Subject: Naive pointer question ?
Posted by the_cacc on Tue, 22 Jan 2002 21:59:25 GMT
View Forum Message <> Reply to Message

From my understanding, pointers are useful in C where you have the option of passing by value or by reference, but in IDL it seems one *always* passes by reference (insofar as any variable passed to a function and then changed within that function is also changed in the calling function).

So I was wondering, what is the benefit of explicitly using pointers in IDL ?

Ciao.

Subject: Re: Naive pointer question ?
Posted by the_cacc on Thu, 24 Jan 2002 10:45:51 GMT
View Forum Message <> Reply to Message

Yo,

Thanks for the input - I've been looking through my IDL library looking for routines to adapt for pointers but can't see any very obvious candidates (don't need data structures beyond trivial in complexity).
However, one useful app for me is Paul's suggestion for 'a data array where each "element" is another array, and each one is a different size'. Would I need to store the length of each element in a separate array ? And then store both arrays in a structure ?

I didn't understand what Bhautik said, that '[use of pointers] runs against the method of passing data via common blocks (which are EVIL! EVIL! EVIL!) -
data is instead passed down a heirachy of functions'. Since much of my code is cursed with COMMON blocks (eg. when using CURVEFIT, AMOEBA, etc), any way to avoid this would be preferable. Well, not *any* ;)

Ciao.

Subject: Re: Naive pointer question ?
Posted by Paul van Delst on Thu, 24 Jan 2002 14:53:14 GMT
View Forum Message <> Reply to Message

trouble wrote:
>

> Yo,
>
> Thanks for the input - I've been looking through my IDL library
> looking for routines to adapt for pointers but can't see any very
> obvious candidates (don't need data structures beyond trivial in
> complexity).
> However, one useful app for me is Paul's suggestion for 'a data array
> where each "element" is another array, and each one is a different
> size'. Would I need to store the length of each element in a separate
> array ? And then store both arrays in a structure ?

I don't think that's necessary. A simple example would be (excuse any syntax errors - it's been a while since I've actually used IDL....):

```
  ; -- Define array sizes simply for this example
  ; -- In real app, array sizes are determined at runtime
  array_sizes = [ 10, 2300, 45, 300, 3 ]
  n = N_ELEMENTS( array_sizes )

  x = PTRARR( n )
  FOR i = 0, n - 1 DO BEGIN
    x[i] = PTR_NEW( FLTARR( array_sizes[ i ] ) )
  ENDFOR
```

I can then stick data in any particular array and/or check it's length

```
  length = N_ELEMENTS( *x[i] )
```

etc etc. layering onwards to further complexity as needed. Usually I have structures that contain pointer arrays to other structures which contain pointer to arrays of varying sizes. Sounds messy, and sometimes is, but it does make for quite flexible data structures.

Depending on the complexity of your data structure, using an object might be the way to go - you get a bit more protection wrt to killing/dropping/losing pointer references and a simpler interface to the data but at the expense of even more complexity (and time designing it all in your head/on paper).

cheers,

paulv

p.s. Common blocks are not evil. They are unfairly maligned (and misaligned) but should be used judiciously.... just like GOTOs.

--
Paul van Delst Religious and cultural
CIMSS @ NOAA/NCEP purity is a fundamentalist
Ph: (301)763-8000 x7274 fantasy

Fax:(301)763-8545 V.S.Naipaul

Subject: Re: Naive pointer question ?
Posted by David Burridge on Thu, 24 Jan 2002 15:30:35 GMT
View Forum Message <> Reply to Message

Hi Paul et al.

> <BIG snip>
> p.s. Common blocks are not evil. They are unfairly maligned (and misaligned) but should be used
> judiciously.... just like GOTOs.

Sorry, I couldn't resist this!!! The problem with common blocks two-fold, both relating to their global scope (ie: the reason for using them in the first place). First, if you have two programs running in an IDL session (e.g. a pair of widget programs) which both use the same common block, each will corrupt the others data. In fact, you might even just be using two routines that don't realise another is accessing the common block. Secondly, you cannot trace the changes in a common block, so debugging becomes a nightmare. This is even worse in IDL when variables can change TYPE as well as value.

That's it. Rant over:-) Let's just say, I gave them up a LONG time ago!-)

Cheers,

Dave

Subject: Re: Naive pointer question ?
Posted by Paul van Delst on Thu, 24 Jan 2002 17:52:45 GMT
View Forum Message <> Reply to Message

David Burridge wrote:
>
> Hi Paul et al.
>
>> <BIG snip>
>> p.s. Common blocks are not evil. They are unfairly maligned (and
> misaligned) but should be used
>> judiciously.... just like GOTOs.
>
> Sorry, I couldn't resist this!!! The problem with common blocks two-fold,
> both relating to their global scope (ie: the reason for using them in the

> first place). First, if you have two programs running in an IDL session
> (e.g. a pair of widget programs) which both use the same common block, each
> will corrupt the others data. In fact, you might even just be using two
> routines that don't realise another is accessing the common block.

This would most definitely be a non-judicious use of common blocks, i.e. badly designed code. Forgoing common's in those cases usually isn't a solution. (Sure, the code *may* work)

> Secondly,
> you cannot trace the changes in a common block, so debugging becomes a
> nightmare. This is even worse in IDL when variables can change TYPE as well
> as value.

I'll trust you on this one.

paulv

--
Paul van Delst Religious and cultural
CIMSS @ NOAA/NCEP purity is a fundamentalist
Ph: (301)763-8000 x7274 fantasy
Fax:(301)763-8545 V.S.Naipaul

---

Subject: Re: Naive pointer question ?
Posted by Bhautik Joshi on Thu, 24 Jan 2002 22:17:33 GMT
View Forum Message <> Reply to Message

>> <BIG snip>
>> p.s. Common blocks are not evil. They are unfairly maligned (and
> misaligned) but should be used
>> judiciously.... just like GOTOs.
>
> Sorry, I couldn't resist this!!! The problem with common blocks two-fold,
> both relating to their global scope (ie: the reason for using them in the
> first place). First, if you have two programs running in an IDL session
> (e.g. a pair of widget programs) which both use the same common block, each
> will corrupt the others data. In fact, you might even just be using two
> routines that don't realise another is accessing the common block. Secondly,
> you cannot trace the changes in a common block, so debugging becomes a
> nightmare. This is even worse in IDL when variables can change TYPE as well
> as value.
> That's it. Rant over:-) Let's just say, I gave them up a LONG time ago!-)
I'll concur with you there :)

Yes, when a lot of time is spent designing the program, common blocks
can be a reasonable and an easy alternative to more complex options.
However, things like goto and global variables went out of fashion a

very long time ago and have long since proven to be, in event driven programming at least, unsafe and unpredictable. When you are maintaining a piece of legacy code with common blocks all over the place you don't really have any choice but to use them (or alternatively rebuild the whole damn thing from the ground up!) but, I for one prefer to avoid them.

In terms of event driven programming, without proper logical locking mechanisms (mutual exclusion -> semaphores/monitors etc.), global variables are dangerous. Your widget program that reads and writes to, or worse, depends on a global variable shared between many widgets all doing things at different times, might get into a race condition and then things get *really* ugly. I can't really think of an example off the top of my head, so go see:

http://www.cs.mtu.edu/~shene/NSF-3/e-Book/RACE/1st-attempt.h tml

(just something I found via google)

Perhaps I'm just nuts and its too early in the morning for me; please correct me if I'm wrong :)

cheers,
Bhautik
--
/-----------------------------------------------(__)----- ----\
| nbj@imag.wsahs.nsw.gov.au | phone: 0404032617 |..|--\ -moo |
| ICQ #: 2464537 | http://cow.mooh.org | |--| |
|-------------------------+---------------------\OO/|| ------/
| international |
| roast. my sanity has gone |
| its lost forever |
\-------------------------/

---

Subject: Re: Naive pointer question ?
Posted by Craig Markwardt on Thu, 24 Jan 2002 23:17:02 GMT
View Forum Message <> Reply to Message

"David Burridge" <dave@clogic.f9.co.uk> writes:
> Hi Paul et al.
>
>> <BIG snip>
>> p.s. Common blocks are not evil. They are unfairly maligned (and
> misaligned) but should be used
>> judiciously.... just like GOTOs.
>
>

> Sorry, I couldn't resist this!!! The problem with common blocks two-fold,
> both relating to their global scope (ie: the reason for using them in the
> first place). First, if you have two programs running in an IDL session
> (e.g. a pair of widget programs) which both use the same common block, each
> will corrupt the others data. In fact, you might even just be using two
> routines that don't realise another is accessing the common block. Secondly,
> you cannot trace the changes in a common block, so debugging becomes a
> nightmare. This is even worse in IDL when variables can change TYPE as well
> as value.
>
> That's it. Rant over:-) Let's just say, I gave them up a LONG time ago!-)

I think I've weighed in on this before, but I won't let that stop my yapping. I think there are at least two cases where common blocks are pretty nice.

The first one is where you need a persistent store of information. For example, CMPS_FORM() keeps a list of printer configurations in a common block. I also keep large tables in a common block, so they are initialized only once to save CPU cycles. Any time you need a procedure to "remember" something from one call to the next, common blocks are actually a pretty good idea.

Another case is for argument passing between tightly coupled procedures, typically ones in the same file. I use this in MPFITFUN to pass the user's arguments. Since the underlying driver, MPFIT(), doesn't understand about the user function, there must be some way to pass the data along. I tried pointers (handles) for awhile, but a few heap leaks later I was hatin' that. I also use a common so that the user function can (optionally!) return an error code.

The key is access control. In the first case the access to the common block is through a single function, so no other functions should interfere. Part of that solution is giving a name to your common block which is unique so there is virtually no chance of conflict with some other common.

In the second case there is still access control. Generally speaking, I think there should only be one assigned writer to a common block, and any other functions should be readers. Mixed writing and reading can get hairy. Furthermore, in the "use-commons-to-pass-arguments" approach, I always guarantee that when control returns to the user there is nothing vitally important in the common block, so that it can be wiped clean with impunity on the next call.

Common-ly yours,
Craig

--
 ---------------------------------------------------------- --------------
Craig B. Markwardt, Ph.D.  EMAIL:  craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 ---------------------------------------------------------- --------------

---

Subject: Re: Naive pointer question ?
Posted by btupper on Fri, 25 Jan 2002 13:58:31 GMT
View Forum Message <> Reply to Message

On 24 Jan 2002 17:17:02 -0600, Craig Markwardt
<craigmnet@cow.physics.wisc.edu> wrote:

> I think there are at least two cases where common blocks are
> pretty nice.
>
> The first one is where you need a persistent store of information.
> For example, CMPS_FORM() keeps a list of printer configurations in a
> common block. I also keep large tables in a common block, so they are
> initialized only once to save CPU cycles. Any time you need a
> procedure to "remember" something from one call to the next, common
> blocks are actually a pretty good idea.
>
Howdy,

While reading Craig's description of this particular 'memory'
advantage of common blocks, I realized that the word 'object' could be
slipped into the place of 'common block'. Hmmm.

Objectively yours,

Ben

---

Subject: Re: Naive pointer question ?
Posted by James Kuyper on Fri, 25 Jan 2002 16:43:07 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
...

> initialized only once to save CPU cycles. Any time you need a
> procedure to "remember" something from one call to the next, common
> blocks are actually a pretty good idea.

However, it often better if it does that remembering in a block of

memory provided for it by the calling function, and passed in by
reference as an argument. That way it can remember different things on
different calls to the same function, by providing it with different
blocks of memory for each use.

---

Subject: Re: Naive pointer question ?
Posted by Craig Markwardt on Sat, 26 Jan 2002 21:16:17 GMT
View Forum Message <> Reply to Message

btupper@bigelow.org (Ben Tupper) writes:

> On 24 Jan 2002 17:17:02 -0600, Craig Markwardt
> <craigmnet@cow.physics.wisc.edu> wrote:
>
>> I think there are at least two cases where common blocks are
>> pretty nice.
>>
>> The first one is where you need a persistent store of information.
>> For example, CMPS_FORM() keeps a list of printer configurations in a
>> common block. I also keep large tables in a common block, so they are
>> initialized only once to save CPU cycles. Any time you need a
>> procedure to "remember" something from one call to the next, common
>> blocks are actually a pretty good idea.
>>
> Howdy,
>
> While reading Craig's description of this particular 'memory'
> advantage of common blocks, I realized that the word 'object' could be
> slipped into the place of 'common block'. Hmmm.
>
> Objectively yours,

I appreciate that, however, common blocks appear to be the only way to
give a *function* persistent memory. If you use an object or a
pointer instead, you still have to pass this info into the function on
every call. There is indeed a time and a place for that technique,
but I think common blocks can be extremely useful and safe, if used
extremely carefully.

Commonly yours,
Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response

---

Subject: Re: Naive pointer question ?
Posted by Craig Markwardt on Sat, 26 Jan 2002 21:18:33 GMT
View Forum Message <> Reply to Message

James Kuyper <kuyper@gscmail.gsfc.nasa.gov> writes:

> Craig Markwardt wrote:
> ...
>
>> initialized only once to save CPU cycles. Any time you need a
>> procedure to "remember" something from one call to the next, common
>> blocks are actually a pretty good idea.
>
> However, it often better if it does that remembering in a block of
> memory provided for it by the calling function, and passed in by
> reference as an argument. That way it can remember different things on
> different calls to the same function, by providing it with different
> blocks of memory for each use.

Yes, that can be a good way too, especially when there can be multiple
"sessions."

Craig

--
---
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---

Subject: Re: Naive pointer question ?
Posted by btupper on Sun, 27 Jan 2002 22:17:06 GMT
View Forum Message <> Reply to Message

On 26 Jan 2002 15:16:17 -0600, Craig Markwardt
<craigmnet@cow.physics.wisc.edu> wrote:

>
> I appreciate that, however, common blocks appear to be the only way to
> give a *function* persistent memory. If you use an object or a
> pointer instead, you still have to pass this info into the function on
> every call.

Hi Craig,

Ah!  I hadn't thought of that.  Well, no objection here!

Does it count if the object contains the function as a method?

Ben

---

Subject: Re: Naive pointer question ?
Posted by Craig Markwardt on Tue, 29 Jan 2002 00:00:52 GMT
View Forum Message <> Reply to Message

btupper@bigelow.org (Ben Tupper) writes:
> On 26 Jan 2002 15:16:17 -0600, Craig Markwardt
> <craigmnet@cow.physics.wisc.edu> wrote:
>>
>> I appreciate that, however, common blocks appear to be the only way to
>> give a *function* persistent memory.  If you use an object or a
>> pointer instead, you still have to pass this info into the function on
>> every call.
>
> Hi Craig,
>
> Ah!  I hadn't thought of that.  Well, no objection here!
>
> Does it count if the object contains the function as a method?

Hmm, maybe.  What are objects?

Craig

--
 ------------------------------------------------------------ --------------
Craig B. Markwardt, Ph.D.  EMAIL:  craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 ------------------------------------------------------------ --------------

---

Subject: Re: Naive pointer question ?
Posted by the_cacc on Tue, 29 Jan 2002 10:33:40 GMT
View Forum Message <> Reply to Message

Craig Markwardt <craigmnet@cow.physics.wisc.edu> wrote:
> Yes, that can be a good way too, especially when there can be multiple
> "sessions."

>

I'm not sure what people mean when they talk about multiple sessions
and COMMON block memory being corrupted/overwritten. Countless times
I've had several xterms open running the same IDL code (containing
COMMON blocks) on the same machine and never had a problem.

Obviously, multi-threading within a single IDL session will be
vulnerable but this was only made available in the latest release so
they can't mean this. Can they ?

Ciao.

---

Subject: Re: Naive pointer question ?
Posted by Craig Markwardt on Tue, 29 Jan 2002 14:28:01 GMT
View Forum Message <> Reply to Message

the_cacc@hotmail.com (trouble) writes:

> Craig Markwardt <craigmnet@cow.physics.wisc.edu> wrote:
>> Yes, that can be a good way too, especially when there can be multiple
>> "sessions."
>>
>
> I'm not sure what people mean when they talk about multiple sessions
> and COMMON block memory being corrupted/overwritten. Countless times
> I've had several xterms open running the same IDL code (containing
> COMMON blocks) on the same machine and never had a problem.
>
> Obviously, multi-threading within a single IDL session will be
> vulnerable but this was only made available in the latest release so
> they can't mean this. Can they ?

Hey "trouble" --

The basic idea is this. Common blocks are a great way to store
globally accessible information in one place. The disadvantage is,
they store global information in one place.

Let's say you have a widget application that stores its information in
a common block. Maybe you're not familiar with these types of
applications in IDL, but basically you need some kind of memory area,
which keeps information about widget ids, values selected by the user,
etc.

Now if you choose to keep that information in a common block, then
you've basicaly guaranteed that you can never have more than one

widget of that type on the screen at the same time. The reason? A common block can only store one set of information at a time (unless you get really clever). One widget will overwrite the other's data.

Another example is the (currently obsolete) routine called WRITE_GIF, which allows you to write multiple gifs images into a single file. The way this is accomplished is with a little common block inside WRITE_GIF, which keeps the file unit information. There is a problem here, namely you can't have more than one gif file open at a time.

My point was to *use* the fact that a common block can store global and/or persistent information, but *recognize* that it is inherently vulnerable to modification from the outside.

* maintain tight access control over common block
  (give common a unique name; make only one point of access to user)
* enforce programming practices
  (allow as few procedures as possible to write to common block;
   consider that common may be modified when it leaves your control)
* use sparingly
  (use other data structures more appropriate if possible)

The use sparingly part is, if you can find something other than a common block to accomplish the same thing, then use that instead. In the case of widgets, that would be the use of a widget structure stored in the UVALUE, that was pioneered by our very own David Fanning.

Craig

--

 ------------------------------------------------------------ --------------
Craig B. Markwardt, Ph.D.  EMAIL:  craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 ------------------------------------------------------------ --------------


---

Subject: Re: Naive pointer question ?
Posted by the_cacc on Wed, 30 Jan 2002 13:02:48 GMT
View Forum Message <> Reply to Message

Craig Markwardt <craigmnet@cow.physics.wisc.edu> wrote:
>
> Let's say you have a widget application that stores its information in
> a common block. Maybe you're not familiar with these types of
> applications in IDL, but basically you need some kind of memory area,
> which keeps information about widget ids, values selected by the user,
> etc.

>
> -snip-

Ah, widgets - I've not delved far into that area of IDL programming.
Well thanks for the explanation, "Craig" ;)

---