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:
>
> "J.D. Smith" wrote:
>
>>
>> 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.
>>
>>
>
> Hello,
>
> This doesn't sound like a quibble to a newbie. So, the filling out of the
> skeleton may or may not occur in the INIT function, but not in the __DEFINE
> procedure.
>
> You have brought up the the issue of SELF ... which is another source of
> confusion. I have just written the SetProperty and GetProperty methods. They
> work just fine (so far I can get and set what I need). However, it feels a little
> bit like living in Flatland where things pop in and out of my two dimensions from
> some unkown third dimension. Magically, SELF appears as if out of thin air; how
> does it get there if there is no SELF argument in the procedure... and how come I
> don't have to call the structure BLAH (rather than self) if it is a named
> structure?

Think of self as an invisible final argument, passed by reference to every single method of the class. This is most certainly how RSI even implemented it. All object oriented languages have some similar way to access easily the "member data" of an object, whether it be called "self", "this", "here", etc. So, it's not too much magic, just a little bit of argument hiding. Just as a regular variable can hold a named structure... i.e. a={THIS_STRUCT,data1:1}, the specially named variable "self", which implicitly exists in all methods, can hold a named class "structure" (really it's a full fledged object... the distinction being that I couldn't do a->Print or some such on the above definition). The "self" is "a good thing", and provides part of the object oriented solution.

Heavy duty magic makes use of the fact that self is more or less just a passed

in, by reference variable, which allows you to switch, in place, the self object (just as you could overwrite any other variable passed in by reference). For instance, in one of my applications, a simple "restore from disk" menu option running *within a method* on a given object performs such a voodoo transmogrification to undo all changes made since the last save without tediously setting all of the (many) data members. So my advice: have fun with your self.

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 |*|