Subject: Re: simple question (I hope)
Posted by Foldy Lajos on Fri, 30 Mar 2007 18:39:10 GMT
View Forum Message <> Reply to Message

On Fri, 30 Mar 2007, JD Smith wrote:

> This isn't quite correct. Everything, and I mean everything, in IDL is
> passed by reference. However, when IDL encounters a statement like
> `array[x]', or `struct.y', or total(array), it first creates a temporary
> variable to hold the results of the array indexing or structure
> de-reference, or function call. Other than the fact that this variable
> isn't accessible externally, it is just a regular old IDL variable (does
> this remind you of heap variables in the pointer tutorial?). This
> temporary variable is passed, just like all other variables in IDL, *by
> reference* into a calling procedure, e.g.:
>
> mypro, array[x] ---> mypro, some_internal_idl_temp_var1234
>
> Since you can't access that temporary variable explicitly, this is
> effectively the same as pass by value. You can now set
> some_internal_idl_temp_var1234 to your heart's content, but you'll never
> be able to recover the special value you put there:
>
> pro mypro, arr
> arr[0]=42
> end
>
> IDL> a=randomu(sd,100,1000,100)
> IDL> mypro, a[0:800,*,*]
> IDL> help,a
> A FLOAT = Array[1000, 1000, 100]
> IDL> print,a[0]
> 0.776156 ; wherefore art though, 42?
>
> The one difference which makes this distinction more than pedantic is
> that true pass by value is very inefficient for large arrays. In a
> pass-by-value scheme, all of that data (801x1000x100) would be copied
> via the stack into the local address space of the routine MYPRO. It may
> sound like a subtle difference, but it does represent a real gain in
> efficiency, in particular when the temporary variable has a life outside
> the called routine. Eventually, all temporary variables are harvested,
> and their memory freed. So while you can't ever get at them yourself,
> they do offer advantages.
>
> JD
>

I don't know how IDL is implemented, but I use pass-by-value for temporary

variables in FL. Here "pass" means move, not copy ("move semantics"). The original temporary does not exist after entering the called routine, it is undefined. The called routine gets values, not references. It is faster than pass-by-reference, since no de-referencing is needed for these variables.

regards,
lajos
