Subject: Re: compile a routine wich inlude a commun
Posted by Paul Van Delst[1] on Tue, 24 Jan 2006 00:14:08 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Paul Van Delst writes:
>
>
>> Well, they're called pointers but they're not, really. You can't actually "point" to
>> anything - just make copies.
>
>
> Well, they aren't fingers, if that's what you mean. :-)

Actually, yes, that's pretty much exactly what I mean. Hence the name: pointers.

>> But, not being a pointer expert, let me ask the question:
>> How *do* you use a pointer in IDL to, uh, well, "point" to an already created variable? Or
>> just parts of an already created array?
>
>
> They are variables that live in a different place
> than local variables.

Well, then they're *not* pointers. They're, umm, "copy/move enablers". :o)

> Outside the room, if you like.
> A pointer can't point to the memory locations of your local
> variable because there aren't enough technical support
> engineers in the world to sort out why your programs
> are suddenly crashing right and left.
>
> Goodness, *all* variables work like this, as far as I
> can tell.
>
> a = 5
> b = a
> b = 3
> print, a
>
> And I don't think you really want to see a 3 printed out!

Sorry, but I don't see the analogy here at all.

```
a=5
b=>a (b points to a)
a=3
print, *b
```

should give you "3", and

```
*b=7
print, a
```

should give you "7". Because in this example b is a (actual) pointer, it refers to the same memory as the variable a. Changing either via assignment changes the value in the memory location.

> Chaos (and I have a high tolerance, myself) would reign supreme.

Erm, not really. This is how actual pointers work (pretty much).

The IDL analogy of a "pointer" is

```
a=5
b=a
b=3
a=b
print, a
```

i.e. it's just copying stuff. Not pointing.

> But, there is no reason you can do what you want to do.
>
> ; Create a local variable.
> x = Indgen(4,4)
>
> ; Move the variable out of the room.
> ptr = Ptr_New(x, /No_Copy)
> help, x
> X UNDEFINED = <Undefined>

See, here's the problem. You can't point to a named variable with IDL pointers. You can copy it onto the heap, but not point to it.

> ; Change a subset of the data
> (*ptr)[1:2, 1:2] = (*ptr)[1:2, 1:2] * 100
>
> ; Move the variable back into the room
> x = Temporary(*ptr)
> Help, *ptr
> <PtrHeapVar2> UNDEFINED = <Undefined>
>
> Print, x
> 0 1 2 3
> 4 500 600 7

```
>       8     900    1000      11
>      12      13      14      15
```

I take issue with the need to "move" these variables around at all. They're pointers. You shouldn't have to move anything. That's, well, the point.

How is the above example any different from

```
IDL> x = Indgen(4,4)
IDL> ptr=temporary(x)
IDL> help, x
X               UNDEFINED = <Undefined>
IDL> ptr[1:2,1:2]=ptr[1:2,1:2]*100
IDL> x=temporary(ptr)
IDL> print, x
       0       1       2       3
       4     500     600       7
       8     900    1000      11
      12      13      14      15
```

??

In both cases you're effectively copying data

I use pointers in Fortran95 for only two things, different things.
1) Point to some nameless space via an allocate statement, e.g.

```
integer,pointer::x(:)
allocate(x(100))
```

This is similar to the IDL statement

```
x = PTR_NEW(lonarr(100))
```

2) Alias subsections of an array, or different arrays

```
integer,target::a(100,100),b(30)
integer,pointer::x(:)
! Make x point to b. Modify elements of x and b changes
x=>b
! Now make x point to slice of a. Modify elements of x and a(:,20) changes
x=>a(:,20)
```

As far as I can tell, there is no equivalent to the above (2) in IDL. The ability to create an alias for a data object without having to copy it - *that's* powerful.

I must be missing something, somewhere.

Anyway, I gotta get home before the bottle shop shuts! :o)

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC