Subject: pointers in IDL
Posted by Christian Oehreneder on Mon, 28 Apr 1997 07:00:00 GMT
View Forum Message <> Reply to Message

Dear colleagues!

I need help on the following problem:

My application uses a not a priori known number of images with different sizes, which I want to access easily and flexible, preferrably by writing img_i = img(i) or something similar. Because of the different sizes of the images this is not easily done in IDL.
In common programming lanuages it would be natural to have an array of pointers to each of the images.
So far I have not found a possibilty to do this in IDL. Handles are nice but they do not allow to have two pointers on the same data. The effect is that once you take the value of a handle (without copying it of course !!) the handle itself contains an undefined value. It is not possible to have a list of images and pass them to some manipulation routines by pointer, e.g. multiplying all images by a factor of 2. Of course it is possible to take the value form the handle, multiply it and set it as value of the handle, but that's rather cumbersome.

Has anyone an idea how to work arround this??

Thanks in advance

Christian Oehreneder
********** **********
Christian Oehreneder

Institute for Photogrammetry and Remote Sensing
Technical University Vienna

TU Wien, Institut 122 Tel.: +43 1 58801-3730
Gusshausstrasse 27-29 FAX: +43 1 5056268
A-1040 Wien Email: co@ipf.tuwien.ac.at
AUSTRIA ((coehrene@fbgeo1.tuwien.ac.at))
********** **********

Subject: Re: Pointers
Posted by Liam Gumley on Mon, 09 Aug 1999 07:00:00 GMT
View Forum Message <> Reply to Message

RLeejoice wrote:
> Been using IDL for a year. Want to expand into what I feel are three last
> topics of interest; pointers, widgets, and object graphics.

>
> I've read the entire pointer section of the help file and have the following
> question:
>
> What are pointers used for in IDL. I only precieve that they lead to the new
> object paradyme. Is this correct? I use structures in most of my programming
> and pass the complete structure to the appropaite procedures. I suppose I
> could creat a pointer to a structure, but since IDL passes structures by
> reference, what is the need?

There are at least two cases I know of where pointers are indispensable:

(1) When the size and/or type of an item contained in a structure is not known until runtime (you cannot change the size or type of a variable in a structure). For example, say you have a widget program with a large information structure. At some point in the widget program, you want to be able to select sub-regions of an image and store the data from the selected regions (e.g. to save it to file elsewhere in the program). The most convenient way to save this data in the information structure is to use an array of pointers, where each pointer in the array points to the data for each selected region.

(2) When you wish to return information from a dying widget. If you want a widget program to return an item of information, then it must be passed in a variable which has global scope. A pointer is the most convenient way to pass this kind of information. The method is
- Create the information structure,
- Store the information structure using a pointer,
- Store the pointer in the user value of the widget top level base,
- Invoke XMANAGER to handle widget events with the CLEANUP keyword set to the name of the routine to be called when the top level widget dies,
- In the cleanup routine, save whatever information is required via the pointer,
- After XMANAGER is done, retrieve the information from the pointer.

Cheers,
Liam.

--
Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

---

Subject: Re: Pointers
Posted by davidf on Mon, 09 Aug 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Robert Leejoice (rleejoice@aol.com) writes:

> What are pointers used for in IDL. I only precieve that they lead to the new
> object paradyme. Is this correct? I use structures in most of my programming
> and pass the complete structure to the appropaite procedures. I suppose I
> could creat a pointer to a structure, but since IDL passes structures by
> reference, what is the need?

Pointers are used for all kinds of things, but I think
the most obvious good use of pointers is in structures.
(Or objects, which are implemented as named structures.)

Suppose, for example, that you have a field named IMAGE
in your structure. But the image data that is stored there
might vary in size and data type. Without pointers, you
would have to use an anonymous structure and redefine it
when the image data changed. With pointers, you can continue
to use a named structure with the IMAGE field a pointer to
whatever you like:

```
struct = {MYSTRUCT, Image:Ptr_New(image), ...}
*struct.image = newimage
```

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: Pointers in IDL
Posted by Antonio Santiago on Tue, 13 Apr 2004 17:08:35 GMT
View Forum Message <> Reply to Message

Well, first of all sorry about my poor english (Perhaps i not understand
well your message).

A pointer is a type of variable that points to another variable.
For example, i have an struct "image":

```
{table, $
 parent:0L, $
 boxtable:0L, $
 n_rows:0, $
```

```
 n_columns:0, $
 type: 0, $
 data: ptr_new() $
}
```

At first time i dont know number of columns, rows and  data. Because this i create 'data' as a pointer (a NULL pointer).

After this when user specifies me a file a restore the data and fill the struct fiels with values and initialize the pointer:

```
IF PTR_VALID( self.child ) THEN $
 PTR_FREE, self.child

self.child = ptr_new( REPLICATE({children}, n_columns, n_rows) )
```

---

If you use ptr_new() you get a NULL pointer that dont points to anything.
If you use ptr_new(/allocate_heap) yo obtain a valid pointer taht points to a undefined variable at heap.

If you use the second form you must to free the data before to re-reference the pointer other time.

If you do:

```
a=ptr_new(bytarr(10))
```

you must free 'a' before do:  a=ptr_new(bytarr(50)), if not you will have a lake.

Benjamin Hornberger wrote:
> Hi all,
>

> I still don't understand all aspects of pointers in IDL. 2 Questions:
>
> 1. What are null pointers for? I read that they can't be dereferenced. What
> is their purpose then? The Gumley book writes (pg. 61): "Null pointers are
> used when a pointer must be created, but the variable ... does not yet
> exist." What would I do then when the variable does exist later and I want
> the pointer to point to it? Wouldn't I use ptr_new(/allocate_heap) in the
> first place, i.e. not create a null pointer but a pointer to an undefined
> variable? Can anyone give an example when I would use ptr_new()?
>
> 2. If I point a pointer to a variable (e.g. *ptr=indgen(100)) and later
> point it to a smaller variable (*ptr=indgen(50)), do I have a memory leak?
> I.e., do I have to free it before I re-reference it?
>
> I want to write a GUI which can open files which contain arrays of varying
> size. Is it ok to define a pointer in the GUI to hold these arrays
> (ptr=ptr_new(/allocate_heap)), and then whenever I open a new file, just
> dereference to the new array (*ptr=array)? Or do I have to free the pointer
> when I close one file and open another one?
>
> Thanks for your help,
> Benjamin

---

Subject: Re: Pointers in IDL
Posted by R.Bauer on Tue, 13 Apr 2004 17:37:02 GMT
View Forum Message <> Reply to Message

Benjamin Hornberger wrote:

> Hi all,
>
> I still don't understand all aspects of pointers in IDL. 2 Questions:
>
> 1. What are null pointers for?

If you have a special list (in german it is verkettete Liste) then it is
easy to recognize the end of the list. Because this is normally a null
pointer. Otherwise it is a pointer to a pointer in the list.
This is often used for search algorithm.
You get a null pointer if you free a pointer by ptr_free too. The status of
a pointer could be tested by ptr_valid(). If it is 1 you could dereference
the pointer otherwise it is a null pointer.

At the end of r program you should free all your pointers. Because pointer
are global variables in case of the memory usage.

I read that they can't be dereferenced.
> What is their purpose then? The Gumley book writes (pg. 61): "Null
> pointers are used when a pointer must be created, but the variable ...
> does not yet exist." What would I do then when the variable does exist
> later and I want the pointer to point to it? Wouldn't I use
> ptr_new(/allocate_heap) in the first place, i.e. not create a null pointer
> but a pointer to an undefined variable? Can anyone give an example when I
> would use ptr_new()?

```
ptr1=ptr_new(5)
a=findgen(10)
ptr2=ptr_new(a,/no_copy)
help,a
ptr3=ptr_new(/allocate_heap)
*ptr3='ALPHA'
```

If you would like to define a complex structure with pointers and you
haven't all data at the moment of the definition I myself define them with
a string constant ('UNDEFINED')
e.g.

```
pi={name:'UNDEFINED','email':'UNDEFINED'}
```

Later on it is easy to replace the value by a new assignment and you have
not to test always if it is a Null Pointer or could be written.
Another reason for this is you can easily check if the value is different
from 'UNDEFINED'. tags with values of 'UNDEFINED' could be removed.

>
> 2. If I point a pointer to a variable (e.g. *ptr=indgen(100)) and later
> point it to a smaller variable (*ptr=indgen(50)), do I have a memory leak?
> I.e., do I have to free it before I re-reference it?

This makes no difference, because Pointer in idl are totally different from
pointer in C. There is a heap control mechanism in idl which controls all
the pointer assignment and the memory usage of them.

With heap_gc you could do a garbage collection of the heap memory.

There was in the past a discussion about memory leaks. Please have a look
into the google archive
http://groups.google.de/groups?hl=de&lr=&ie=UTF-8&am p;group=comp.lang.idl-pvwave

>
> I want to write a GUI which can open files which contain arrays of varying

> size. Is it ok to define a pointer in the GUI to hold these arrays
> (ptr=ptr_new(/allocate_heap)), and then whenever I open a new file, just
> dereference to the new array (*ptr=array)? Or do I have to free the
> pointer when I close one file and open another one?

I prefer *ptr=array

But you should think about a structure if you use more than one pointer.
This structure could be then a pointer too.

Cheers
Reimar

>
> Thanks for your help,
> Benjamin

--
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de
http://www.fz-juelich.de/icg/icg-i/
========== ======
a IDL library at ForschungsZentrum Juelich
http://www.fz-juelich.de/icg/icg-i/idl_icglib/idl_lib_intro. html

---

Subject: Re: Pointers in IDL
Posted by Rick Towler on Tue, 13 Apr 2004 18:06:11 GMT
View Forum Message <> Reply to Message

"Benjamin Hornberger" wrote...

> 1. What are null pointers for?

There may be others but the big one is object class definition. I am going to butcher the explanation but...

When you define the named structure of your object class you define the element names and types. When you create an instance of your object pointers and objects in this "self" structure are null, regardless of how you defined them.

So say you wrote your object definition like so, using the /ALLOCATE_HEAP keyword when defining your myPointer:

```
pro myobj__define

   self= {myobj, $
         myPointer:PTR_NEW(/ALLOCATE_HEAP) $
         }
end
```

If, when creating an instance of your object, myPointer was undefined (not null) you should be able to assign it a value by simply dereferencing it:

```
function myobj::init, data

   *self.myPointer = data

   RETURN, 1

end
```

But this would fail with an "Unable to dereference null pointer" error. IDL ignores your /ALLOCATE_HEAP keyword and always assigns null pointers to pointer and object reference variables in class definition structures.

Our simple object would typically be written like:

```
function myobj::init, data

   self.myPointer = PTR_NEW(data)

   RETURN, 1

end

pro myobj__define

   null = {myobj, $
          myPointer:PTR_NEW() $
          }
end
```

> 2. If I point a pointer to a variable (e.g. *ptr=indgen(100)) and later
> point it to a smaller variable (*ptr=indgen(50)), do I have a memory leak?
> I.e., do I have to free it before I re-reference it?

No. You only "leak" when you forget or are unable to free the pointer when

you are finished with it.

So the following is fine:

```
ptr = PTR_NEW(indgen(100))
*ptr = 'String'
*ptr = FINDGEN(50)
PTR_FREE, ptr
```

But this is bad:

```
ptr = PTR_NEW(indgen(100))
ptr = 'String'
```

ptr used to contain the reference to a heap variable, but we lost that reference when set ptr = 'String'. Since we have lost our reference we can't free the pointer and that memory will be lost for the rest of the IDL session. (This isn't entirely true. You can reclaim lost heap variables using the PTR_VALID function. Check the docs.)

> I want to write a GUI which can open files which contain arrays of varying
> size. Is it ok to define a pointer in the GUI to hold these arrays
> (ptr=ptr_new(/allocate_heap)), and then whenever I open a new file, just
> dereference to the new array (*ptr=array)? Or do I have to free the pointer
> when I close one file and open another one?

No need to free until you exit your application.

-Rick

---

Subject: Re: Pointers in IDL
Posted by JD Smith on Wed, 14 Apr 2004 01:08:01 GMT
View Forum Message <> Reply to Message

On Tue, 13 Apr 2004 11:29:45 -0400, Benjamin Hornberger wrote:

> Hi all,
>
> I still don't understand all aspects of pointers in IDL. 2 Questions:
>
> 1. What are null pointers for? I read that they can't be dereferenced. What
> is their purpose then? The Gumley book writes (pg. 61): "Null pointers are
> used when a pointer must be created, but the variable ... does not yet

> exist." What would I do then when the variable does exist later and I want
> the pointer to point to it? Wouldn't I use ptr_new(/allocate_heap) in the
> first place, i.e. not create a null pointer but a pointer to an undefined
> variable? Can anyone give an example when I would use ptr_new()?
>
> 2. If I point a pointer to a variable (e.g. *ptr=indgen(100)) and later
> point it to a smaller variable (*ptr=indgen(50)), do I have a memory leak?
> I.e., do I have to free it before I re-reference it?
>
> I want to write a GUI which can open files which contain arrays of varying
> size. Is it ok to define a pointer in the GUI to hold these arrays
> (ptr=ptr_new(/allocate_heap)), and then whenever I open a new file, just
> dereference to the new array (*ptr=array)? Or do I have to free the pointer
> when I close one file and open another one?

Here is the absolute best way to think of pointers in IDL (and it has the advantage that it's actually the real way they are handled internally, I believe). A pointer is *nothing* more than a specially accessed, but otherwise regular-old variable. Anything you can do with a variable, you can do with a de-referenced pointer. Actions like:

```
IDL> *ptr=indgen(100)
IDL> *ptr=indgen(5)
```

are just as allowed as if you had used a regular variable:

```
IDL> var=indgen(100)
IDL> var=indgen(5)
```

Another particular application of the "a deferenced-pointer is really just a variable" rule which many may not know.... you can pass them by reference! Suppose you have a function set_to_pi which sets its argument to PI:

```
pro set_to_pi,arg
  arg=!PI
end
```

You won't be surprised when:

```
IDL> set_to_pi,a
IDL> print,a
      3.14159
```

but would you believe:

```
IDL> b=ptr_new(/ALLOCATE_HEAP)
IDL> set_to_pi,*b
IDL> print,*b
      3.14159
```

That's right, *b is just a variable, and set_to_pi doesn't know the difference: it's passed in by reference and dutifully set to PI. When b is a null pointer, it is *not* a variable, it's just a loose end waiting to be tied to one.

In IDL, pointers always point to a special stash of regular-old variables called "heap variables". There's nothing out of the ordinary about them, except they can only be accessed through pointers (or objects, but that's a side issue). In all other ways, they are just normal IDL variables. Heap variables even get funny names internally:

```
IDL> print,b
<PtrHeapVar2>
```

Here we see b points to "ptrheapvar2", i.e. the second variable on the "pointer heap". We can even have a look at that variable directly:

```
IDL> help,/heap
Heap Variables:
   # Pointer: 2
   # Object : 0

<PtrHeapVar1>   INT      =       1
<PtrHeapVar2>   FLOAT    =      3.14159
```

In some languages you can have pointers pointing to normal variables, but not in IDL: a variable is either of the normal variety (like 'a' above), or of the heap variety (like 'ptrheapvar2' above). The only distinction, again, is how you access them. The first kind spring into existence as IDL runs, the latter have to be specifically requested with PTR_NEW(). Once you have this mental model in mind (and specifically forget any baggage you may bring from an understanding of pointers in C), it will all seem much clearer.

JD

---

Subject: Re: Pointers in IDL
Posted by Peter Clinch on Wed, 14 Apr 2004 14:55:03 GMT
View Forum Message <> Reply to Message

Benjamin Hornberger wrote:

> 1. What are null pointers for?

Primarily as checks and balances, I think. If you check a pointer before use and find it's null then you know you shouldn't be using it, so it helps writing robust code.

So, for example, in a linked list where the last thing in the list element structure is a pointer to the next element, the last element's pointer to the "next" element would be a null (there is no next at the end of the list). Traversing the list, you'd check what the next element is, and finding it's a null you'd abandon the list traverse knowing you'd got to the end.

Pete.

--
Peter Clinch University of Dundee
Tel 44 1382 660111 ext. 33637 Medical Physics, Ninewells Hospital
Fax 44 1382 640177 Dundee DD1 9SY Scotland UK
net p.j.clinch@dundee.ac.uk http://www.dundee.ac.uk/~pjclinch/