Subject: Re: User selectable lower array bound?
Posted by John-David T. Smith on Tue, 07 Aug 2001 15:24:58 GMT
View Forum Message <> Reply to Message

bennetsc@NOSPAMucs.orst.edu wrote:
>
> In article <3B6F037D.B17A6287@astro.cornell.edu>,
> JD Smith <jdsmith@astro.cornell.edu> wrote:
>> Paul van Delst wrote:
>>>
>>> "Pavel A. Romashkin" wrote:
>>>>
>>>> Craig Markwardt wrote:
>>>> >
>>>> > Well, as grumpy as I have been in the past about IDL
>> wishes, this is
>>>> > one thing I do not want to have in IDL now!
>>>>
>>>> I am with you Craig. Besides, for the purists of array
>> indexing, I think
>>>> it is unfair to dasignate a *lower* array bounds. We don't
>> designate the
>>>> *upper* one.
>>>
>>> In the context of initially declaring an array in IDL, sure you do:
>>>
>>> x = fltarr(10)
>>>
>>> declares the upper bound as 9. We also designate a lower
>> bound: 0. The difference between
>>> the two is that I can change the former.
>>>
> So how about if {flt,dbl,complex,int,lon,dcomplex,byt,str}arr
> and make_array could accept both the form shown above and this form:
>
> y = fltarr(-5:10)
>
> which would declare the lower bound as -5 and the upper bound as 9,
> giving a total of 16 elements, including the zero element? This
> isn't quite as nice as PL/1's method because of the zero element,
> but it would be usable and wouldn't break any existing code. Future
> programs would have to take into consideration that
>
> y = fltarr(-5:-1)
>
> would have a lower bound of -5 and an upper bound of -1, giving
> a total of only 5 elements due to the lack of a zero element. PL/1's
> syntax avoided this problem by having the lower bound default to 1

> if not coded, but I think I could live with it as long as I were
> aware of it.

I guess I was not so concerned with the syntax of indexing, as with the capability of extending lists in two directions without jumping through hoops.

Think Perl. For example, to add an element or elements to the end of a possibly non-existent list, you say:

```
push @list, $elem;
```

To pull one element off the end:

```
pop @list;
```

To pull one off the front:

```
shift @list;
```

To push one onto the front:

```
unshift @list, $elem;
```

With such a collection we could rid ourselves of all those silly statements like:

```
if n_elements(list) eq 0 then list=[elem] else list=[list, elem]
```

As for actual negative indices, I think that's probably an impossible idea to implement at this point, given how hardcoded the notion of zero offset arrays is in so many processing routines, both built-in and otherwise.

This all harkens back to the notion of zero-length or null vectors, e.g. as a return from where(), and as a possibility for indexing arrays. There were other technical difficulties with implementing this idea, but I find more instances where it would have been useful all the time.

JD