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   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.