Subject: Structure field concatenation
Posted by Ben Tupper on Thu, 31 Aug 2000 15:02:11 GMT
View Forum Message <> Reply to Message

Hello,

I know that this has been a subject of some discussion recently... but I'm still not on firm footing on the best method of changing the size of an anonymous structure's fields (i.e. I want to increase or decrease the size of a field). The code below shows an example of how I do it now: creating a new structure with the appropriately sized fields. Is there a better method?

```
;++++++++++++ START
PRO Concatenate_Str_Fields

D = {A:Indgen(12), B:Indgen(12)}            ;define a dummy structure

Help, D, /STR

Tags = Tag_Names(D)

NewD = Create_Struct(Tags[0], [D.(0), Indgen(10)])    ;define the new
structure with amended field

For i = 1, N_ELEMENTS(Tags) -1 Do $            ;for each tag recreate
the structure
 NewD = Create_Struct( NewD, Tags[i],[D.(i), Indgen(10)] )

Help, NewD, /STR

END

;--------------------END
```

Thanks,

Ben

BTW: I don't want to steer the discourse toward a scandalous sidebar discussion, but... I'm wrestling with this because a shrimp starts out as a male and then ends up as a female a few years later. I'm working with a database that has the shrimp broken down into records by sex... but I need to add new records for aggregate sex (that is the sum of males, transitionals, females,...) I never thought IDL programming could be so titillating.

--

Ben Tupper
248 Lower Round Pond Road
POB 106
Bristol, ME 04539

Tel: (207) 563-1048
Email: PemaquidRiver@tidewater.net

Subject: Re: Structure field concatenation
Posted by Martin Schultz on Wed, 06 Sep 2000 14:16:02 GMT
View Forum Message <> Reply to Message

Amara Graps wrote:
>
> (at the risk of being the only one here who still hasn't
> figured this out)
>

Reset.

Now try again from scratch:

```
;; create template structure and structure array
template = {orbit:'',freq:ptr_new()}
periodcube = replicate(template, 20)

;; fill first element with data
periodcube.orbit = 'G2'
periodcube.freq  = Ptr_New( DIndgen(100) )

;; Work with data
plot, *(periodcube.freq), title=periodcube.orbit

;; Free all pointers
Ptr_Free, periodcube.freq
```

You probably didn't want to use a second Ptr_New statement out of
fear that would allocate extra memory and create a memory leak.
This is no problem, because the Ptr_New() statement with no
argument only "declares" a pointer but does not allocate any
memory for the data it will eventually point to. Only if you want
to replace the data in a structure element, then you need to free
the pointer beforehand:

```
;; Replace data of first structure
IF Ptr_Valid(periodcube.freq) THEN Ptr_Free,
```

periodcube.freq
  periodcube.freq = Ptr_New(DIndgen(200)*0.1)

Hope this will clear your mind,
Martin

PS:
>   "Never fight an inanimate object." - P. J. O'Rourke
No, it's far better to write them ;-)

--
[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie
[[
[[Bundesstr. 55, 20146 Hamburg
[[
[[phone: +49 40 41173-308
[[
[[fax: +49 40 41173-298
[[
[[martin.schultz@dkrz.de
[[
[[[[[[[[[[

Subject: Re: Structure field concatenation
Posted by Liam E. Gumley on Wed, 06 Sep 2000 14:26:54 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Amara Graps (Amara.Graps@mpi-hd.removethis.mpg.de) writes:
>
>> I appreciate your answer, but then I am back to the same error
>> I inquired about a couple of weeks ago, i.e.:
>>
>> If I do this:
>> thisstruc = {orbit:'',freq:ptr_new()}
>> instead of this:
>> thisstruc = {orbit:'',freq:ptr_new(/allocate_heap)}
>>
>> I get this error when I start to create an array of structures
>> and fill it:
>>
>> periodcube = replicate(thisstruc,1)
>> periodcube(0).orbit = 'G2'

>> *periodcube(0).freq=DINDGEN(100) ;first pointer array is len 100
>>
>> % Unable to dereference NULL pointer: <POINTER (<NullPointer>)>.
>
> Exactly. A NULL pointer is an *invalid* pointer. Hence,
> it cannot be dereferenced. Only valid pointers can be
> dereferenced. A pointer to an undefined variable *is*
> a valid pointer and can be dereferenced, but if you
> replicate the same pointer in a bunch of structures
> all the pointers are to the same variable. It is
> indeed an oscillating universe. :-)
>
> The solution, I think, is to check to see (if you have
> no other way of knowing in your code) if you have
> a valid pointer reference before trying to fill the
> field with data. Something like this:
>
>   thisstruc = {orbit:'',freq:ptr_new()}
>   structs = Replicate(thisStruc, 10)
>   IF Ptr_Valid(structs[5].freq) THEN $
>   *structs[5].freq = FLTARR(100) ELSE $
>   structs[5].freg = Ptr_New(FLATARR(100))

Or you could create the valid pointers first:

;- Create template for one record which contains a single NULL pointer
thisstruc = {orbit:'',freq:ptr_new()}

;- Make an array which has the same NULL pointer in 10 places
structs = replicate(thisstruc, 10)

;- Replace the null pointer with an array of 10 valid pointers
structs.freq = ptrarr(10, /allocate_heap)

;- Store your data
*(structs[0].freq) = findgen(25)
*(structs[1].freq) = dist(256)

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: Structure field concatenation
Posted by davidf on Wed, 06 Sep 2000 14:35:15 GMT
View Forum Message <> Reply to Message

Martin Schultz (martin.schultz@dkrz.de) writes:

> Only if you want
> to replace the data in a structure element, then you need to free
> the pointer beforehand:
>
> ;; Replace data of first structure
> IF Ptr_Valid(periodcube[0].freq) THEN Ptr_Free,
> periodcube[0].freq
> periodcube[0].freq = Ptr_New(DIndgen(200)*0.1)

Actually, as I've been trying to point out for
months now to no avail, it is NOT necessary to
free the pointer in this instance. IDL *takes
care of the memory management for you*. :-)

```
IF Ptr_Valid(periodcube[0].freq) THEN $
  *periodcube[0].freq  = newThingy
```

Cheers,

David

P.S. I'm sure Martin knows this. He is just being
thorough. A trait I have noticed among Germans. :-)

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

Subject: Re: Structure field concatenation
Posted by Martin Schultz on Wed, 06 Sep 2000 15:13:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Martin Schultz (martin.schultz@dkrz.de) writes:
>
>> Only if you want
>> to replace the data in a structure element, then you need to free
>> the pointer beforehand:
>>
>> ;; Replace data of first structure
>> IF Ptr_Valid(periodcube.freq) THEN Ptr_Free,
>> periodcube.freq

>>   periodcube.freq = Ptr_New(DIndgen(200)*0.1)
>
> Actually, as I've been trying to point out for
> months now to no avail, it is NOT necessary to
> free the pointer in this instance. IDL *takes
> care of the memory management for you*. :-)
>
>   IF Ptr_Valid(periodcube.freq) THEN $
>   *periodcube.freq = newThingy
>
> Cheers,
>
> David
>
> P.S. I'm sure Martin knows this. He is just being
> thorough. A trait I have noticed among Germans. :-)

Actually, I didn't know! Well, I've heard this before, but I never believed it would be that easy. Maybe I am just old-fashioned, but I always feel like an equilibrist with no safety net if I replace the contents of a pointer before actually deallocationg the memory it occupies. And I think I will stick to this habit if only for compatibility reasons with FORTRAN. Just imagine I would allow our models to deallocate memory automatically - ain't never gonna happen I fear...

The second motive for doing it my way is that you will need to have two statements anyhow. In your example: what happens if the pointer is not valid? Well, then you need to allocate memory for it, so you write:

```
  IF Ptr_Valid(periodcube[0].freq) THEN $
       *periodcube[0].freq  = newThingy  $
  ELSE  $

       periodcube[0].freq  = Ptr_New(newthingy)
```

The only reason to do this that I could accept without further quirking is if you tell me there is a lot of penalty if you manually deallocate and reallocate the memory instead of letting IDL do it. Haven't tested, but I would doubt that it makes a big difference.

Cheers,
Martin

--
[[[[[[[[[

[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[

Subject: Re: Structure field concatenation
Posted by davidf on Wed, 06 Sep 2000 15:24:37 GMT
View Forum Message <> Reply to Message

Martin Schultz (martin.schultz@dkrz.de) writes:

> The only reason to do this that I could accept without further
> quirking is if you tell me there is a lot of penalty if you manually
> deallocate and reallocate the memory instead of letting IDL do it.
> Haven't tested, but I would doubt that it makes a big difference.

I really don't think there is any difference at all.
If it makes you feel safer, by all means free pointers
yourself. I just wanted to make the point once again that
IDL really does have some nice features. This aspect
of pointer memory management is one of them. :-)

Cheers,

David

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

Subject: Re: Structure field concatenation
Posted by John-David T. Smith on Wed, 06 Sep 2000 20:34:41 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Martin Schultz (martin.schultz@dkrz.de) writes:
>
>> The only reason to do this that I could accept without further
>> quirking is if you tell me there is a lot of penalty if you manually

>> deallocate and reallocate the memory instead of letting IDL do it.
>> Haven't tested, but I would doubt that it makes a big difference.
>
> I really don't think there is any difference at all.
> If it makes you feel safer, by all means free pointers
> yourself. I just wanted to make the point once again that
> IDL really does have some nice features. This aspect
> of pointer memory management is one of them. :-)

I just wanted to point out that this technically isn't pointer memory management at all. Rather, it's simply the same old variable memory management we know and love:

IDL> a=[1,2,3]
IDL> a=1

...no memory loss there! The only difference is that *heap* variables are being handled in the pointer case. So Martin, if you're happy with this, you should be happy with David's method. Of course you might always use "delvar,a", but somehow I doubt it.

I'm being pedantic only to prevent readers (especially the Java-enabled among them) getting confused about what kind of memory management IDL really provides. The best way to stay clear on the issue is to think about pointers as what they are: references to IDL variables which are exactly the same as any other variable except for being hidden ("on the heap") and persistent ("not cleaned up by function/procedure exits"). They do *not* point directly to memory (just as variables like "a" above don't directly map to memory -- thankfully for us).

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: Structure field concatenation
Posted by davidf on Wed, 06 Sep 2000 20:43:54 GMT
View Forum Message <> Reply to Message

J.D. Smith (jdsmith@astro.cornell.edu) writes:

> The best way to stay clear on the issue is to think about pointers as
> what they are: references to IDL variables which are exactly the same as any

> other variable except for being hidden ("on the heap") and persistent ("not
> cleaned up by function/procedure exits"). They do *not* point directly to
> memory (just as variables like "a" above don't directly map to memory --
> thankfully for us).

Hear! Hear!

Cheers,

David

P.S. Let's just say that if pointers *did* map directly to
memory, I could understand RSI's Unlimited Right to
Distribute pricing schedule. And even that probably
wouldn't be enough to hire all the technical support
people they would need. :-)

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

Subject: Re: Structure field concatenation
Posted by Martin Schultz on Thu, 07 Sep 2000 07:46:22 GMT
View Forum Message <> Reply to Message

Now, if an american is getting pedantic on this issue, I would really
like to be thorough here ;-)

Let's state the "problem" again:
We already have a pointer variable, and we want to reassign a new
value to it. But, as typical for "real life" situations, we don't know
for sure if the pointer that we have is valid or not.

This leads to two possible solutions:

dwf:
```
  IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
  *myptr = thingy
```

mgs:
```
  IF Ptr_Valid(myptr) THEN Ptr_Free, myptr
  myptr = Ptr_New(thingy, /No_Copy)
```

Both solutions should lead to the same result and with approximately

the same speed (at least if I add the No_Copy keyword to my solution).
So, the difference is that thingy is still accessible in David's
solution while it becomes undefined in mine (and David's solution
saves a couple of keystrokes which you can then use for
documentation).

Well, if there were no other hidden differences, I would declare
victory to David. BUT, there is in fact a difference!! If you want to
point to an undefined variable (well, who wants to do this anyway),
David's solution breaks, whereas my way happily creates a new pointer
pointing to an undefined variable. Thus, the fail-safe dwf solution
would be:

```
  IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
  IF N_Elements(thingy) GT 0 THEN *myptr = thingy $
  ELSE ...  ; either stop, print a warning, create a new pointer, or
...
```

And this gives me leeway for more documentation ;-)

Cheers,
Martin

"J.D. Smith" wrote:
>
> David Fanning wrote:
>>
>> Martin Schultz (martin.schultz@dkrz.de) writes:
>>
>>> The only reason to do this that I could accept without further
>>> quirking is if you tell me there is a lot of penalty if you manually
>>> deallocate and reallocate the memory instead of letting IDL do it.
>>> Haven't tested, but I would doubt that it makes a big difference.
>>
>> I really don't think there is any difference at all.
>> If it makes you feel safer, by all means free pointers
>> yourself. I just wanted to make the point once again that
>> IDL really does have some nice features. This aspect
>> of pointer memory management is one of them. :-)
>
> I just wanted to point out that this technically isn't pointer memory management
> at all. Rather, it's simply the same old variable memory management we know and
> love:
>
> IDL> a=
> IDL> a=1
>

> ...no memory loss there! The only difference is that *heap* variables are being
> handled in the pointer case. So Martin, if you're happy with this, you should
> be happy with David's method. Of course you might always use "delvar,a", but
> somehow I doubt it.
>
> I'm being pedantic only to prevent readers (especially the Java-enabled among
> them) getting confused about what kind of memory management IDL really
> provides. The best way to stay clear on the issue is to think about pointers as
> what they are: references to IDL variables which are exactly the same as any
> other variable except for being hidden ("on the heap") and persistent ("not
> cleaned up by function/procedure exits"). They do *not* point directly to
> memory (just as variables like "a" above don't directly map to memory --
> thankfully for us).
>
> JD
>
> --
> J.D. Smith /*\ WORK: (607) 255-6263
> Cornell University Dept. of Astronomy */ (607) 255-5842
> 304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
> Ithaca, NY 14853 */

--
[[[[[[[[[
[[Dr. Martin Schultz Max-Planck-Institut fuer Meteorologie [[
[[Bundesstr. 55, 20146 Hamburg [[
[[phone: +49 40 41173-308 [[
[[fax: +49 40 41173-298 [[
[[martin.schultz@dkrz.de [[
[[[[[[[[[

Subject: Re: Structure field concatenation
Posted by davidf on Thu, 07 Sep 2000 14:31:54 GMT
View Forum Message <> Reply to Message

Ben Tupper (btupper@bigelow.org) writes:

> Martin Schultz wrote:
>
>>
>> If you want to
>> point to an undefined variable (well, who wants to do this anyway),
>>
>
> Hello,
>
> I hate to pipe up because I'm the gear-slipping dope that may have started this mess

> (although I'm learning much just listening.) I find utility in a pointer to an
> undefined variable useful when working with lists of things that a user can
> completely empty. (Like a base map with or without any number of overlays, or the
> datasets have not been loaded yet.) Isn't it analogous to a container object that is
> waiting for additions?

Maybe Ben *used* to be a "gear-slipping dope", but
that hardly describes him lately. Let's just say I've
been pushed over my knowledge horizon more times than
I like to admit in the past couple of months by his
thought-provoking questions. :-)

In any case, I'd like to provide more support for the
utility of pointers to undefined variables. Let me give
you a specific example: CW_FIELD.

If you set CW_FIELD up so that it will accept, say,
integer values, and you happen to leave the field
blank, then when you go and get the value in the field
it will return a 0 to you. What's wrong with that?, you
say.

What is wrong is that a 0 is a valid integer value.
So you go willy-nilly on with your code thinking that
you have got something decent. But suppose the number
were the X Size of an image. And now suppose you want
to Congrid your image into this size:

```
displayImage = Congrid(image, xsize, ysize)
```

This causes an error. But now the error message is
very strange (probably impossible to understand if
you don't have a lot of IDL experience) and is one
step (and many lines of code) removed from where the
error really occurred.

If CW_FIELD had returned an undefined variable (well,
what else would it be if the field was blank?), then
the error would have been something that is easily
understood. What is more, it is something that could
be easily checked for:

```
Widget_Control, fieldID, Get_Value=theValue
IF Size(theValue, /TName) EQ 'UNDEFINED' THEN $
   Message, 'Whoops. Field is blank. Try again.'
```

You could argue that you could as easily check to see
if the size is 0, and you would probably be right,

except in those cases where 0 is a valid value. Then
you are really out of luck. (Unless you decide, as
the authors of CW_FIELD did, that 0 is the default
value if the value is undefined. Dubious, at best.)

In any case, I find it *much* more useful to get
an undefined variable when the field is undefined,
so that is why FSC_INPUTFIELD, which is my CW_FIELD
replacement that looks editable on Windows machines,
uses pointers to store the value. If the field is
undefined, then the pointer points to an undefined
variable.

The lucky side effect of using a pointer to store
the value, is that I can also include the value
of the field in the event structure itself, which
is something CW_FIELD has never been able to do.
So I get two major benefits from using pointers.
Hard to argue with those economics. :-)

Cheers,

David

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

Subject: Re: Structure field concatenation
Posted by Amara Graps on Mon, 11 Sep 2000 11:04:06 GMT
View Forum Message <> Reply to Message

Hi folks,

I want to thank all of you for your help on this topic. After
trying the suggestions, I was successful at performing a flexible
concatenation of an array of structures with elements that are
pointers. I append the following small program that demonstrates
how to do this, in case anyone else is interested.

Thanks again!

Amara

```
;==========cut here=======================================

;Program testpointer.pro

;PURPOSE: This IDL program is an example of an array of anonymous
;structures of fields including a pointer field. Here we don't know how
;many structures we want in our array of structures, nor do we know the
;data definition of of our pointer. This program is one way to create an
;array of structures on the fly, with data that is not defined
beforehand.
;Note: tested on IDL Version 5.3 (sunos sparc)
 ;------------------------------------------------------------ --
;Amara Graps 9 Septempber 2000
 ;------------------------------------------------------------ --

;STEP 1
;Create an anonymous structure
thisstruc = {orbit:'',freq:PTR_NEW()}

;Create a 1-element array of anonymous structure  (we will concatenate
it,
;to make a longer array, when we need it).
periodcube = REPLICATE(thisstruc,1)

;Assign the structure values: Gal orbit G2, index array of length 100
periodcube[0].orbit = 'G2'
CASE 1 OF
      PTR_VALID(periodcube[0].freq):
*periodcube[0].freq=DINDGEN(100)
      ELSE: BEGIN
            ;Make it a valid pointer and fill it
            periodcube[0].freq = PTR_NEW(DINDGEN(100))
      END
ENDCASE

;set a variable to the pointer, if we want to play with it
test1 = *periodcube[0].freq
;Take a look
help, test1

;STEP 2
;Update the structure by creating a temporary structure like the
;original, and then concantenating
tempperiod = thisstruc

;Assign the structure values, Gal orbit C3, index array of length 50
tempperiod.orbit = 'C3'
CASE 1 OF
```

```
        PTR_VALID(tempperiod.freq): *tempperiod.freq=DINDGEN(50)+50
        ELSE: BEGIN
              ;Make it a valid pointer and fill it
              tempperiod.freq = PTR_NEW(DINDGEN(50)+50)
        END
ENDCASE

;Set a variable to the pointer, if we want to play with it
test2 = *tempperiod.freq
;Take a look
help, test2

;Concatenating
new = [periodcube,tempperiod]

;Set variables to the pointers in the structure
;in order to perform more manipulation
freq1 = *new[0].freq
freq2 = *new[1].freq

;Take a look  (These are OK)
help, freq1, freq2
help, *new[0].freq,*new[1].freq
help, new[0].orbit, new[1].orbit

;To concantenate indefinitely, rename and repeat STEP 2
periodcube =  new

STOP, 'look at structure periodcube'

END     ;of program testpointer.pro
```

--

 *** ***
Amara Graps | Max-Planck-Institut fuer Kernphysik
Interplanetary Dust Group | Saupfercheckweg 1
+49-6221-516-543 | 69117 Heidelberg, GERMANY
                    * http://galileo.mpi-hd.mpg.de/~graps
 *** ***

"Never fight an inanimate object." - P. J. O'Rourke

Subject: Re: Structure field concatenation
Posted by John-David T. Smith on Mon, 11 Sep 2000 18:59:59 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:

>
> Now, if an american is getting pedantic on this issue, I would really
> like to be thorough here ;-)
>
> Let's state the "problem" again:
> We already have a pointer variable, and we want to reassign a new
> value to it. But, as typical for "real life" situations, we don't know
> for sure if the pointer that we have is valid or not.
>
> This leads to two possible solutions:
>
> dwf:
>   IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
>   *myptr = thingy
>
> mgs:
>   IF Ptr_Valid(myptr) THEN Ptr_Free, myptr
>   myptr = Ptr_New(thingy, /No_Copy)
>
> Both solutions should lead to the same result and with approximately
> the same speed (at least if I add the No_Copy keyword to my solution).
> So, the difference is that thingy is still accessible in David's
> solution while it becomes undefined in mine (and David's solution
> saves a couple of keystrokes which you can then use for
> documentation).
>
> Well, if there were no other hidden differences, I would declare
> victory to David. BUT, there is in fact a difference!! If you want to
> point to an undefined variable (well, who wants to do this anyway),
> David's solution breaks, whereas my way happily creates a new pointer
> pointing to an undefined variable. Thus, the fail-safe dwf solution
> would be:
>
>   IF NOT Ptr_Valid(myptr) THEN Ptr_New(myptr)
>   IF N_Elements(thingy) GT 0 THEN *myptr = thingy $
>   ELSE ... ; either stop, print a warning, create a new pointer, or
> ...
>
> And this gives me leeway for more documentation ;-)
>

Time to double-down the pedantry. Both methods you mention are perfectly valid and useful, in different ways, but your argument is unrelated to the topic I was addressing. The notion I attempted to dispel was the need for a "safety net" of any kind when reassigning an already allocated pointer's value, as if IDL's memory management in this case could not be trusted. By arguing that this pointer heap variable management was tantamount to traditional variable memory management, I showed that if you trust the latter (a condition I cannot prove),

then you implicity trust the former.

And now to address your slightly rescoped points along with David and Ben's rejoinders...

You could slightly improve your method by changing the first line to simply:

Ptr_Free, myptr

which is faster than an "if" test in the case of myptr being null (and saves you even more keystrokes). This allows very simple pointer cleanup when used appropriately (as I discussed a few weeks back).

As for David and Ben's argument for undefined ptr values, I argue that a non-existent pointer is just as valid a placeholder of a null field or empty list as a pointer pointing to an undefined value. The test simply changes from:

IF Size(theValue, /TName) EQ 'UNDEFINED' THEN

to

IF NOT ptr_valid(theValue) THEN

Also, how is an undefined pointer created manually after the fact -- i.e., how do you "empty" am already filled list? Something awkward like:

```
ptr_free, theValue
theValue=ptr_new(/alloc)
```

where I would only need the first line. To be fair to the advantages of the Undefined method... if changing the pointed-to value altogether, they need only:

*theValue=newvalue

whereas I require:

if ptr_valid(theValue) then *theValue=newvalue else theValue=ptr_new(newvalue)

A remaining issue which affects me by far the most, is one of appending data to a pointer heap variable which is an array (possibly not yet existing). I end up with much code looking like:

```
if ptr_valid(self.arr) then *self.arr=[*self.arr,newval] else $
 self.arr=ptr_new([newval])
```

Which seems wordy. Unfortunately, and undefined value method does not help:

```
IDL> arr=ptr_new(/alloc)
IDL> *arr=[*arr,1] ; oops, won't work!
% Variable is undefined: <PtrHeapVar4>.
```

you end up needing:

```
if Size(self.arr, /TName) EQ 'UNDEFINED' THEN *self.arr=[newval] else $
 *self.arr=[*self.arr,newval]
```

not any better.

And what about Martin's method of pointer rebirth? It simply doesn't work for extending pointed-to arrays without an awkward temp variable.

```
tmp=*self.arr
ptr_free,self.arr
if Size(tmp, /TName) EQ 'UNDEFINED' then self.arr=ptr_new(newval) else $
  self.arr=ptr_new([tmp,newval])
```

So, as is usual with these arguments, there is no right or wrong answer. Here are the choices with the pros and cons:

1. I prefer keeping my heap variables in place, and indicate lists are empty with null pointers (by freeing the pointers as appropriate), which becomes quite easy. My "empty" lists take up no memory on the heap. I pay for this emptying ease when reassigning a pointed-to value, where I require a test to ensure the pointer heap variable exists (creating one if necessary). Appending to pointed-to arrays is reasonably easy.

2. David and Ben prefer to keep their variables around even longer, indicating empty lists with pointers to undefined variables. They can easily reassign pointed-to values (since the heap variable will always be there), but manually emptying a list is more awkward, requiring a variable to be freed and reassigned to a newly created undefined heap variable (ptr_new(/alloc)). Appending to pointed-to arrays is about as hard as method #1.

3. Martin prefers continuously recreating his pointer heap variables, which allows him to assign undefined variables, but not really more easily than in #2, since he's always freeing them anyway just as when emptying a list in that method. It does allow him to assign *unexpectedly* undefined variables (like typos? ... hmm, not sure if this is a pro or a con, or unpassed arguments, as David comments), since the semantics are exactly the same either way, in contrast to #2. The empty list could reasonably be either that of #1 or #2. Appending to (possibly nonexistent) arrays is very awkward compared to the former two.

So there you have it. None exist in exclusion of the others, and each can borrow from the other as appropriate. But note that it is very convenient to

adopt a single "list is empty" paradigm so your code easily interoperates.

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: Structure field concatenation
Posted by John-David T. Smith on Mon, 11 Sep 2000 19:22:39 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Ben Tupper (btupper@bigelow.org) writes:
>
>> Martin Schultz wrote:
>>
>>>
>>> If you want to
>>> point to an undefined variable (well, who wants to do this anyway),
>>>
>>
>> Hello,
>>
>> I hate to pipe up because I'm the gear-slipping dope that may have started this mess
>> (although I'm learning much just listening.) I find utility in a pointer to an
>> undefined variable useful when working with lists of things that a user can
>> completely empty. (Like a base map with or without any number of overlays, or the
>> datasets have not been loaded yet.) Isn't it analogous to a container object that is
>> waiting for additions?
>
> Maybe Ben *used* to be a "gear-slipping dope", but
> that hardly describes him lately. Let's just say I've
> been pushed over my knowledge horizon more times than
> I like to admit in the past couple of months by his
> thought-provoking questions. :-)
>
> In any case, I'd like to provide more support for the
> utility of pointers to undefined variables. Let me give
> you a specific example: CW_FIELD.
>
> If you set CW_FIELD up so that it will accept, say,
> integer values, and you happen to leave the field
> blank, then when you go and get the value in the field

> it will return a 0 to you. What's wrong with that?, you
> say.
>
> What is wrong is that a 0 is a valid integer value.
> So you go willy-nilly on with your code thinking that
> you have got something decent. But suppose the number
> were the X Size of an image. And now suppose you want
> to Congrid your image into this size:
>
> displayImage = Congrid(image, xsize, ysize)
>
> This causes an error. But now the error message is
> *very* strange (probably impossible to understand if
> you don't have a lot of IDL experience) and is one
> step (and many lines of code) removed from where the
> error really occurred.
>
> If CW_FIELD had returned an undefined variable (well,
> what else would it be if the field was blank?), then
> the error would have been something that is easily
> understood. What is more, it is something that could
> be easily checked for:
>
> Widget_Control, fieldID, Get_Value=theValue
> IF Size(theValue, /TName) EQ 'UNDEFINED' THEN $
> Message, 'Whoops. Field is blank. Try again.'
>
> You could argue that you could as easily check to see
> if the size is 0, and you would probably be right,
> except in those cases where 0 is a valid value. Then
> you are really out of luck. (Unless you decide, as
> the authors of CW_FIELD did, that 0 is the default
> value if the value is undefined. Dubious, at best.)
>
> In any case, I find it *much* more useful to get
> an undefined variable when the field is undefined,
> so that is why FSC_INPUTFIELD, which is my CW_FIELD
> replacement that looks editable on Windows machines,
> uses pointers to store the value. If the field is
> undefined, then the pointer points to an undefined
> variable.

Using undefined variables as placeholders for empty lists or null fields is a fine idea on the surface (usually we must resort to things like -1, c.f. where()). The problem is you can't return them from functions or assign them to variables. If so I could have:

wh=where(indgen(10) lt 0)

if defined(wh) then blah

but alas, I must use -1 as the test. If we collected all of the semantics for functions which need to indicate that they are returning nothing, the list would be frightfully long...

-1 where >0 is expected
0 where >1 is expected
any scalar where an array is expected
any number where a string is expected
any string (like '') where a number is expected
a number (like 0) where a pointer is expected
...

and so on. A natural value for *all* of these is "undefined", which can be returned through arguments (keyword or positional), but not by functions. We wouldn't have to scratch our heads every time we wanted to rest a return value. I myself use a collection of the above methods in my own code.... wasted time thinking about which test to use.

If we wanted to live with the present situation, we could adopt the policy that any routine which is to return a value which may be null or empty should be implemented as a procedure. But wait, even that won't work. Why? Consider:

```
pro getit, var, SOMETHING=something
 if something then var=1
end

IDL> getit,var
IDL> print,size(var,/TNAME)
UNDEFINED
IDL> getit,var,/SOMETHING
IDL> print,size(var,/TNAME)
INT
IDL> getit,var
IDL> print,size(var,/TNAME)
INT
```

Uh oh, we can't tell our variable wasn't set on the last one... it retains its previous value. The pass-by-reference of arguments has defeated us. This is where a language like Perl excels. There you could simply say "var=undef". If we had a little more control over when and where our variables get undefined, we'd be much better off. Heck I might even switch to the pointer-to-undefined-heap-var for my empty lists on the heap.

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: Structure field concatenation
Posted by Ben Tupper on Mon, 11 Sep 2000 22:28:54 GMT
View Forum Message <> Reply to Message

Hello,

Geez, you post such great stuff. Thank you for taking the time to do so.

I hadn't thought that the empty list method I lean on was awkward until you mentioned it... now I'm all thumbs. I can see the utility in each method you outline.

I don't know how David deals with emptied lists, but here's how I usually do it.
In general, when a list must be emptied, I have employed the UNDEFINE procedure (available from David's or Martin's website.)

```
UNDEFINE, *myPtr
```

The UNDEFINE procedure calls the SIZE() function...

```
Sz = SIZE(TEMPORARY(*myPtr))
```

I test the emptiness of a pointer with N_ELEMENTS() before adding to the list...

```
If N_ELEMENTS(*Ptr) EQ 0 then *Ptr = NewValue Else *Ptr = [*Ptr, NewValue]
```

"J.D. Smith" wrote:

> Also, how is an undefined pointer created manually after the fact -- i.e., how
> do you "empty" am already filled list? Something awkward like:
>
> ptr_free, theValue
> theValue=ptr_new(/alloc)
>
> where I would only need the first line. To be fair to the advantages of the
> Undefined method... if changing the pointed-to value altogether, they need only:
>

> *theValue=newvalue
>
> whereas I require:
>
> if ptr_valid(theValue) then *theValue=newvalue else theValue=ptr_new(newvalue)
>

I was curious about the time it takes to run method 1 versus method 2.
Below is
a quick test of each method.
I did not use UNDEFINE procedure directly because it checks for
arguments which
slows things down. I placed the Sz = SIZE(TEMPORARY(*myPtr)) statement
in its
stead.

There doesn't seem to be any time difference, which makes suspicious
that I don't
have a good test here. (BTW, the ratio is 2.2 if I call UNDEFINE,
which I usually
do.) Here are the results:

```
IDL> list_ptr
Elapsed time to empty/fill list using method #1
     0.066666603 seconds
Elapsed time to empty/fill list using method #2
     0.066666603 seconds
Ratio of #2 / #1
       1.0000000
```

Ben

```
;------START
PRO LIST_PTR

X = Findgen(100)

A = PTR_NEW(X)
B = PTR_NEW(X)

Start1 = SysTime(/Seconds)
For i = 0L, 10000 Do Begin
 Ptr_Free, B
 If NOT Ptr_Valid(B) Then B = Ptr_NEW(X) Else *B = [*B, X]
EndFor
Fini1= SysTime(/Seconds)
```

```
Print, 'Elapsed time to empty/fill list using method #1'
Print, Fini1-Start1, ' seconds'

Start2 = SysTime(/Seconds)
For i = 0L, 10000 Do begin
 tempvar = SIZE(TEMPORARY(*A))
 If N_elements(*A) EQ 0 Then *A = X Else *A = [*A,X]
EndFor
Fini2 = Systime(/Seconds)

Print, 'Elapsed time to empty/fill list using method #2'
Print, Fini2-Start2, ' seconds'


Print, 'Ratio of #2 / #1'
print, (Fini2-Start2)/(Fini1-Start1)

Ptr_Free, A, B
END
;----------END
```

--
Ben Tupper
Bigelow Laboratory for Ocean Science
West Boothbay Harbor, Maine
btupper@bigelow.org
    note: email address new as of 25JULY2000