar1 is not referenced by anyone at all. What can be done about this?

1. The first and best answer is: don't let it happen in the first place. Careful programming can avoid all leaks like this.

2. For debugging purposes, if you happen to "lose" some data you had pointed to, you can recover it, i.e. reconnect a pointer to it. In the above example, you

would say:

```
IDL> b=ptr_valid(/CAST,1)
IDL> help
% At  $MAIN$
A               POINTER   = <PtrHeapVar2>
B               POINTER   = <PtrHeapVar1>
```

so now PtrHeapVar1 has a pointer to it after all. The memory leak has been averted, and more importantly, you now have access to that possibly important data. Use this method only in emergencies, and for debugging code which develops leaks (see 1.).

3. When desperate to stem the flood of leaking memory, you can use heap_gc, which looks for all heap variables without one or more pointers pointing to them, and free's them. This is not recommended for anything other than debugging purposes, except as a quick fix when something has to be done yesterday.

It's not terribly hard to write leak free programs. Just keep these few things in mind.

Good Luck,

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 |*|