Subject: Arrays of Structures
Posted by Ben Tupper on Wed, 21 Jun 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Hello,

I have a 'how come?' question.

How come I can make an array of named structures

but not of anonymous structures?

Thanks,

Ben

Ben Tupper

Bigelow Laboratory for Ocean Science
tupper@seadas.bigelow.org

pemaquidriver@tidewater.net

Subject: Re: Arrays of Structures
Posted by Brian Larsen on Thu, 08 Feb 2007 17:41:48 GMT
View Forum Message <> Reply to Message

Mick,

you just need to dig a little deeper with help for the answer.

This is what you did and it is true this is an array with 50 elements
IDL> HELP, structs.a
<Expression> INT = Array[50]

but if you look at structs by itself not structs.a, it is the array,
NOT structs.a

IDL> HELP, structs
STRUCTS STRUCT = -> <Anonymous> Array[50]

So what is inside the structure?

Here a is inside the struct and it is an int

IDL> HELP, structs, /str
** Structure <84bd40c>, 1 tags, length=2, data length=2, refs=1:
   A   INT   1

Meaning that structs.a[23] doesn't make sense because structs.a is an
int. While structs[23].a does make sense because structs is an array.

This is just one of those lessons that take a while to get a hold of.

Make any sense?

Brian

-- ---------
Brian A. Larsen
Dept. of Physics
Space Science and Engineering Lab (SSEL)
Montana State University - Bozeman
Bozeman, MT 59717

On Feb 8, 10:03 am, "Mick Brooks" <mick.bro...@gmail.com> wrote:
> Hi,
>
> Can anyone help my understanding of what happens when I apply a tag-
> name to an array of structures? All of my previous questions have been
> answered by searching on eitherwww.dfanning.comor this newsgroup,
> but this one has me stumped.
>
> Take an array of (very boring, anonymous) structures:
>
> IDL> structs = REPLICATE({a:1}, 50)
>
> If I look-up a tag-name on this array, I see this:
>
> IDL> HELP, structs.a
> <Expression>   INT   = Array[50]
>
> That looks like wonderful magic to me (I don't have much experience
> with array-based languages) - IDL knew to apply the tag-name to each
> structure in turn, and return me an array of the values - in this case
> an array of INTs. Now what if I want the 3rd element of this array?
> Let's try:
>
> IDL> HELP, structs.a[2]
> % Subscript range values of the form low:high must be >= 0, < size,

> with low
> <= high: <No name>.
> % Execution halted at: $MAIN$
>
> Not so good. I know of two workarounds for this. Either take the 3rd
> element of the array of structures before looking up the tag-name (I
> understand why this one works):
>
> IDL> HELP, structs[2].a
> <Expression> INT = 1
>
> Or, put some extra parentheses in (I've no idea why this one fixes
> it):
>
> IDL> HELP, (structs.a)[2]
> <Expression> INT = 1
>
> Why does my first attempt fails, and why do the parentheses help?
>
> For more confusion, look what happens if I try and lookup the 2nd, 4th
> and 28th element all at once:
>
> IDL> indices = [1, 3, 27]
> IDL> HELP, structs.a[indices]
> <Expression> INT = Array[3, 50]
>
> Where did that extra dimension come from? What is the type of
> structs.a? It seems that if I don't put parentheses around it, some of
> the magic leaks out...
>
> Cheers,
>
> Mick Brooks

Subject: Re: Arrays of Structures
Posted by Mick Brooks on Thu, 08 Feb 2007 18:04:46 GMT
View Forum Message <> Reply to Message

On Feb 8, 5:41 pm, "Brian Larsen" <balar...@gmail.com> wrote:

> This is what you did and it is true this is an array with 50 elements
> IDL> HELP, structs.a
> <Expression> INT = Array[50]
>
> but if you look at structs by itself not structs.a, it is the array,
> NOT structs.a
>

> IDL> HELP, structs
> STRUCTS STRUCT = -> <Anonymous> Array[50]
>
> So what is inside the structure?
>
> Here a is inside the struct and it is an int
>
> IDL> HELP, structs, /str
> ** Structure <84bd40c>, 1 tags, length=2, data length=2, refs=1:
> A INT 1

(I think) I'm following you to here...

> Meaning that structs.a[23] doesn't make sense because structs.a is an
> int.

... but this is where you lose me. Doesn't the the first line that you showed say that structs.a is an array of 50 ints?
Maybe I just can't read the output of HELP properly, but struts.a certainly sometimes behaves as an array. Try PRINTing it, for instance.
Or let's compare the output of HELP on a real array of 50 ints:

IDL> HELP, structs.a
<Expression> INT = Array[50]
IDL> HELP, indgen(50)
<Expression> INT = Array[50]

> While structs[23].a does make sense because structs is an array.

Yes, I'm happy with why this is right, but still not clear why structs.a[23] is wrong.

> This is just one of those lessons that take a while to get a hold of.
>
> Make any sense?

Not really, but thanks for trying.

Cheers,

Mick

Subject: Re: Arrays of Structures
Posted by Brian Larsen on Thu, 08 Feb 2007 18:59:56 GMT
View Forum Message <> Reply to Message

Mick,

>
> (I think) I'm following you to here...
>
>> Meaning that structs.a[23] doesn't make sense because structs.a is an
>> int.
>
> ... but this is where you lose me. Doesn't the the first line that you
> showed say that structs.a is an array of 50 ints?
> Maybe I just can't read the output of HELP properly, but struts.a
> certainly sometimes behaves as an array. Try PRINTing it, for
> instance.
> Or let's compare the output of HELP on a real array of 50 ints:

True, this is a little odd. I like to think of structures as a
container holding things like a. In this case a is an integer and you
have an array of the containers. The other way to do it would be
IDL> structs = {a:intarr(50)}
IDL> help, structs
STRUCTS STRUCT = -> <Anonymous> Array[1]
IDL> help, structs, /str
** Structure <84bd4a4>, 1 tags, length=100, data length=100, refs=1:
A INT Array[50]

Where the thing in the container, a, is an integer array and you only
have one container.

The way that I use structures on a day to day basis is that I read in
a satellite data file that has a bunch of fields in certain coordinate
systems, I store all that in a structure, I then call routines to
change around the coordinate systems and add the parameters to the
structure in that routine so that the container now has more stuff in
it. Then I repeat this for lots of things. So the original structure
is not an array but has arrays and scalars in it, then if I want more
files I read them into the same structure making the structure an
array, that still contains scalars and vectors.

IDL> dat=read_sim('2001101131206.sim')
IDL> help, dat
DAT STRUCT = -> <Anonymous> Array[1]
IDL> help, dat, /str
** Structure <82731ec>, 7 tags, length=301112, data length=301108,
refs=1:
THETA FLOAT Array[12540]
R FLOAT Array[12540]
INTENSITY FLOAT Array[12540]
FIELD4 FLOAT Array[12540]

```
  HEADER          STRUCT    -> <Anonymous> Array[1]
  X               FLOAT     Array[12540]
  Y               FLOAT     Array[12540]
```

Of course to make it more confusing still there is a structure header
inside my structure dat

```
IDL> help, dat.header
<Expression>    STRUCT    = -> <Anonymous> Array[1]
IDL> help, dat.header, /str
** Structure <84d4ccc>, 21 tags, length=152, data length=148, refs=2:
   TIME            STRING    '2001101131206'
   FORMAT          STRING    'PT'
   NITER           INT              6
   SDEV            FLOAT         0.490877
   XSTART          FLOAT         -3.14159
   XEND            FLOAT          3.14159
   YSTART          FLOAT          1.00000
   YEND            FLOAT          7.00000
   XLENGTH         INT            209
   YLENGTH         INT             60
   XSTORAGE        STRING    'POINT'
   YSTORAGE        STRING    'POINT'
   GSTORAGE        STRING    'ROW'
   ZEROS           STRING    'IGNORE'
   ORDER           STRING    'NEW'
   UNFILLED        INT             0
   BAD             FLOAT         -5.00000
   GMIN            FLOAT          0.00000
   GMAX            FLOAT       1.00000e+30
   XCYCLIC         STRING    'YES'
   YCYCLIC         STRING    'NO'
```

So I print those elements like this:
```
IDL> print, dat.header.time
2001101131206
```

We'll get to the bottom of this.

Brian

---------- ---------
Brian A. Larsen
Dept. of Physics
Space Science and Engineering Lab (SSEL)
Montana State University - Bozeman

Subject: Re: Arrays of Structures
Posted by Paul Van Delst[1] on Thu, 08 Feb 2007 19:02:08 GMT
View Forum Message <> Reply to Message

Mick Brooks wrote:
> On Feb 8, 5:41 pm, "Brian Larsen" <balar...@gmail.com> wrote:
>
>> This is what you did and it is true this is an array with 50 elements
>> IDL> HELP, structs.a
>> <Expression> INT = Array[50]
>>
>> but if you look at structs by itself not structs.a, it is the array,
>> NOT structs.a
>>
>> IDL> HELP, structs
>> STRUCTS STRUCT = -> <Anonymous> Array[50]
>>
>> So what is inside the structure?
>>
>> Here a is inside the struct and it is an int
>>
>> IDL> HELP, structs, /str
>> ** Structure <84bd40c>, 1 tags, length=2, data length=2, refs=1:
>> A INT 1
>
> (I think) I'm following you to here...
>
>> Meaning that structs.a[23] doesn't make sense because structs.a is an
>> int.
>
> ... but this is where you lose me. Doesn't the the first line that you
> showed say that structs.a is an array of 50 ints?
> Maybe I just can't read the output of HELP properly, but struts.a
> certainly sometimes behaves as an array. Try PRINTing it, for
> instance.
> Or let's compare the output of HELP on a real array of 50 ints:
>
> IDL> HELP, structs.a
> <Expression> INT = Array[50]
> IDL> HELP, indgen(50)
> <Expression> INT = Array[50]
>
>> While structs[23].a does make sense because structs is an array.
>
> Yes, I'm happy with why this is right, but still not clear why

> structs.a[23] is wrong.

You know, I'm confused now too. Check out the precedence in the IDL help:

Table 12-9: Operator Precedence

Priority Operator
First (highest) ( ) (parentheses, to group expressions)
[ ] (brackets, to concatenate arrays)
Second . (structure field dereference)
[ ] (brackets, to subscript an array)
( ) (parentheses, used in a function call)
etc...

So, you see that the the structure field dereference operator, ".", and the array subscript operator, "[]" , have the same precedence. Operators with equal precedence are evaluated from left to right.

So,

structs.a[23]

means FIRST dereference the structure field (structs.a), THEN index the array ([23]). The way I see it,

structs.a[23]
and
(structs.a)[23]

should be equivalent.

Maybe the weirdness has something to do with where the result of the dereference "goes" ? Hence the "<No name>" in the error message?

Hmm.

paulv

--
Paul van Delst Ride lots.
CIMSS @ NOAA/NCEP/EMC Eddy Merckx

Subject: Re: Arrays of Structures
Posted by news.qwest.net on Thu, 08 Feb 2007 19:09:58 GMT
View Forum Message <> Reply to Message

"Mick Brooks" <mick.brooks@gmail.com> wrote in message

news:1170957886.387622.208430@l53g2000cwa.googlegroups.com.. .
...
> ... but this is where you lose me. Doesn't the the first line that you
> showed say that structs.a is an array of 50 ints?

Not exactly. The key point is that structs.a is an EXPRESSION.
Also, "struct" is an array of structures.

You cannot do
IDL> EXPRESSION(10)
, you must first cast the expression
IDL> (EXPRESSION)[10]

It is not an operator precedence thing.

Cheers,
bob

Subject: Re: Arrays of Structures
Posted by Brian Larsen on Thu, 08 Feb 2007 19:11:54 GMT
View Forum Message <> Reply to Message

Paulv,

I think this is really just a wholly bad idea to do that at all. The
reason is what about this structure?
IDL> dat=replicate({a:findgen(50)}, 50)
IDL> help, dat
DAT STRUCT = -> <Anonymous> Array[50]
IDL> help, dat, /str
** Structure <84ccb14>, 1 tags, length=200, data length=200, refs=1:
A FLOAT Array[50]

Here the difference between dat[23].a and dat.a[23] is really
obvious.

IDL> print, dat[23].a
0.00000 1.00000 2.00000 3.00000 4.00000
5.00000 6.00000 7.00000 8.00000
9.00000 10.0000 11.0000 12.0000 13.0000
14.0000 15.0000 16.0000 17.0000
18.0000 19.0000 20.0000 21.0000 22.0000
23.0000 24.0000 25.0000 26.0000
27.0000 28.0000 29.0000 30.0000 31.0000
32.0000 33.0000 34.0000 35.0000
36.0000 37.0000 38.0000 39.0000 40.0000

41.0000 42.0000 43.0000 44.0000
45.0000 46.0000 47.0000 48.0000 49.0000
IDL> print, dat.a[23]
23.0000 23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000 23.0000
23.0000 23.0000 23.0000 23.0000

Be sure to just use the structure correctly and put the [ ] on the
thing that is an array whether that is the structure or the thing in
the structure (all the dereferencing stuff is true but tends to just
confuse the issue in my opinion)

Brian

Brian A. Larsen
Dept. of Physics
Space Science and Engineering Lab (SSEL)
Montana State University - Bozeman
Bozeman, MT 59717

On Feb 8, 12:02 pm, Paul van Delst <Paul.vanDe...@noaa.gov> wrote:
> Mick Brooks wrote:
>> On Feb 8, 5:41 pm, "Brian Larsen" <balar...@gmail.com> wrote:
>
>>> This is what you did and it is true this is an array with 50 elements
>>> IDL> HELP, structs.a
>>> <Expression> INT = Array[50]
>
>>> but if you look at structs by itself not structs.a, it is the array,
>>> NOT structs.a
>
>>> IDL> HELP, structs
>>> STRUCTS STRUCT = -> <Anonymous> Array[50]
>
>>> So what is inside the structure?
>
>>> Here a is inside the struct and it is an int
>

>>> IDL> HELP, structs, /str
>>> ** Structure <84bd40c>, 1 tags, length=2, data length=2, refs=1:
>>>   A   INT   1
>
>> (I think) I'm following you to here...
>
>>> Meaning that structs.a[23] doesn't make sense because structs.a is an
>>> int.
>
>> ... but this is where you lose me. Doesn't the the first line that you
>> showed say that structs.a is an array of 50 ints?
>> Maybe I just can't read the output of HELP properly, but struts.a
>> certainly sometimes behaves as an array. Try PRINTing it, for
>> instance.
>> Or let's compare the output of HELP on a real array of 50 ints:
>
>> IDL> HELP, structs.a
>> <Expression>   INT   = Array[50]
>> IDL> HELP, indgen(50)
>> <Expression>   INT   = Array[50]
>
>>> While structs[23].a does make sense because structs is an array.
>
>> Yes, I'm happy with why this is right, but still not clear why
>> structs.a[23] is wrong.
>
> You know, I'm confused now too. Check out the precedence in the IDL help:
>
> Table 12-9: Operator Precedence
>
> Priority   Operator
> First (highest) ( ) (parentheses, to group expressions)
>   [ ] (brackets, to concatenate arrays)
> Second   . (structure field dereference)
>   [ ] (brackets, to subscript an array)
>   ( ) (parentheses, used in a function call)
> etc...
>
> So, you see that the the structure field dereference operator, ".", and the array
> subscript operator, "[]" , have the same precedence. Operators with equal precedence are
> evaluated from left to right.
>
> So,
>
>   structs.a[23]
>
> means FIRST dereference the structure field (structs.a), THEN index the array ([23]). The
> way I see it,

>
> structs.a[23]
> and
> (structs.a)[23]
>
> should be equivalent.
>
> Maybe the weirdness has something to do with where the result of the dereference "goes" ?
> Hence the "<No name>" in the error message?
>
> Hmm.
>
> paulv
>
> --
> Paul van Delst Ride lots.
> CIMSS @ NOAA/NCEP/EMC Eddy Merckx

Subject: Re: Arrays of Structures
Posted by Mick Brooks on Thu, 08 Feb 2007 19:27:48 GMT
View Forum Message <> Reply to Message

On Feb 8, 7:09 pm, "R.G. Stockwell" <n...@email.please> wrote:

>> ... but this is where you lose me. Doesn't the the first line that you
>> showed say that structs.a is an array of 50 ints?
>
> Not exactly. The key point is that structs.a is an EXPRESSION.
> Also, "struct" is an array of structures.

Yes, this struck me as I drove home from work. I don't have IDL here,
but will be trying
IDL> INDGEN(50)[23]
first thing in the morning

> You cannot do
> IDL> EXPRESSION(10)
> , you must first cast the expression
> IDL> (EXPRESSION)[10]
>
> It is not an operator precedence thing.

Yes, I'd looked at the precedence table and made the same conclusion
as you and Paulv.

All that's left is to explain why we get the extra leading dimension
when subscripting an expression-which-evaluates-to-an-array-of-ints

thing (i.e. structs.a) with multiple indices (or [*]).
This was the actual question I was asked by a user of the module I'd written - he was seeing these extra leading dimensions, and queried whether my structure was somehow broken. I couldn't answer, and so came up with the simpler case we're discussing.

Thanks for all your help,

Mick

Subject: Re: Arrays of Structures
Posted by Paul Van Delst[1] on Thu, 08 Feb 2007 19:32:20 GMT
View Forum Message <> Reply to Message

R.G. Stockwell wrote:
> "Mick Brooks" <mick.brooks@gmail.com> wrote in message
> news:1170957886.387622.208430@l53g2000cwa.googlegroups.com.. .
> ...
>> ... but this is where you lose me. Doesn't the the first line that you
>> showed say that structs.a is an array of 50 ints?
>
> Not exactly. The key point is that structs.a is an EXPRESSION.
> Also, "struct" is an array of structures.
>
> You cannot do
> IDL> EXPRESSION(10)
> , you must first cast the expression
> IDL> (EXPRESSION)[10]
>
>
> It is not an operator precedence thing.

Ah. I knew it would be something simple I hadn't considered -- seems to be happening a lot these days... :o(

paulv

--
Paul van Delst Ride lots.
CIMSS @ NOAA/NCEP/EMC Eddy Merckx

Subject: Re: Arrays of Structures
Posted by news.qwest.net on Thu, 08 Feb 2007 19:44:48 GMT
View Forum Message <> Reply to Message

"Mick Brooks" <mick.brooks@gmail.com> wrote in message
news:1170962868.330264.257780@q2g2000cwa.googlegroups.com...
> On Feb 8, 7:09 pm, "R.G. Stockwell" <n...@email.please> wrote:
>
>>> ... but this is where you lose me. Doesn't the the first line that you
>>> showed say that structs.a is an array of 50 ints?
>>
>> Not exactly. The key point is that structs.a is an EXPRESSION.
>> Also, "struct" is an array of structures.
>
> Yes, this struck me as I drove home from work. I don't have IDL here,
> but will be trying
> IDL>print, INDGEN(50)[23]

syntax error

> All that's left is to explain why we get the extra leading dimension
> when subscripting an expression-which-evaluates-to-an-array-of-ints
> thing (i.e. structs.a) with multiple indices (or [*]).

ah yes, the extra leading dimension. That is because IDL hates you (lol).

Cheers,
bob

Subject: Re: Arrays of Structures
Posted by Michael Galloy on Thu, 08 Feb 2007 20:21:16 GMT
View Forum Message <> Reply to Message

On Feb 8, 12:09 pm, "R.G. Stockwell" <n...@email.please> wrote:
> "Mick Brooks" <mick.bro...@gmail.com> wrote in message
>
> news:1170957886.387622.208430@l53g2000cwa.googlegroups.com.. .
> ...
>
>> ... but this is where you lose me. Doesn't the the first line that you
>> showed say that structs.a is an array of 50 ints?
>
> Not exactly. The key point is that structs.a is an EXPRESSION.
> Also, "struct" is an array of structures.
>
> You cannot do
> IDL> EXPRESSION(10)
> , you must first cast the expression
> IDL> (EXPRESSION)[10]
>

> It is not an operator precedence thing.
>
> Cheers,
> bob

I'm not sure that is an expression thing either. When I try to index an expression, I get a syntax error:

```
IDL> print, findgen(10)[5]

print, findgen(10)[5]
                  ^
% Syntax error.
```

which can be fixed by using parenthesis:

```
IDL> print, (findgen(10))[5]
      5.00000
```

But Mick's error is giving the "out of bounds index" error. Using parenthesis is not fixing a syntax error -- it's indexing something else.

More poking around with a slight variation of Brian's weird example:

```
IDL> dat2 = replicate({a:findgen(25)}, 50)
IDL> help, dat2
DAT2            STRUCT    = -> <Anonymous> Array[50]
IDL> help, dat2, /structures
** Structure <2614a44>, 1 tags, length=100, data length=100, refs=1:
   A              FLOAT     Array[25]
IDL> help, dat2.a
<Expression>    FLOAT     = Array[25, 50]
```

OK, now try the unholy notation:

```
IDL> help, dat2.a[24]
<Expression>    FLOAT     = Array[50]
IDL> print, dat2.a[24]
      24.0000      24.0000 ... (50 times)
IDL> help, dat2.a[25]
% Subscript range values of the form low:high must be >= 0, < size,
with low <= high: <No name>.
% Execution halted at: $MAIN$
```

I think that the only way to interpret this is that the structure dat2 is being dereferenced by a[24] (although I agree it seems inconsistent with the precedence rules). Check out the example of:

CONTOUR, cat[5:50].inten[2:8]

in the online help:

http://idlastro.gsfc.nasa.gov/idl_html_help/
Arrays_of_Structures.html

Also,

On Feb 8, 10:03 am, "Mick Brooks" <mick.bro...@gmail.com> wrote:
> IDL> indices = [1, 3, 27]
> IDL> HELP, structs.a[indices]
> <Expression> INT = Array[3, 50]

> Where did that extra dimension come from?

Using the logic from above, I think the same place as:

```
IDL> k = 1
IDL> print, k[0]
      1
IDL> print, k[1]
% Attempt to subscript K with <INT  (  1)> is out of range.
% Execution halted at: $MAIN$
IDL> print, k[[1]]
      1
IDL> print, k[[1, 2, 3]]
      1      1      1
IDL> help, k[[1, 2, 3]]
<Expression>  INT  = Array[3]
```

Mike
--
www.michaelgalloy.com

Subject: Re: Arrays of Structures
Posted by JD Smith on Thu, 08 Feb 2007 20:42:48 GMT
View Forum Message <> Reply to Message

On Thu, 08 Feb 2007 14:02:08 -0500, Paul van Delst wrote:

> Mick Brooks wrote:

> You know, I'm confused now too. Check out the precedence in the IDL help:

> So, you see that the the structure field dereference operator, ".", and the array

> subscript operator, "[]" , have the same precedence. Operators with equal precedence are
> evaluated from left to right.
>
> So,
>
>   structs.a[23]
>
> means FIRST dereference the structure field (structs.a), THEN index the array ([23]). The
> way I see it,
>
>   structs.a[23]
> and
>   (structs.a)[23]
>
> should be equivalent.

The problem is that structs.a is not an array until the ".a" part has been applied. What if structs had been;

structs={a:indgen(25)}

what should structs.a[23] do then, and how should IDL know the difference. Until IDL knows what shape "structs.a" will have, it cannot make any informed decision about how to index it. You might object, saying that IDL should just evaluate that to begin with always, but think how expensive that is. Creating "structs.a" would cause a large temporary array to be created, only to finally index a single element. Compare the memory usage of the following:

```
IDL> struct=replicate({a:lindgen(100,100,100)},100)
IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
     382.472MB
IDL> val=struct[10].a[4]
IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
     0.00000MB extra
IDL> val2=(struct.a)[10,4]
IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
     381.470MB extra
```

So the latter method first creates a temporary variable of size 100,100,100,100, and then pulls a single element out of it. Not exactly good form.

JD

Subject: Re: Arrays of Structures

<mgalloy@gmail.com> wrote in message
news:1170966075.702031.236230@k78g2000cwa.googlegroups.com.. .
> On Feb 8, 12:09 pm, "R.G. Stockwell" <n...@email.please> wrote:
>> "Mick Brooks" <mick.bro...@gmail.com> wrote in message
>>
>> news:1170957886.387622.208430@l53g2000cwa.googlegroups.com.. .
...
> I'm not sure that is an expression thing either. When I try to index
> an expression, I get a syntax error:
>
> IDL> print, findgen(10)[5]
>
> print, findgen(10)[5]
>   ^
> % Syntax error.

Right. This is a syntax error. IDL does not know how to parse it.
The difference is that struct.a[5] is not a syntax error, idl does know
and must know how to parse that expression. In this case though, it
is trying to access the 5th element of a, which is out of bounds.
Note that the following works

IDL> help,struct.a[0]
<Expression> INT = Array[50]

(and the rest of your post made sense to me.)

The point is, that struct is an array. If you want to acces
the array elements you must do
IDL> structs[*].a.
Or IDL lets you cast the expression into a temporary array as follows:
IDL> help, (struct.a)[22]
<Expression

IDL> help,junk.a

<Expression> INT = Array[50]

IDL> help,(junk.a)

<Expression> INT = Array[50]

> INT = 1

The original point I made is that you cannot dereference an expression, you

have to have parenthesis on it.

Cheers,
bob

Subject: Re: Arrays of Structures
Posted by Mick Brooks on Fri, 09 Feb 2007 10:13:56 GMT
View Forum Message <> Reply to Message

On Feb 8, 8:42 pm, JD Smith <jdsm...@as.arizona.edu> wrote:

> The problem is that structs.a is not an array until the ".a" part has been
> applied.

<snip>

> Creating "structs.a" would cause a large temporary
> array to be created, only to finally index a single element. Compare the
> memory usage of the following:
>
> IDL> struct=replicate({a:lindgen(100,100,100)},100)
> IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
> 382.472MB
> IDL> val=struct[10].a[4]
> IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
> 0.00000MB extra
> IDL> val2=(struct.a)[10,4]
> IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
> 381.470MB extra
>
> So the latter method first creates a temporary variable of size
> 100,100,100,100, and then pulls a single element out of it. Not
> exactly good form.

It seems that this is a reason to prefer my first "workaround" to my
second one, but it doesn't tell us anything about my problem ("unholy
notation" - I like that), which here would be
represented by val3=struct.a[10,4] i.e. leaving off the temporary-
creating parentheses.

If I try, I get the following:

IDL> struct=replicate({a:lindgen(100,100,100)},100)
IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"

```
      385.836MB
IDL> val=struct[10].a[4]
IDL> print,((n=memory(/HIGHWATER))-m)/1024/1024.,"MB Extra"
     -3.81445MB Extra
IDL> val3=struct.a[10,4]
IDL> print,(memory(/HIGHWATER)-n)/1024/1024.,"MB Extra"
      0.00000MB Extra
IDL> HELP, val, val3
VAL             LONG      =           4
VAL3            LONG      = Array[100]
IDL> PRINT, val3
         410  [+ another 99 of the same]
```

The problem case doesn't use any extra memory (great!), but it gives a different result (boo!).

Bob's and Mike's posts made me think that my original "structs.a" has a leading shallow dimension, but that IDL elides it when evaluating it.
So,

```
IDL> structs = replicate({a:1},50)
IDL> HELP, structs.a[0]
<Expression>    INT       = Array[50]
```

works, because our array subscript is within bounds on our leading shallow dimension, but

```
IDL> HELP, structs.a[1]
% Subscript range values of the form low:high must be >= 0, < size,
with low
  <= high: <No name>.
% Execution halted at: $MAIN$
```

is out of range.

If we ask for everything from the array that is structs.a

```
IDL> HELP, structs.a[*]
<Expression>    INT       = Array[1, 50]
```

we see the entire thing, leading shallow dimension included.

However, if we simply evaluate structs.a, IDL drops the leading dimension, like so:

```
IDL> HELP, structs.a
<Expression>    INT       = Array[50]
```

Creating a temporary with parentheses also causes IDL to drop the leading dimension too:

```
IDL> HELP, (structs.a)[*]
```

<Expression>   INT   = Array[50]

Does this make any more sense to anyone? I still can't use this idea
to work out what's going on
with val3 above though...

Thanks again for everybody's help,

Mick

Subject: Re: Arrays of Structures
Posted by Paul Van Delst[1] on Fri, 09 Feb 2007 16:01:24 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
> On Thu, 08 Feb 2007 14:02:08 -0500, Paul van Delst wrote:
>
>> Mick Brooks wrote:
>
>> You know, I'm confused now too. Check out the precedence in the IDL help:
>
>> So, you see that the the structure field dereference operator, ".", and the array
>> subscript operator, "[]" , have the same precedence. Operators with equal precedence are
>> evaluated from left to right.
>>
>> So,
>>
>>   structs.a[23]
>>
>> means FIRST dereference the structure field (structs.a), THEN index the array ([23]). The
>> way I see it,
>>
>>   structs.a[23]
>> and
>>   (structs.a)[23]
>>
>> should be equivalent.
>
> The problem is that structs.a is not an array until the ".a" part has been
> applied. What if structs had been;
>
> structs={a:indgen(25)}
>
> what should structs.a[23] do then, and how should IDL know the difference.
> Until IDL knows what shape "structs.a" will have, it cannot make any
> informed decision about how to index it.

How is it different from:

```
IDL> x=findgen(10,20)
IDL> help, x
X               FLOAT     = Array[10, 20]
IDL> help, x[23]
<Expression>    FLOAT     =       23.0000
IDL> help, (x)[23]
<Expression>    FLOAT     =       23.0000
```

?

Or

```
IDL> x=findgen(10,20,30)
IDL> help,x[1000]
<Expression>    FLOAT     =       1000.00
IDL> help,x[0,0,5]
<Expression>    FLOAT     =       1000.00
```

?

The reference [23] is being applied to an array. The rank of that array doesn't matter since IDL allows you to reference multi-rank arrays with a single index (treating it as a "flat" array).

If we have

```
IDL> structs=replicate({a:indgen(25)},10)
IDL> help, structs
STRUCTS         STRUCT    = -> <Anonymous> Array[10]
IDL> help, structs.a
<Expression>    INT       = Array[25, 10]
```

Then, via the precedence rules,

```
IDL> help, structs.a[23]
```

should be equivalent to

```
IDL> help, structs[0].a[23]
```

> You might object, saying that
> IDL should just evaluate that to begin with always, but think how
> expensive that is. Creating "structs.a" would cause a large temporary
> array to be created, only to finally index a single element.

Yes, I agree that is bad but it is an implementation detail. The precedence rules seem quite clear that this should work. Please correct me if my interpretation of the rules is

wrong.

Maybe it was disallowed in general so they (RSI) wouldn't have to special case structures containing pointers,

```
IDL> structs=replicate({a:ptr_new(/allocate_heap)},10)
IDL> *structs[0].a=findgen(20)
IDL> *structs[1].a=indgen(74)
IDL> *structs[2].a=dindgen(3)
IDL> *structs[4].a=sindgen(13)
IDL> help, *structs.a[4]
% Subscript range values of the form low:high must be >= 0, < size, with low <= high: <No
name>.
% Execution halted at: $MAIN$
```

Although, again, the rules indicate that the pointer dereference operator "*" has a lower precedence that either "." or "[]" so I would contend that even the above

```
IDL> help, *structs.a[4]
% Subscript range values of the form low:high must be >= 0, < size, with low <= high: <No
name>.
% Execution halted at: $MAIN$
```

is valid and that parentheses should not be required,

```
IDL> help, *(structs.a)[4]
<PtrHeapVar1>   STRING    = Array[13]
```

I also think the case of

```
IDL> help, *structs.a[23]
```

is also well defined.... it's invalid (at least it is until IDL allows arrays composed of different different objects).

I reckon these sorts of special cases are why the more general case is disallowed. (But what the hell do I know! :o)

```
>  Compare the
>  memory usage of the following:
>
>  IDL> struct=replicate({a:lindgen(100,100,100)},100)
>  IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
>        382.472MB
>  IDL> val=struct[10].a[4]
>  IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
```

> 0.00000MB extra
> IDL> val2=(struct.a)[10,4]
> IDL> print,(memory(/HIGHWATER)-m)/1024/1024.,"MB extra"
> 381.470MB extra
>
> So the latter method first creates a temporary variable of size
> 100,100,100,100, and then pulls a single element out of it. Not
> exactly good form.

No, I agree, not good. But the precedence rules for operators not being followed isn't too crash hot either. :o)

cheers,

paulv

--
Paul van Delst Ride lots.
CIMSS @ NOAA/NCEP/EMC Eddy Merckx

Subject: Re: Arrays of Structures
Posted by JD Smith on Fri, 09 Feb 2007 20:51:50 GMT
View Forum Message <> Reply to Message

On Fri, 09 Feb 2007 02:13:56 -0800, Mick Brooks wrote:

> If I try, I get the following:
>
> IDL> struct=replicate({a:lindgen(100,100,100)},100)
> IDL> print,(m=memory(/HIGHWATER))/1024/1024.,"MB"
> 385.836MB
> IDL> val=struct[10].a[4]
> IDL> print,((n=memory(/HIGHWATER))-m)/1024/1024.,"MB Extra"
> -3.81445MB Extra
> IDL> val3=struct.a[10,4]
> IDL> print,(memory(/HIGHWATER)-n)/1024/1024.,"MB Extra"
> 0.00000MB Extra
> IDL> HELP, val, val3
> VAL LONG = 4
> VAL3 LONG = Array[100]
> IDL> PRINT, val3
> 410 [+ another 99 of the same]
>
> The problem case doesn't use any extra memory (great!), but it gives a
> different result (boo!).

I probably should have used a better example, that was confusing. The

reason val3 "works" is because "a", the field being de-referenced, already has 3 dimensions. So it grabs the [10,4] element of all a's, and then proceeds to thread that across all 100 structures in the structure array, creating a new array in the process. Here's a better example illustrating this issue:

```
IDL> val4=(struct.a)[10,4,1,1]
IDL> val5=struct[10].a[4,1,1]
IDL> print,val4,val5
      10410       10104
IDL> val6=struct.a[10,4,1,1]
% Subscript range values of the form low:high must be >= 0, < size, with low
  <= high: <No name>.
```

Here's the real issue: since "struct.a" doesn't exist as an array anywhere in memory, but instead is a composite entity, it must be formed anew by:

a) allocating a new array of enough memory to hold all of
   n_elements(struct) x size(a,/dimensions) values.
b) going through each structure in the structure array, and copying its
   copy of "a" into the new array.

I agree the precedence table is misleading on this point. I do think there is a reasonable argument that IDL should understand struct.a[10,4,1,1] and not need to first create a 325MB array to de-reference it. Here's an easy rule of thumb though: in order to avoid the "new array creation" process described above, just attach all indices as close as possible to the quantity they are indexing. "10" goes with struct (we want the 10th struct). [4,1,1] goes with "a" (we want element [4,1,1] of a).

JD
