Subject: Re: Passing Image Data :)
Posted by John-David T. Smith on Tue, 23 Oct 2001 21:29:22 GMT
View Forum Message <> Reply to Message

Logan Lindquist wrote:
>
> Dr(.) Fanning,
> [I thought Dr. had an period after it because it is an abbrevation of
> doctor? I do not know what Andrew Cool is talking about.]
>
>> What you want in your info structure image field is a pointer
>> to the image:
>>
>> info= { image:Ptr_New(myimage), ...}
>
> I am wondering if you could clear up a couple of things about pointers in
> IDL. How come myimage does not have to be defined during initalization? Does
> the statement above create space in memory for a variable of indefinite
> size? It seems to operate this way., where the data in memory is allocated
> once the data has to be stored to the pointer array. Maybe I am
> understanding pointers incorrectly.
>
> 1.. The Pointer is created - a variable that 'points' to space in RAM
> reserved for a variable of indefinate size.
> 2.. The data is read into RAM during the read_image.pro.
> 3.. The Pointer then needs to store the image data for future reference.
> This is done by '*info.image = newimage'. Where newimage is the image data
> in RAM.
> 4.. Is the data then copied into the space originally allocated for it or
> does it simply change it's reference so as to point to the location in RAM
> where the image data was read into?
>
>> *info.image = newimage
>>
>> IDL takes care of all the memory management for you. You don't
>> have to worry about it.
>

Please (re)read David's excellent synopsis of what IDL pointers really
are: access points for otherwise normal IDL variables which live on a
global heap. As far as the memory allocation for pointers, you have to
worry about it only as much as you have to worry about memory allocation
for normal IDL variables (i.e., not too much). Example:

IDL> a=fltarr(1000)
IDL> a=5

where did all that memory for the vector go? The ocean floor? Who

knows.... IDL took care of it for us.

For saving memory and speeding pointer assignments, look to the NO_COPY keyword for ptr_new, e.g.:

IDL> info.image=ptr_new(newimage,/NO_COPY)

which causes newimage to be undefined, and simply transforms it into a pointer heap variable, now referenced by info.image.

The equivalent normal-variable operation would be:

IDL> image=temporary(newimage)

which also results in leaving newimage undefined. You could obviously also do something like:

IDL> *info.image=temporary(newimage)

to mix the two technologies. All work exactly the same way.

One caveat: IDL manages memory for individual variables (normal or heap) quite nicely. It does *not* ensure that heap variables which are no longer referenced are freed: you must do this yourself. Note the subtle distinction: data attached to individual variables is book-kept; the collection of variables on the heap is not book-kept.

One more point. Objects variables are really just pointers into a special heap (called, remarkably, the "object heap"), and have the same bookkeeping issues as pointers. The only difference is, they can hold only one type of data, and have special assignment, access, and method invocation syntax. From a memory management point of view, however, they are identical: you can strand unreferenced object variables on the heap just as well as you can pointers variables.

JD
