Subject: Structures:
Posted by Chris Bull on Wed, 21 Feb 2001 19:15:45 GMT
View Forum Message <> Reply to Message

Hi all,

A question regarding IDL structures, I have an application that I am writing that I am using a very complex
structure as a Uvalue to hold my data in, as this application evolves I am wondering whether it
is possible to add 'branches' to a structure after it is defined (and set as a uvalue), from within a seperate subroutine?

I'm Thinking along the lines of describing an application that can hold many data sets, of different types in memory
and load them up seperatly (much like a imaging program or multiple documents in word) except the data sets are
very complex and of differing structures

Any Ideas?

Sorry I dont have the code here! as this is my home PC not work

Regards

Chris Bull

Subject: Re: Structures
Posted by David Fanning on Tue, 01 Mar 2005 21:52:45 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> Do named structures
> actually serve a useful purpose other letting you condense your syntax
> when creating them?

Well, you can count on them, as opposed to everything else
in IDL. I would have thought *that* would be worth something
to a C++ kind a guy. :-)

Cheers,

David

P.S. I've always found I learn more contemplating the vagrancies

of my tennis game than I do the vagrancies of IDL. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Structures
Posted by Michael Wallace on Tue, 01 Mar 2005 22:38:25 GMT
View Forum Message <> Reply to Message

>> Do named structures
>> actually serve a useful purpose other letting you condense your syntax
>> when creating them?
>
>
> Well, you can count on them, as opposed to everything else
> in IDL. I would have thought *that* would be worth something
> to a C++ kind a guy. :-)

It would if it weren't so strict about array sizes. At least with other languages, the concern is about the *type* of variable, rather than the type and size of the variable.

> P.S. I've always found I learn more contemplating the vagrancies
> of my tennis game than I do the vagrancies of IDL. :-)

You're probably right. I need to spend more time thinking about snooker than IDL. ;-)

-Mike

Subject: Re: Structures
Posted by Paul Van Delst[1] on Tue, 01 Mar 2005 22:39:50 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Michael Wallace writes:
>
>
>> Do named structures
>> actually serve a useful purpose other letting you condense your syntax
>> when creating them?
>
>

> Well, you can count on them, as opposed to everything else
> in IDL.

Heh heh. It's true though - all my routines that use named structures check them on input:

```
  ! -- Check that input is, indeed, a structure
  Type_Name = SIZE( MyStruct, /TNAME )
  IF ( STRUPCASE( Type_Name ) NE 'STRUCT' ) THEN $
    MESSAGE, 'Input is not a structure'

  ! -- Check that it's the right type of structure
  Structure_Name = TAG_NAMES( MyStruct, /STRUCTURE_NAME )
  IF ( STRUPCASE( Structure_Name ) NE 'MYSTRUCT' ) THEN $
    MESSAGE, 'Input is not a MyStruct structure'
```

But, then, I also do that for longs and floats sometimes too. I'm a belts and braces type of guy I guess. :o)

> I would have thought *that* would be worth something
> to a C++ kind a guy. :-)

To quote Michael from a previous post:

"I believe Java, C and sometimes C++ are the best languages to use when learning how to design software."

Given that IDL had its beginnings in <shock, gasp> Fortran, maybe that's why he doesn't grok how IDL handles structures. :o)

paulv

p.s. And, yes, I'm just pulling your whizzer Michael. :o)

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

Subject: Re: Structures
Posted by mperrin+news on Wed, 02 Mar 2005 00:21:02 GMT
View Forum Message <> Reply to Message

Michael Wallace <mwallace.no.spam@no.spam.swri.edu.invalid> wrote:
> I learned all this the hard way by trying to figure out why I kept
> getting errors when I'd attempt to use a different array size for some
> of my structure variables. I finally ran across this in the IDL

> documentation, but there wasn't any actual reason given for such
> draconian policy, especially when compared to the rest of the language.

I *think* it's something along these lines: We want to be able to
create arrays of structs, which (from a language designer's
viewpoint!) is much, much easier when all the structs are the same
size. If you let array sizes inside of structs vary, then you might
have foo1.somearray with 10 elements, and foo2.somearray with 100.
Then when the user does bar = [foo1,foo2], suddenly you're dealing
with an array of heterogenous data types and memory access becomes a
whole lot uglier and more inefficient. At that point, it's probably
easiest just to switch to Perl. ;-)

> It's really not a big deal since I can create anonymous structs
> everywhere instead of using named structures. Do named structures

You can also create named structures containing pointers to arrays, too.
This very problem is what finally pushed me over the potential barrier
into learning how to do pointers in IDL!

- Marshall

Subject: Re: Structures
Posted by Michael Wallace on Wed, 02 Mar 2005 00:40:31 GMT
View Forum Message <> Reply to Message

Marshall Perrin wrote:
> Michael Wallace <mwallace.no.spam@no.spam.swri.edu.invalid> wrote:
>
>> I learned all this the hard way by trying to figure out why I kept
>> getting errors when I'd attempt to use a different array size for some
>> of my structure variables. I finally ran across this in the IDL
>> documentation, but there wasn't any actual reason given for such
>> draconian policy, especially when compared to the rest of the language.
>
>
> I *think* it's something along these lines: We want to be able to
> create arrays of structs, which (from a language designer's
> viewpoint!) is much, much easier when all the structs are the same
> size. If you let array sizes inside of structs vary, then you might
> have foo1.somearray with 10 elements, and foo2.somearray with 100.
> Then when the user does bar = [foo1,foo2], suddenly you're dealing
> with an array of heterogenous data types and memory access becomes a
> whole lot uglier and more inefficient. At that point, it's probably
> easiest just to switch to Perl. ;-)
>

I can understand their decision if it had to do with memory access.
With static sizes you can easily index into an array of structures
rather that having to dynamically determine where the data is in memory.
  Actually it makes sense considering that IDL is array based.

> You can also create named structures containing pointers to arrays, too.
> This very problem is what finally pushed me over the potential barrier
> into learning how to do pointers in IDL!

Seems like I'm going down the same road you went. That little thought
about needing to use pointers in my structures has been trundling
through my mind the last couple days. :-)

-Mike

Subject: Re: Structures
Posted by David Fanning on Wed, 02 Mar 2005 00:49:53 GMT
View Forum Message <> Reply to Message

Michael Wallace writes:

> Seems like I'm going down the same road you went. That little thought
> about needing to use pointers in my structures has been trundling
> through my mind the last couple days. :-)

What I don't understand is how people can build
useful structures *without* using pointers. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: Structures
Posted by marc schellens[1] on Wed, 02 Mar 2005 03:43:32 GMT
View Forum Message <> Reply to Message

IDL> s={s,s:indgen(42)}
IDL> ss=[s,s,s,s]
IDL> help,ss[*].s

<Expression>   INT   = Array[42, 4]

Thats why they need to be fixed size.
Cheers,
marc

Subject: Re: Structures
Posted by Paul Van Delst[1] on Wed, 02 Mar 2005 15:33:46 GMT
View Forum Message <> Reply to Message

m_schellens@hotmail.com wrote:
> IDL> s={s,s:indgen(42)}
> IDL> ss=[s,s,s,s]
> IDL> help,ss[*].s
> <Expression>   INT   = Array[42, 4]
>
> Thats why they need to be fixed size.

Why's that? Doesn't it depend on what you want to do/how flexible you want the code to be?

IDL> s={s,s:ptr_new()}
IDL> ss=[s,s,s,s]
IDL> help, ss[*].s
<Expression>   POINTER   = Array[4]
IDL> ss[0].s=ptr_new(indgen(731))
IDL> ss[1].s=ptr_new(indgen(20))
IDL> ss[2].s=ptr_new(indgen(5))
IDL> ss[3].s=ptr_new(indgen(7))
IDL> help, *(ss[0].s)
<PtrHeapVar1>   INT   = Array[731]
IDL> help, *(ss[2].s)
<PtrHeapVar3>   INT   = Array[5]

Granted, the way you access all the bits and pieces has become more complex due to the pointer dereferencing, but you wouldn't do all this stuff on the command line anyway. (Right? Like objects. :o)

And, if I was doing it the way you suggested, I wouldn't access the data in ss[*].s as a rank-2 array, e.g.

IDL> help, (ss.s)[20,3]
<Expression>   INT   =   20

to get the 21st element from the 4th structure. I would reference it as

IDL> help, ss[3].s[20]
<Expression>   INT   =   20

My personal preference only of course. Doing it the first way seems sneaky

paulv

p.s. As an aside - in this context, this is another reason I like Fortran90/95 pointers. The dereferencing is implicit (i.e. no "*" out the front) so your accessing code doesn't have to change when you modify your structure to use allocatable pointers rather than fixed size arrays. E.g. in f95, ss(3)%s(20) gets the same info whether or not the "s" component is a fixed size array or a pointer in it's definition.

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

Subject: Re: Structures
Posted by Michael Wallace on Wed, 02 Mar 2005 16:59:42 GMT
View Forum Message <> Reply to Message

> To quote Michael from a previous post:
>
> "I believe Java, C and sometimes C++ are the best languages to use when
> learning how to design software."
>
> Given that IDL had its beginnings in <shock, gasp> Fortran, maybe that's
> why he doesn't grok how IDL handles structures. :o)

Don't you know that computer scientists are supposed to hate Fortran?!
I remember back in our "Theory of Programming Languages" or whatever the class was called, there were 10 rules for having a well-formed programming language, and Fortran broke about 7 or 8 of the rules!
Besides, even before that class, we had been conditioned to hate Fortran and make fun of the poor engineering students that had to endure the class. ;-)

-m

Subject: Re: Structures
Posted by Michael Wallace on Wed, 02 Mar 2005 17:04:36 GMT
View Forum Message <> Reply to Message

> p.s. As an aside - in this context, this is another reason I like
> Fortran90/95 pointers. The dereferencing is implicit (i.e. no "*" out
> the front) so your accessing code doesn't have to change when you modify

> your structure to use allocatable pointers rather than fixed size
> arrays. E.g. in f95, ss(3)%s(20) gets the same info whether or not the
> "s" component is a fixed size array or a pointer in it's definition.

Same is true of Java. Everything in Java, except the primitive types, are actually pointers, but those details are hidden from you. In Java it's even possible to define a multi-dimensional array such the length of the extra dimensions is variable. :-o

-m

Subject: Re: Structures
Posted by Paul Van Delst[1] on Wed, 02 Mar 2005 17:31:50 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
>> To quote Michael from a previous post:
>>
>> "I believe Java, C and sometimes C++ are the best languages to use
>> when learning how to design software."
>>
>> Given that IDL had its beginnings in <shock, gasp> Fortran, maybe
>> that's why he doesn't grok how IDL handles structures. :o)
>
>
> Don't you know that computer scientists are supposed to hate Fortran?! I
> remember back in our "Theory of Programming Languages" or whatever the
> class was called, there were 10 rules for having a well-formed
> programming language, and Fortran broke about 7 or 8 of the rules!
> Besides, even before that class, we had been conditioned to hate Fortran
> and make fun of the poor engineering students that had to endure the
> class. ;-)

Well, it does depend on your frame of reference. Given the time, and inclination, I could come up with similar studies (i.e. does the theoretical well-formed-ness of a particular language translate to increased productivity, lower incidience of bugs etc) that show Fortran to be better in that respect than more popular languages (e.g. C, C++ etc). I believe the DoD did a lot of those sorts of studies in the 70's (probably earlier too) and that the development of Ada is somehow intertwined with that effort.[*]

I just finished reading the "Mismeasure of Man" by SJGould for me bookclub, so I'm newly aware of how instilled prejudices can unconsciously affect people's decisions and general outlook. With regards to your programming class, the question that first pops to mind is who made up the rules? And for what applications? I could come up with 10 rules for a well-formed programming language that Fortran statifies but other, more popular, languages such as C/C++/Java/etc don't.

As you can probably tell, I'm still smarting from the disdain of the lecturer of my "Programming in Fortran 251" class back in '86 (or '87?). He was a computer scientist type and his dislike of Fortran and having to teach it was palpable. (Poor poor me... :o)

I'm going to retreat to the fartherest corner of my cubicle and sulk for a bit.... :oD

cheers,

paulv

[*] A bit of googling turned up:
[DoD 1978] U.S. Department of Defense. June 1978. ``Department Of Defense Requirements for High Order Computer Programming Languages: "Steelman"''

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

Subject: Re: Structures
Posted by Benjamin Hornberger on Wed, 02 Mar 2005 17:33:56 GMT
View Forum Message <> Reply to Message

Michael Wallace wrote:
> remember back in our "Theory of Programming Languages" or whatever the
> class was called, there were 10 rules for having a well-formed
> programming language, and Fortran broke about 7 or 8 of the rules!

How many rules does IDL break then?

Subject: Re: Structures
Posted by David Fanning on Wed, 02 Mar 2005 17:36:31 GMT
View Forum Message <> Reply to Message

Benjamin Hornberger writes:

> How many rules does IDL break then?

IDL always strives to *make* the rules! :-)

Cheers,

David

--
David Fanning, Ph.D.

Subject: Re: Structures
Posted by marc schellens[1] on Thu, 03 Mar 2005 03:14:26 GMT
View Forum Message <> Reply to Message

Paul,

I didn't comment on any particular programming style or whatever,
but with the expression I showed, which is valid in IDL it is clear
there
is no other way than that structures must have fixed size fields.
Otherwise the expression could not be valid or at least only with some
weired rules (like pad to largest, clip to shortest, fixed only if
array member...).
My point was that IDL could not have been made to accept variable size
arrays in structures without being then more restricted on another side
(which answeres Michaels question I guess).

Cheers,
marc