Subject: Re: Naive pointer question ?
Posted by Paul van Delst on Tue, 22 Jan 2002 22:14:05 GMT
View Forum Message <> Reply to Message

trouble wrote:
>
> 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 ?

Let's you easily create complex data structures at the very least (which may or may not be a Good Thing). E.g. imagine a data array where each "element" was another array, and each one was
a different size. One probably could gin together a regular ol' structure containing
(non-pointer) arrays I'm sure, but the code would probably look like chook scratchings through your dog's dinner.

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 James Kuyper on Tue, 22 Jan 2002 22:34:36 GMT
View Forum Message <> Reply to Message

trouble wrote:

> 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

That's one use of pointers. Another, more important one, is in data
structures. A single piece of data can be referred to in two or more
different structures by have a pointer to that data stored in each
structure. In languages without pointers, you can often achieve similar
effects by storing an array index instead of a pointer. However, code
that uses this index needs to know both the array name in order to use
the index to retrieve the value it refers to. That's far clumsier than
the equivalent pointer code.

For instance, try implementing a linked list without pointers (or
equivalent constructs), and then compare the resulting code to the
equivalent code in C. Of course, to appreciate how much simpler the C
code is, you have to be fairly familiar with C, otherwise it will just
look like gibberish.

Subject: Re: Naive pointer question ?
Posted by Bhautik Joshi on Wed, 23 Jan 2002 04:51:58 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 ?

Couple of reasons:

* speed & efficiency (you are passing only a reference to a variable,
but we've covered this already)
* flexibility (easy to create and change dynamic data types)
* future expansion (if you want to temporarily change the data type of
something in a structure during the lifetime of a program while it is
running, create that bit of data as a pointer and simply change where it
points when you want to change the data)

and what I think is the coolest:

* double, triple or higher (!!) dereferencing - a pointer pointing to a
pointer pointing to a pointer pointing to a pointer pointing to a
pointer pointing to a pointer blah blah *foam at mouth & fall over*

They allow you to make complex data structures and types that are good
for many excellent and useful algorithms (such as ones based on linked
lists or trees etc.).

However, on the flipside, if you want to effectively use pointers, you
need to design your program with 'em in mind. Also, it runs against the
method of passing data via common blocks (which are EVIL! EVIL! EVIL!) -
data is instead passed down a heirachy of functions.

Well, anyway, thats my insane rant justifying their use, feel free to
correct me if I'm wrong about anything :)

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 David Fanning on Wed, 23 Jan 2002 05:03:58 GMT
View Forum Message <> Reply to Message

Bhautik Joshi (nbj@imag.wsahs.nsw.gov.au) writes:

> and what I think is the coolest:
>
> * double, triple or higher (!!) dereferencing - a pointer pointing to a
> pointer pointing to a pointer pointing to a pointer pointing to a
> pointer pointing to a pointer blah blah *foam at mouth & fall over*

Having just spent the past three days chasing
leaking memory in a complicated object program
with LOTS of pointers to pointers, I have to say
that this prospect is not as cool to me as it *used*
to be. :-(

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Naive pointer question ?
Posted by James Kuyper on Wed, 23 Jan 2002 16:02:04 GMT
View Forum Message <> Reply to Message

trouble wrote:

> 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).

IDL does NOT always pass by reference. It passes expressions, constants, and system variables by value. For constants, that's quite reasonable. However, for system variables that are writeable, it means that they can't be updated by reference, but only by explicitly assigning them.

Also, subscripted arrays and references to fields of a structure count as expressions, and hence are not passed by reference, a fact that surprised me the first time I got bit by it. From my C background, I expected to be able to pass structure.array_member to a function, and have that function be able to update elements of that array_member. I also expected to be able to pass array(0,*) to a function, and have that function be able to update array(0,2). I understand now why that doesn't work; I'm just saying that it's not what I expected.