Subject: Pointer Help - Referencing/Dereferencing in Functions & Procedures
Posted by Patrick Serengulian on Wed, 12 Mar 2003 20:40:20 GMT
View Forum Message <> Reply to Message

I'm new to this concept of newsgroups as well as the IDL language. I've been studying and writing IDL code for only six weeks. I have read L. Gumley's book 'Practical IDL Programming' cover to cover and still can't figure out what to do about pointers. From a co-worker, I borrowed Dr. Fanning's book, 'IDL Programming Techniques' and still no luck on the issue of referencing/dereferencing pointers.

I want to create a pointer in $MAIN$, reference and update the pointer in a function or procedure, return to $MAIN$ and echo the new value of the pointer. Here is the code I have written:

```
PRO NUMBER_PROC
  ; store 5L in the memory location that number_ptr is pointing to
  *number_ptr = 5L
END

;
; $MAIN$
;

; create number pointer using allocate heap
number_ptr = ptr_new(/allocate_heap)

; echo pointer information
print, 'Pointer created and this is the help info:'
help, *number_ptr

; store 17L in the mem location of the number pointer
*number_ptr = 17L

; call procedure to have pointer store a new number in the mem location
number_proc

; echo pointer information
print, *number_ptr
help,  *number_ptr, number_ptr

; destroy the pointer if it's still valid
if ( ptr_valid(number_ptr) EQ 1 ) then ptr_free, number_ptr

END
```

That's it. It compiles fine and then when I run the program, execution halts and I get the message "% Pointer type required in this context: NUMBER_PTR"

What am I doing wrong? I looked at a widget example from Gumley that uses pointers and it's almost identical in terms of how the pointer is dereferenced. Any information on how to correct this issue would be useful. Thank you.

Patrick Serengulian
Software Engineer,
U.S. Naval Research Laboratory,
Space Science Division,
Washington, D.C.

Subject: Re: Pointer Help - Referencing/Dereferencing in Functions & Procedures
Posted by Patrick Serengulian on Thu, 13 Mar 2003 13:45:02 GMT
View Forum Message <> Reply to Message

Thank you Mr. Smith and Chris for responding so promptly. I went home and gave it some thought. I don't think it's worth the hassle to figure out pointer in IDL. I had no problems using pointers in C/C++, but with IDL it's over my head. I don't have the time to properly learn IDL syntax on pointers. Rather than using the common block (aka global variables), I'm just going to create a structure to house all the variables I want to pass in and out of procedures. I think this will be the most efficient method to solve my issue of passing in and returning multiple variables. Thank you again for your help.

Patrick Serengulian
Software Engineer,
U.S. Naval Research Laboratory,
Space Science Division,
Washington, D.C.

"JD Smith" <jdsmith@as.arizona.edu> wrote in message
news:pan.2003.03.12.23.57.32.253437.6986@as.arizona.edu...
> On Wed, 12 Mar 2003 13:50:55 -0700, Chris wrote:
>
>> You need to pass an argument into your procedure; as far as it knows
>> "number_ptr" hasn't been declared.
>>
>> Change the first line of number_proc to:
>>
>> pro number_proc, number_ptr
>>
>> and the call in $MAIN$ to
>>
>> number_proc,number_ptr

>>
>>
>>
>> and it should work.
>>
>>
>> Chris
>
>
> Which is to say that, even though the heap of data to which a pointer
> points is available globally, the pointer itself is not. In fact, when
> you lose the pointer, but the heap data remains, this is a memory leak:
>
> IDL> a=ptr_new(fltarr(1000))
> IDL> a=1 ; uh oh, where's the pointer?
> IDL> help,/heap
> Heap Variables:
>   # Pointer: 1
>   # Object : 0
>
> <PtrHeapVar1>   FLOAT   = Array[1000]
>
> Here you see data on the global "pointer heap", but since you
> overwrote the pointer referring to it with "1", it's lost. It's still
> on the heap, but you just can't get to it (unless you know some arcane
> tricks). You can clean it up with:
>
> IDL> heap_gc,/verbose
> <PtrHeapVar1>   FLOAT   = Array[1000]
>
> That got rid of it. So, in order to use the data a pointer points to,
> you need to pass the pointer in as an argument, or perhaps save it in
> a common block so you can get to it from anywhere.
>
> JD
