Subject: losing pointers when concatenating array of structures
Posted by MarioIncandenza on Mon, 09 Jan 2017 06:35:37 GMT
View Forum Message <> Reply to Message

Hi IDL Wizards,

That topic is a mouthful, but I haven't encountered a tricky problem like this in a while, and I wanted your help to understand what's really going on here. Here's the code sample:

```
for i=0,1 do begin
  undefine,struct
  struct=create_struct('array',ptr_new(lindgen(i+1)))
  if n_elements(structarray) eq 0 then structarray=struct else structarray=[structarray,struct]
  help,*structarray[i].array   ; looking for error
; this is where the error will happen
  if(i gt 0) then help,*structarray[i-1].array
endfor
end
```

Here's the result:

```
% Compiled module: $MAIN$.
<PtrHeapVar23543>
              LONG      = Array[1]
<PtrHeapVar23544>
              LONG      = Array[2]
% Invalid pointer: <POINTER  (<PtrHeapVar23543>)>.
% Execution halted at: $MAIN$             9
```

Here's the fix:

```
  if n_elements(structarray) eq 0 then structarray=temporary(struct) else
structarray=[structarray,temporary(struct)]
```

If you use TEMPORARY(), this works. If you do not, it fails. Can someone help me understand this?

Happy New Year's everyone,

--Edward H.

Subject: Re: losing pointers when concatenating array of structures
Posted by Heinz Stege on Mon, 09 Jan 2017 10:10:27 GMT
View Forum Message <> Reply to Message

Well, pointers are tricky sometimes. I will try to explain. The
statement

```
  structarray=struct
```

in the first loop-path creates a new structure with a new pointer variable pointing to the _same_ heap variable as struct.array. The heap variable is _not_ copied.

Undefine is not an IDL routine. I assume you have downloaded this procedure from the Coyote library. In the second loop path undefine destroys the struct.array pointer. I.e. it deletes the heap variable pointed to by struct.array as well as structarray.array.

Your fix with temporary(struct) moves the structure from struct to structarray. The undefine procedure can't delete the heap variable anymore, because struct is undefined at that time.

I'm not sure, that I use the right terms in my explanation. I hope, it gets clear for you yet.

Heinz

Subject: Re: losing pointers when concatenating array of structures
Posted by MarioIncandenza on Mon, 09 Jan 2017 16:44:29 GMT
View Forum Message <> Reply to Message

On Monday, January 9, 2017 at 2:09:58 AM UTC-8, Heinz Stege wrote:
> Well, pointers are tricky sometimes. I will try to explain.

That is a good answer, thank you Heinz. So here's a question: what if I want to make a *copy* of a structure-containing-pointers, creating new heap variables for all the pointers in the structure? Maybe this cannot be done automatically? This is the problem that had me searching the IDL help for a STRUCT_COPY procedure.

Subject: Re: losing pointers when concatenating array of structures
Posted by Jim Pendleton on Mon, 09 Jan 2017 19:05:22 GMT
View Forum Message <> Reply to Message

On Monday, January 9, 2017 at 9:44:30 AM UTC-7, Edward Hyer wrote:
> On Monday, January 9, 2017 at 2:09:58 AM UTC-8, Heinz Stege wrote:
>> Well, pointers are tricky sometimes. I will try to explain.
>
> That is a good answer, thank you Heinz. So here's a question: what if I want to make a *copy* of a structure-containing-pointers, creating new heap variables for all the pointers in the structure? Maybe this cannot be done automatically? This is the problem that had me searching the IDL help for a STRUCT_COPY procedure.

A quick trick for making a *deep copy* is to SAVE the structure to an output file, then RESTORE the data, making sure you don't accidentally overwrite your original variable when you restore.

All included heap variable references (as well as all their dependencies) will be duplicated.

If you need a procedure that is not as deep, you will need to write your own logic. In ENVI, this is accomplished through dehydrate/rehydrate techniques.

Jim P.

Subject: Re: losing pointers when concatenating array of structures
Posted by Heinz Stege on Mon, 09 Jan 2017 20:33:05 GMT
View Forum Message <> Reply to Message

There is a deep_copy function in the JHU/APL IDL Library:
http://fermi.jhuapl.edu/idl/. Seems to me, that this is what Edward is
looking for.

More practicable than saving to a file and restoring it. ;-)

Good luck, Heinz
