Subject: Re: User selectable lower array bound?
Posted by Jeff Guerber on Fri, 03 Aug 2001 00:39:50 GMT
View Forum Message <> Reply to Message

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)

Subject: Re: User selectable lower array bound?
Posted by bennetsc on Fri, 03 Aug 2001 00:52:37 GMT
View Forum Message <> Reply to Message

In article <3B69CA57.FD3B1D8D@noaa.gov>,
Paul van Delst <paul.vandelst@noaa.gov> wrote:
> Hey there,
>
> 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.?

    Yes, that would make a lot of code much more understandable
and less prone to errors during development.
>
> I know this can be done now with judicious use of proxy indices, e.g.
>
>   FOR i = -5, 5 DO BEGIN
>   ix = i + 5
>   PRINT, x[ix]
>   ....do other stuff with negative i's....
>   ENDFOR
>
> but sometimes this makes code hard to follow (or explain to
> someone who's never used the
> code before) in direct correspondence with a physical process.
>
> It seems like such a simple thing to be able to do (with default
> action being start at
> index 0) although I'm sure the amount of work required to
> implement this would be
> horrendous. Still, it shur would be nice.....
>
    That depends upon how IDL already keeps track of arrays

internally. In PL/1, for example, one declared an array with the boundaries for each dimension in the form lowerbound:upperbound, where specification of the lower bound and the colon were optional. If only the upper bound were specified, then the lower bound defaulted to 1. In its internal representation of arrays, IIRC, PL/1 kept the lower and upper boundaries of each dimension as part of a control block preceding the actual array memory. If a language implementation doesn't already store both boundaries, or equivalently, the lower boundary and number of elements, for each dimension, then yes, adding such support might well be a major headache.

Scott Bennett, Comm. ASMELG, CFIAG
College of Oceanic and Atmospheric Sciences
Oregon State University
Corvallis, Oregon 97331

****

* Internet: sbennett at oce.orst.edu *

---------- ---------

* "Lay then the axe to the root, and teach governments humanity. *
* It is their sanguinary punishments which corrupt mankind." *
* -- _The_Rights_of_Man_ by Tom Paine (1791.) *

****

Subject: Re: User selectable lower array bound?
Posted by Paul van Delst on Fri, 03 Aug 2001 14:08:24 GMT
View Forum Message <> Reply to Message

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.)

Well, maybe WHERE could work as it does now, but for cases where the start index is not zero, a function like the Fortran 90 intrinsic LBOUND() could be used.

BTW, I never check the WHERE result either, always the COUNT value.

paulv

--
Paul van Delst A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP Drink deep, or taste not the Pierian spring;
Ph: (301)763-8000 x7274 There shallow draughts intoxicate the brain,
Fax:(301)763-8545 And drinking largely sobers us again.
Alexander Pope.

Subject: Re: User selectable lower array bound?
Posted by Paul van Delst on Fri, 03 Aug 2001 14:36:33 GMT
View Forum Message <> Reply to Message

bennetsc@NOSPAMucs.orst.edu wrote:
>
> In article <3B69CA57.FD3B1D8D@noaa.gov>,
> Paul van Delst <paul.vandelst@noaa.gov> wrote:
>> Hey there,
>>
>> 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.?
>
> Yes, that would make a lot of code much more understandable
> and less prone to errors during development.

Tell me about it! :o)

>>

>> I know this can be done now with judicious use of proxy indices, e.g.
>>
>>   FOR i = -5, 5 DO BEGIN
>>   ix = i + 5
>>   PRINT, x[ix]
>>   ....do other stuff with negative i's....
>>   ENDFOR
>>
>> but sometimes this makes code hard to follow (or explain to
>> someone who's never used the
>> code before) in direct correspondence with a physical process.
>>
>> It seems like such a simple thing to be able to do (with default
>> action being start at
>> index 0) although I'm sure the amount of work required to
>> implement this would be
>> horrendous. Still, it shur would be nice.....
>>
>   That depends upon how IDL already keeps track of arrays
> internally. In PL/1, for example, one declared an array with the
> boundaries for each dimension in the form lowerbound:upperbound,
> where specification of the lower bound and the colon were optional.
> If only the upper bound were specified, then the lower bound defaulted
> to 1. In its internal representation of arrays, IIRC, PL/1 kept
> the lower and upper boundaries of each dimension as part of a control
> block preceding the actual array memory. If a language implementation
> doesn't already store both boundaries, or equivalently, the lower
> boundary and number of elements, for each dimension, then yes, adding
> such support might well be a major headache.

One big problem that occurred to me was how one would implicitly or explicitly specify the array bounds over a procedure or function call in IDL.

Consider the following Fortran 90 code:

```
  program test_bounds

    integer, parameter :: n = 20
    real, dimension( 0:n ) :: x
    integer :: i

    ! -- Fill the array (like FINDGEN)
    x = (/ (real(i),i=0,n) /)

    print *, 'In Main'
    print *, 'LBOUND(x)=',LBOUND( x )
    print *, 'UBOUND(x)=',UBOUND( x )
    print *, 'SIZE(x)  =',SIZE( x )
```

```
   call sub( x )

 contains

   subroutine sub( sx )

     ! -- Asummed shape dummy argument
     real, dimension( : ) :: sx

     print *, 'In Sub'
     print *, 'LBOUND(sx)=',LBOUND( sx )
     print *, 'UBOUND(sx)=',UBOUND( sx )
     print *, 'SIZE(sx)  =',SIZE( sx )

   end subroutine sub

 end program test_bounds
```

The results of which are:

```
 In Main
 LBOUND(x)=          0
 UBOUND(x)=         20
 SIZE(x)  =         21
 In Sub
 LBOUND(sx)=          1
 UBOUND(sx)=         21
 SIZE(sx)  =         21
```

So the upper and lower bounds as declared in the "Main" program are by default not preserved when passing arrays unless your subroutine declaration of "sx" is

```
  real, dimension( 0: ) :: sx
```

i.e. from index 0->however-big-the-array-is minus 1.

So you can specify whether you wanted the lower bound of sx in Sub to be 0 or 1 (or anything else for that matter). This seems like a simple thing but it can be a tremendously useful feature. I don't know how you would replicate that in IDL since you don't declare stuff in procedures/functions.

Hmmm.

paulv

--

Paul van Delst
CIMSS @ NOAA/NCEP
Ph: (301)763-8000 x7274
Fax:(301)763-8545

A little learning is a dangerous thing;
Drink deep, or taste not the Pierian spring;
There shallow draughts intoxicate the brain,
And drinking largely sobers us again.
Alexander Pope.

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

Subject: Re: User selectable lower array bound?
Posted by david[2] on Mon, 06 Aug 2001 14:45:58 GMT
View Forum Message <> Reply to Message

Ben Tupper writes:

> I just whipped up an object to allow you to do just this kind of
> pseudo-indexing for 1d vectors.

Oh, sure, you can do it with objects. But
isn't that, like, cheating? In any case,

scientists are too busy to learn yet one
more useful programming practice.

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: User selectable lower array bound?
Posted by Jeff Guerber on Thu, 09 Aug 2001 06:39:38 GMT
View Forum Message <> Reply to Message

On 3 Aug 2001 bennetsc@NOSPAMucs.orst.edu wrote:

>> It seems like such a simple thing to be able to do (with default
>> action being start at
>> index 0) although I'm sure the amount of work required to
>> implement this would be
>> horrendous. Still, it shur would be nice.....
>>
> That depends upon how IDL already keeps track of arrays
> internally. In PL/1, for example, one declared an array with the
> boundaries for each dimension in the form lowerbound:upperbound,
> where specification of the lower bound and the colon were optional.
> If only the upper bound were specified, then the lower bound defaulted
> to 1. In its internal representation of arrays, IIRC, PL/1 kept
> the lower and upper boundaries of each dimension as part of a control
> block preceding the actual array memory. If a language implementation
> doesn't already store both boundaries, or equivalently, the lower
> boundary and number of elements, for each dimension, then yes, adding
> such support might well be a major headache.

Well, IDL does perform bounds checking, even for arrays passed into a procedure as arguments, so it must already store at least either the upper bound or the number of elements (which are equivalent since the lower bound is fixed). It's likely that this is only done in one place, so implementing lower bounds in the IDL core might not be all _that_ much work. HOWEVER...

Having thought about this further, I now think the more serious problem would be all the library procedures (and not just RSI's!) that assume you can loop over the elements of any array by going from 0 to n_elements(array)-1. (Aiiigh!) Unless the bounds are lost across procedure calls (as Paul pointed out that Fortran does), which can sometimes be useful but which kind of defeats the point of having definable bounds, if you ask me.

Jeff Guerber
Raytheon ITSS
NASA Goddard Space Flight Ctr
Oceans & Ice Branch (code 971)
