Subject: Re: allocate heap? yes or no? Posted by marc schellens[1] on Mon, 17 May 2004 06:45:39 GMT View Forum Message <> Reply to Message

- > Here's a quick pointer question for the gurus in the group.
- > When creating a new pointer, when is it appropriate or not appropriate
- > to set the allocate heap keyword? In a nutshell, could someone please
- > summarize the issues that need to be considered. Are there downsides
- > in terms of speed or extra commands that need to be considered. Is
- > array size a consideration?

Internally, each pointer is a long integer, but IDL knows that these are not to be treated like integers but are indices to a special data structure, the "heap" (to keep is simple, think of it as a dynamic array, ie. an array which can be resized (in real it is probabaly some kind of tree) each single element of this heap can hold *any* IDL data structure (arrays, structs, scalars...). If you create a pointer with ptr_new() or ptrarr(), each of this pointers (long integers) is set to 0. (IDL knows, that heap index 0 cannot be dereferenced).

If you create a pointer with eg.:

p=ptr new(10), the heap is extended by one element, and into this element the integer 10 is stored.

And p itself is set to the index to the heap array (eq. 1 if it is the first allocation)

if you do now:

q=ptr new(dblarr(3,3,3)) again the heap is extended by one element and in this elements a dblarr(3,3,3) is stored. (q is set to eq. 2)

If you do:

r=ptr_new(/ALLOCATE_HEAP) the heap is extended by one, but nothing is stored there (ie. <Undefined>). (in our example, r is set to 3)

But later you can say:

*r=something

(*r is the heap element with index 'r')

for eg:

t=ptr new() (the heap is not extended yet) you would have to say later: t=ptr new(something)

- > I rely on pointers quite heavily these days. I just wish I knew more
- > about the dos and don'ts.

Mainly you want to use ptr new() and ptrarr() when you know, that each

of this poiters is set (allocated) and freed at a specific time. (Eg. you want to read in 10 images of different size, at one point of your program).

If the allocation of a pointer is variable (data or user input dependent), and/or the pointer might be reset to another value later, you are probably better of with ptr_new(/ALLOC). Because for an allocated pointer you can check if it is already set just with:

if n_elements(*p) ne 0 then ...

while for non allocated pointers you would have to check first if you can dereference it:

if ptr_valid(p) then if n_elements(*p) then ...

HDH, marc

Subject: Re: allocate heap? yes or no?
Posted by David Fanning on Mon, 17 May 2004 13:26:34 GMT
View Forum Message <> Reply to Message

Marc Schellens writes:

Marc pretty much covers all the ground related to pointers. I was trying to think if I use a rule of thumb about when and where to allocate heap. Then I realized I *used* to do a lot of heap allocating, but these days I hardly do it at all.

I think when I started working with pointers (and especially when they were associated with objects) I had several "entry" points in the program where pointers could be set (INIT, SetProperty methods, etc.) It seemed smart to allocate heap in the INIT method, then I didn't have to worry about checking if the pointer was valid in the SetProperty method.

But I think what happened is that I found it too easy to leak pointer memory with this approach. These days I do my "checking" with PTR_VALID instead of N_ELEMENTS and if the pointer isn't valid, I just make a new one. This makes the pointer easy to destroy at appropriate times without worrying about whether I have to also allocate a new pointer with heap memory. It just makes the pointer accounting a LOT easier for me if I *don't* allocate heap.

\sim	h۰	_	-
C	ne	;ei	5

David

--

David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/