Subject: Re: structures still confusing
Posted by Mark Hadfield on Tue, 13 Jun 2006 01:42:50 GMT
View Forum Message <> Reply to Message

Martin Rother wrote:

```
 > ...
>      so far ok, I guess. but
>
>  (X). help,s.m
>  <Expression>    LONG      = Array[3, 3]
>  (X). print, s.m
>           1          2          3
>           1          2          3
>           1          2          3
>  (X). print, s.m[0]
>           1          1          1
>  (X). print, s.m[1]
>           2          2          2
>  (X). print, s.m[2]
>           3          3          3
>
>       is a *bit* confusing. isn't it?
```

Not as confusing as a procedure named "!". You're never going to live that one down!

--
Mark Hadfield   "Kei puwaha te tai nei, Hoea tahi tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: structures still confusing
Posted by Allan Whiteford on Tue, 13 Jun 2006 08:13:49 GMT
View Forum Message <> Reply to Message

Martin,

```
print,(s.m)[0]
```

is probably less confusing. Similarly you can do:

```
print,(s.m)[0,1]
```

but you can't do:

```
print,s.m[0,1]
```

or, you can also do:

```
print,(s.m[0])[1]
```

Thanks,

Allan

Martin Rother wrote:
```
>          Hi Gurus,
>
>       finally something, where I don't know,
>       if it's odd idl syntax or not...
>
>       something about structures.
>
>  FUNCTION test_struct
>
>    s = [{ m : [1L, 2L, 3L], n :1L }]
>
>    FOR i = 1L, 2L DO BEGIN
>       ;
>       s = [s, { m : [1L, 2L, 3L], n : i }]
>       ;
>    ENDFOR
>
>    return, s
>
>  END
>
>      this creates an array of structures:
>
>  (X). s = test_struct()
>  (X). help,/struct,s
>  ** Structure <fa330>, 2 tags, length=16, data length=16, refs=1:
>    M              LONG      Array[3]
>    N              LONG                 1
>  (X). print, n_elements(s)
>            3
>  (X).
>
>      so far ok, I guess. but
>
>  (X). help,s.m
>  <Expression>    LONG      = Array[3, 3]
>  (X). print, s.m
>            1          2          3
>            1          2          3
>          1          2          3
> (X). print, s.m[0]
>          1          1          1
> (X). print, s.m[1]
>          2          2          2
> (X). print, s.m[2]
>          3          3          3
>
>     is a *bit* confusing. isn't it?
>
>
>     best regards,
>     martin.
```

Subject: Re: structures still confusing
Posted by btt on Tue, 13 Jun 2006 14:54:00 GMT

View Forum Message <> Reply to Message

Martin Rother wrote:

```
>       Hi Gurus,
>
>     finally something, where I don't know,
>     if it's odd idl syntax or not...
>
>     something about structures.
>
> FUNCTION test_struct
>
>   s = [{ m : [1L, 2L, 3L], n :1L }]
>
>   FOR i = 1L, 2L DO BEGIN
>     ;
>     s = [s, { m : [1L, 2L, 3L], n : i }]
>     ;
>   ENDFOR
>
>   return, s
>
> END
>
>    this creates an array of structures:
>
> (X). s = test_struct()
> (X). help,/struct,s
> ** Structure <fa330>, 2 tags, length=16, data length=16, refs=1:
>   M              LONG      Array[3]
>   N              LONG                1
>  (X). print, n_elements(s)
>            3
>  (X).
>
>       so far ok, I guess. but
>
>  (X). help,s.m
>  <Expression>    LONG      = Array[3, 3]
>  (X). print, s.m
>            1           2           3
>            1           2           3
>            1           2           3
>  (X). print, s.m[0]
>            1           1           1
>  (X). print, s.m[1]
>            2           2           2
>  (X). print, s.m[2]
>            3           3           3
>
>       is a *bit* confusing. isn't it?
>
>
>       best regards,
>       martin.
```

Hi,

Perhaps you are looking to get the entire m array of the ith element in vector s?

```
IDL> s = [{ m : [1L, 2L, 3L], n :1L }]
IDL> FOR i = 1L, 2L DO s = [s, { m : [1L, 2L, 3L], n : i }]
IDL> print, s[0].m
           1           2           3
```

Which is different than asking for the ith elements of the m array in ALL the structures in the vector s.

```
IDL> print, s.m[0]
           1           1           1
```

Or how about the ith element of the m array in the jth element of the vector s?

```
IDL> print, s[0].m[0]
           1
```

I agree it can be confusing (wait till you have pointers in there!) but
is very handy sometimes, too.

Ben

Subject: Re: structures still confusing
Posted by rother on Tue, 13 Jun 2006 15:45:47 GMT

View Forum Message <> Reply to Message

Hi,

On Jun13 10:54, Ben Tupper wrote:
> Martin Rother wrote:
>> (X). print, s.m[2]
>>   3   3   3
> Which is different than asking for the ith elements
> of the m array in ALL the structures in the vector s.
> IDL> print, s.m[0]
>   1   1   1
[...]
> I agree it can be confusing (wait till you have
> pointers in there!) but is very handy sometimes, too.

mmmhhh. I already used nested structures with
and without pointers, but I always used a vast amount
of brackets... so, stupid well-behaving, I indeed never
was aware of this method of 'slicing'! I'm getting
more and more comfortable with that idea... :-)
thanks for all that hints,
m.

--
Martin Rother (rother@gfz-potsdam.de)   +331 / 288-1272   Section 2.3
GeoForschungsZentrum Potsdam, Germany

Subject: Re: structures still confusing
Posted by JD Smith on Tue, 13 Jun 2006 16:33:18 GMT

View Forum Message <> Reply to Message

On Tue, 13 Jun 2006 09:13:49 +0100, Allan Whiteford wrote:

> Martin,
>
> print,(s.m)[0]
>

> is probably less confusing.

But alas, way less efficient, since to access just that one element, it creates an (arbitrarily large) temporary vector (s.m). I don't specifically cover multi-dimensional struct slices, but some info on precedence etc. can be found in:

http://www.dfanning.com/misc_tips/precedence.html

The basic hint here is that structure dereference and array dereference are at the same level of precedence, and don't step on eachother's toes (both left-right associative), so unless you have pointers mixed in, you shouldn't need *any* parentheses to get to an arbitrarily deeply nested array of structure of structure of array of ... and if you do have pointers mixed in, you just need a *single* pair of parentheses around each and every pointer expression (except a top level pointer). Though they result in bizarre looking expressions, the rules are quite simple.

JD