Subject: Re: Looping over parameters without EXECUTE()
Posted by JD Smith on Tue, 03 May 2005 01:57:45 GMT
View Forum Message <> Reply to Message

On Mon, 02 May 2005 12:10:43 -0400, Wayne Landsman wrote:

```
>  The one case where I haven't figured out how to remove EXECUTE() from a
>  program (to allow use with the Virtual Machine) is where one wants to
>  loop over supplied parameters.     For example, to apply the procedure
>  'myproc' to each supplied parameter (which may have different data
>  types) one can use EXECUTE() to write each parameter to a temporary
>  variable:
>
>  ***********************************************
>
>  pro doit,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11
>
>  ;Loop over input parameters
>  Np = N_params()
>  colname = 'p' + strtrim(indgen(Np)+1,2)
>
>  for i=0,Np-1 do begin
>     result = execute('p=' + colname[i] )
>     myproc,p
>  endfor
>  *********************************************
>  Is there a way to avoid EXECUTE() here -- say to identify the 4th
>  parameter as e.g., $4 ?     Of course, one can always avoid the loop and
>  explicitly write out the call for each parameter:
>
>  myproc,p1
>  myproc,p2
>  ....
>  but this probably becomes unreasonable at around 20 parameters.
>
>  One solution is to have the program read an array of pointers rather
>  than multiple parameters.    But this has the disadvantages of losing
>  backwards compatibility, as well as making the program somewhat more
>  complicated to use.     My current default solution is to make a pointer
>  keyword available and say that data must be passed this way instead of
>  via parameters, if the user wants to use the VM.
>
>  Thanks, --Wayne
```

I use a big cascading SWITCH statement which I generate with a little perl script. It works well when you are accumulating things based on the arguments:

```
switch n_params() of
 10: print,v10
  9: print,v9
  ...
  1: print,v1
  0: break
  else: message,'No more than 10 params allowed'
endswitch
```

It will cascade through all existing parameters, and can be used to accumulate the arguments as well. But for long argument lists, it stands out in your code like a sore thumb.

Here's another thought: why not use a set of convenience routines to grab all parameters and package them into an appropriately sized pointer list for the internal consumption of the routine, so that you could shield the user from the pointer symantics, but not have to deal with all those case/switch statements? Also, we want arbitrary input/output options for each variable.

A similar cascading switch with incremented pointer assignment should work. However, that would still leave large explicit v1,v2,v3,...,v50 lists and big switch statements lying around in your code like some sad FORTRAN port. Ugly and hard to manage. So, let's say you really want to class this up, and keep your code neat and clean, with nary a vXXX in site. How about something as simple as this:

```
pro test_args,$
@package_args_list

@package_args

  for i=0,n_elements(args)-1 do $
     *args[i]=42*randomu(sd)

@unpackage_args
end
```

Have a look at:

 turtle.as.arizona.edu/idl/package_args

for the code.

It looks complicated, but basically just uses @batch import to hide the semantics of converting between a long list of arguments and a list of pointers, and back again. It checks for valid arguments (availing

itself of the "check your assumptions" trick at the end of
http://www.dfanning.com/tips/keyword_check.html), puts them on a pointer
list without copying the data, lets you operate on that list
(read/write), and then unpacks the pointers back onto the passed
variable (using TEMPORARY to save memory copying) and finally frees the
intermediary argument pointer array. I have it set up for a maximum of
51 arguments, but it could easily be expanded.

Is this IDL's version of loop unrolling? I think so.

JD

P.S. Reimar's SCOPE_VARFETCH method is nice, but requires v6.1, and also
requires you to test each incoming variable explicitly to see if it was
set. It also would be very cumbersome to *store* values in the passed
arguments (though it can be done). On the other hand, my method can
leave pointer data around if you have an error and don't explicitly
CATCH it.