Subject: Re: User selectable lower array bound?
Posted by btt on Mon, 06 Aug 2001 13:23:07 GMT
View Forum Message <> Reply to Message

Hello,

Sorry to be chiming in so late on this subject; I have a great excuse though. I have been completely absorbed by Liam's newly arrived book. It's wonderful! Yeah, Liam!

I got to thinking about the discussion that David, Pavel and JD had regarding objects when it hit me that you could 'roll-your-own' (I'm not a smoker so I hope I'm using that phrase properly).

I just whipped up an object to allow you to do just this kind of pseudo-indexing for 1d vectors. Be aware that I have included no error handling and haven't given this whole idea much thought (I'm still glassy eyed from following that object thread.) Also, be aware that I have given it a name that may prove to be a poor choice in the long run... but you should get the idea. By the way, the big idea here is that the defined structure persists as long as you keep the object going during a session... that's how it "remember's" the value of lower bound.

Now back to my book,

Ben

```
;--------START HERE
; EXAMPLE
; IDL> x = obj_new('findgen', 11, lower = -5)
; IDL> print, x->GetData([-4, 0, 5])
;      1.00000      5.00000      10.0000
; IDL> obj_destroy, x


;------
; GetData
;------
FUNCTION FINDGEN::GetData, Indices

Return, (*Self.Data) [Indices - Self.Lower]

END; GetData

;------
; SetProperty
;------
PRO FINDGEN::SetProperty, N = N, Lower = Lower
```

```
If n_elements(Data) NE 0 Then *Self.Data = Data

If n_elements(N) NE 0 Then Begin
  ;1d vectors only
 Self.N = N[0]
  ;either 'redefine' the variable or 'undefine' it
 If Self.N GT 0 Then *Self.Data = Findgen(Self.N) Else $
  Dummy = Size(Temporary(*Self.Data))
EndIf

If n_elements(Lower) NE 0 Then Self.Lower = Lower[0]

END ;SetProperty

;------
; GetProperty
;------
PRO FINDGEN::GetProperty, Data = Data, N = N, Lower = Lower

If Arg_present(Data) Then Data =  *Self.Data
N = Self.N
Lower = Self.Lower

END ;GetProperty

;------
; Initialization
;------
FUNCTION FINDGEN::INIT, N, LOWER = lower

If n_elements(N) EQ 0 Then Self.N = 0L Else Self.N = 0L > N[0]
If n_elements(lower) EQ 0 Then Self.Lower = 0L Else Self.Lower = Lower[0]

If Self.N GT 0 Then Self.Data = Ptr_NEW(Findgen(Self.N)) Else $
 Self.Data = Ptr_NEW(/Allocate)

Return, 1
END ;Init


;-----
; CleanUp
;-----
PRO FINDGEN::CleanUp

If Ptr_Valid(Self.Data) Then Ptr_Free, Self.Data
```

```
END ;CleanUp


;-----
; Definiton
;------
PRO FINDGEN__DEFINE

Struct = {FINDGEN, $

 Data: ptr_new(), $ ; the data array
 N: 0L, $   ;this handles only 1d arrays right now
 Lower: 0L}   ;the indexed address of the lower bound

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

Jeff Guerber wrote:
>
> On Thu, 2 Aug 2001, Paul van Delst wrote:
>
>> Is is just me, or would anyone else find useful the ability to define
>> arrays in IDL such that the lower bound is *not* always zero? Sorta
>> like:
>>
>>   x = FINDGEN(11, LOWER = -5)
>> or
>>   y = DBLARR(100, LOWER = 1)
>>
>> so that accessing elements such as x[-4] or y[100] are o.k.? [...]
>
>   Here, here!! This was #1 on my (13-item) contribution to last summer's
> "Top 10 IDL Requests" discussion. As I pointed out then, Fortran's had
> this capability for decades. (And IDL is expressly a data-analysis
> language, like Fortran, not a systems-programming language like C.) The
> biggest problem I see is that certain IDL intrinsics, like WHERE(), return
> -1 to indicate an invalid index. Perhaps WHERE could return
> (lowerbound-1) instead, on the presumption that existing programs would be
> using 0-based arrays? Of course it's much better to check the COUNT=
> keyword anyway. (This would also be a good application for some sort of
> "undefined value" type.)
>
>   (IMHO, the two worst features IDL picked up from (presumably) C are
> starting arrays at 0 (which makes some sense in C, due to the tight
> coupling of arrays and pointers, but this isn't the case in IDL), and
> prefix syntax for the array dereferencing operator. Concerning the
> latter, I've since learned that even Dennis Ritchie apparently now thinks
> it was a mistake; from his "The Development of the C Language"

> (http://cm.bell-labs.com/who/dmr/chist.pdf), page 12:
>
> An accident of syntax contributed to the perceived complexity of
> the language. The indirection operator, spelled * in C, is
> syntactically a unary prefix operator, just as in BCPL and B [C's
> predecessor languages]. ... Sethi [Sethi 81] observed that many of
> the nested declarations and expressions would become simpler if the
> indirection operator had been taken as a postfix operator instead
> of prefix, but by then it was too late to change.
>
> Oh, make that THREE worst features: Also the use of integers for
> Boolean data, instead of having a true logical or Boolean data type.
> VERY confusing.)
>
> Of course, these opinions are my own and don't reflect those of
> Raytheon or NASA.
>
> Jeff Guerber
> Raytheon ITSS
> NASA Goddard Space Flt Ctr
> Oceans & Ice Branch (971)

--
Ben Tupper
Bigelow Laboratory for Ocean Sciences
180 McKown Point Rd.
W. Boothbay Harbor, ME 04575
btupper@bigelow.org