Subject: subscript array question

Posted by bennetsc on Thu, 11 Feb 1999 08:00:00 GMT

View Forum Message <> Reply to Message

I'm using IDL 5.0 and need to be able to use a subscript
array containing duplicate values like this:

```
array = intarr(5)
subs  = [0,2,4,4]
array[subs] = array[subs] + 1
```

and have the resulting values for array be:

```
 1 0 1 0 2
```

Because of the way IDL manages memory for expression evaluation
and assignments, what happens for the last two elements of the
addition is that the original value of array[4] is used twice,
rather than what I want, which is to use the current value of
array[4] each time. I.e. IDL gives the resulting values for
array to be:

```
 1 0 1 0 1
```

So is there a way to do what I want without resorting to
a loop? In my real-world application, I'm using two different
subscript arrays together to index into and update a two-dimensional table. Having duplicate pairs of subscripts from
the two subscript arrays is expected to occur very frequently.

Scott Bennett, Comm. ASMELG, CFIAG
Dept. of Atmospheric Sciences
Oregon State University
Corvallis, Oregon 97331

** **********
* Internet: sbennett@oce.orst.edu   *
-- ---------
* "The jury has a right to judge both the law as well as the fact in *
* controversy."--John Jay, First Chief Justice, U.S. Supreme Court   *
* in Georgia vs. Brailsford, 1794   *
** **********

Subject: Re: subscript array question

Posted by Craig Markwardt on Tue, 16 Feb 1999 08:00:00 GMT

View Forum Message <> Reply to Message

Tom McGlynn <tam@silk.gsfc.nasa.gov> writes:

>
> This kind of thing comes up in a number of contexts, e.g., the
> thread a couple of months ago for calculating cumulative totals
> in arrays. The solution I've suggested to RSI is a new equality
> operator (e.g., :=) which does not implicitly parallelize
> array operations.
>
> In this case one could just write
>
>   array[subs] := array[subs]+1
>
> in a very natural way rather than use devious if clever subterfuges.
>

Yorick, which is very IDL-like, has some pretty handy ideas in it. While it doesn't consider your "anti-parallel" equality operator, it does have operators that *attach* to array subscripts and do most of the things you might want. Consider that there is no nicely vectorized function in IDL which computes the cumulative total of "x" in IDL. In Yorick, you would say:

y = x(cum)

Here "cum" is a special function which operates on a dimension of the array, and cranks out the cumulative total for you. Thus for example,

y = x(dif)   is the finite difference between elements of x
            (this is something I wish I had all the time!)

There are also "rank-reducing" functions which remove one dimension of the array. Thus,

y = x(sum)   is the same as y = total(x) in IDL,
y = x(max)   is the same as y = max(x), etc.

Whis is this nice? Sometimes you want to get the maximum along only a certain dimension. To get the maximum in each row of an array, you might try:

y = x(max,*) which has no equivalent in IDL

As I say, I find myself wishing for a lot of these features almost daily. IDL could become a whole lot more vector-friendly with them. The RSI people could take some lessons from Yorick.

Craig

--
 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@astrog.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: subscript array question
Posted by Thomas A. McGlynn on Tue, 16 Feb 1999 08:00:00 GMT
View Forum Message <> Reply to Message

This kind of thing comes up in a number of contexts, e.g., the thread a couple of months ago for calculating cumulative totals in arrays. The solution I've suggested to RSI is a new equality operator (e.g., :=) which does not implicitly parallelize array operations.

In this case one could just write

```
    array[subs] := array[subs]+1
```

in a very natural way rather than use devious if clever subterfuges.

This simple syntax can solve a variety of problems rather cleanly. E.g., the cumulative total looks something like:

```
    cum[1:n-1] := cum[0:n-2] + cum[1:n-1]
```

Although the loops would not be parallelized, they could still be compiled into very efficient code (since temporaries would be less necessary perhaps faster in some cases than the parallel versions).

Haven't heard anything back alas...

 Tom McGlynn
 tam@silk.gsfc.nasa.gov