

---

Subject: Re: trouble with pointers within array of structures

Posted by [Markus Schmassmann](#) on Wed, 24 May 2017 12:32:20 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On 05/24/2017 08:49 AM, Helder wrote:

> On Wednesday, May 24, 2017 at 3:12:15 AM UTC+2, wlandsman wrote:

>> When you replicate a scalar pointer, you are making duplicate copies of the \*same\* pointer

>>

>> IDL> p = ptr\_new(dist(256))

>> IDL> pp = replicate(p,10)

>> IDL> help,pp[0],pp[1],pp[2]

>> <Expression> POINTER = <PtrHeapVar23>

>> <Expression> POINTER = <PtrHeapVar23>

>> <Expression> POINTER = <PtrHeapVar23>

>>

>> To make an array of distinct pointers, use ptrarr()

>>

>> IDL> p = ptrarr(3,/all)

>> IDL> help,p[0],p[1],p[2]

>> <Expression> POINTER = <PtrHeapVar24>

>> <Expression> POINTER = <PtrHeapVar25>

>> <Expression> POINTER = <PtrHeapVar26>

>>

>> In your structure example, I think what you want is

>>

>> main = { name:"", image : ptrarr(10,/allocate) }

>> \*main.image[5] = world()

>> \*main.image[4] = ct()

>>

>> --Wayne

>>

>> On Tuesday, May 23, 2017 at 8:12:58 PM UTC-4, Ann Nonymous wrote:

>>> I apologize for what I'm sure is a dumb question but I've looked all through the documentation for three days and I apparently am missing something.

>>>

>>> I'm trying to store a pointer to a vector as an entry within a structure of arrays. This is the basic idea:

>>>

>>>

>>> a = { name:"", image : ptr\_new(/allocate) }

>>> main = replicate(a,10)

>>>

>>> ;world() returns a 256x256 floating point image of the world,

>>> ;ct() returns a 256x256 floating point CT image

>>>

>>>

>>> \*main[5].image = world()

>>> \*main[4].image = ct()

```

>>>
>>> tv, *main[4].image ;gives an image of the CT scan, which is what I expected
>>> tv, *main[5].image ;gives an image of the CT scan
>>> tv, *main[0].image ; ditto
>>> tv, *main[9].image ; ditto
>>>
>>> Obviously, I'm missing something fundamental about the syntax here but I've tried every
permutation of parentheses and indices I can think of, and no matter what, the last pointer
assigned overwrites every other pointer in the structure array.
>>>
>>> Can anyone tell me what I should be doing here?
>>>
>>> Thanks very much,
>
> I've come across this problem once and the reasoning of the way around it is as follows:
>
> Define a structure with a single pointer:
> a = { name:", image : ptr_new()}
> Replicate the structure:
> main = replicate(a,10)
> Now the pointers don't point to anything (no allocation yet).
> help, main[0].image
> When you declare the pointers now, you are each time generating a new one:
> for i=0,9 do main[i].image = ptr_new(i)
> Now you can check the contents:
> for i=0,9 do print, *main[i].image
> 0
> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
>
> And everything is working again.
> The reason why I suggest this solution, is that when defining a structure, you often want to think
about it as a representation of "something" (data). This something might have one dynamic value
(one pointer) or an array of dynamic values (array of pointers).
>
> I hope this helps and is easy to understand.
>
> Cheers,
> Helder
>
Wayne's explanation of the underlying problem and Helder's solution are

```

correct.

However, if you want to avoid unnecessary looping and for code readability prefer to use

```
*main[5].image=world()
```

instead of

```
main[5].image=ptr_new(world())
```

then create the structure with

```
a = { name:"", image : ptr_new() }  
main = replicate(a,10)  
main.image=ptrarr(10,/allocate_heap)
```

---Markus

---