Subject: Looping over parameters without EXECUTE()
Posted by Wayne Landsman on Mon, 02 May 2005 16:10:43 GMT
View Forum Message <> Reply to Message

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

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.

Subject: Re: Looping over parameters without EXECUTE()
Posted by Thomas Pfaff on Tue, 03 May 2005 13:43:33 GMT

View Forum Message <> Reply to Message

```
JD Smith schrieb:
>  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.
>

How about putting all those parameters into a (named or anonymous)
struct? Then you can have different types for each parameter and you're
still able to loop over the elements.

```
pro doit, param_struct
  for i=0, n_tags(param_struct) -1 do begin
    arg = param_struct.(i) ;->this way you can even store result values
    myproc, arg
    param_struct.(i) = arg
  endfor
```

end

Would that be a possibility, or am I missing something?

Cheers,

Thomas

Subject: Re: Looping over parameters without EXECUTE()
Posted by JD Smith on Tue, 03 May 2005 17:41:12 GMT
View Forum Message <> Reply to Message

On Tue, 03 May 2005 15:43:33 +0200, Thomas Pfaff wrote:

```
>
>
> JD Smith schrieb:
>> 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:

>
> How about putting all those parameters into a (named or anonymous)
> struct? Then you can have different types for each parameter and you're
> still able to loop over the elements.
>
> pro doit, param_struct
>   for i=0, n_tags(param_struct) -1 do begin
>     arg = param_struct.(i) ;->this way you can even store result values
>     myproc, arg
>     param_struct.(i) = arg
>   endfor
> end
>
> Would that be a possibility, or am I missing something?
```

That works OK, and if you used TEMPORARY() you could cut down the data copying penalty. There are two main problems with this approach: 1) the user has to create a potentially large struct as input and then unpack it as output, which is not convenient from the command line, especially for

output arguments, and 2) the type and size of each argument cannot be
changed, thanks to the nature of structs.

JD

Subject: Re: Looping over parameters without EXECUTE()
Posted by R.Bauer on Tue, 03 May 2005 17:47:13 GMT
View Forum Message <> Reply to Message

>
> 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.
>

It's a parameter list so you have only to test until a parameter isn't
known. I think a while loop and a catch will pretty do the job and this
should be included by e.g. @paramtest, Probably it could transform the
data in a structure. So then you could use n_tags() instead of
n_params() and positional tags .(0-n) in the code.
In the opposite direction it is descripted by Thomas.

If we use @paramtest we have only to change the internal assignment of
variables exchanging the execute statement and not the calling sequence.
So it does not affect current code calling these routines without vm.

cheers

Reimar

--
Reimar Bauer

Institut fuer Stratosphaerische Chemie (ICG-I)
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de
 -- -------
        a IDL library at ForschungsZentrum Juelich
   http://www.fz-juelich.de/icg/icg-i/idl_icglib/idl_lib_intro. html
 == =======

Subject: Re: Looping over parameters without EXECUTE()
Posted by R.Bauer on Tue, 03 May 2005 18:00:28 GMT
View Forum Message <> Reply to Message

JD Smith wrote:

```
>  On Tue, 03 May 2005 15:43:33 +0200, Thomas Pfaff wrote:
>
>
>>
>> JD Smith schrieb:
>>
>>> 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:
>
>
>> How about putting all those parameters into a (named or anonymous)
>> struct? Then you can have different types for each parameter and you're
>> still able to loop over the elements.
>>
>> pro doit, param_struct
>>   for i=0, n_tags(param_struct) -1 do begin
>>     arg = param_struct.(i) ;->this way you can even store result values
>>     myproc, arg
>>     param_struct.(i) = arg
>>   endfor
>> end
>>
>> Would that be a possibility, or am I missing something?
>
>
>  That works OK, and if you used TEMPORARY() you could cut down the data
>  copying penalty.  There are two main problems with this approach: 1) the
>  user has to create a potentially large struct as input and then unpack it
>  as output, which is not convenient from the command line, especially for
>  output arguments, and 2) the type and size of each argument cannot be
>  changed, thanks to the nature of structs.
>
>  JD
```

You could change the type and size of each argument of a struct with

some of our library tools. e.g. replace_tagvalue. But your are right
this is done with a trick by duplicating the structure in a new one.
But in this case it is overkill.

Reimar

--
Reimar Bauer

Institut fuer Stratosphaerische Chemie (ICG-I)
Forschungszentrum Juelich
email: R.Bauer@fz-juelich.de
 -- -------
        a IDL library at ForschungsZentrum Juelich
   http://www.fz-juelich.de/icg/icg-i/idl_icglib/idl_lib_intro. html
 == =======

Subject: Re: Looping over parameters without EXECUTE()

Posted by Ken Mankoff on Tue, 03 May 2005 23:31:43 GMT

View Forum Message <> Reply to Message

On Tue, 3 May 2005, JD Smith wrote:
> On Tue, 03 May 2005 15:43:33 +0200, Thomas Pfaff wrote:
>> JD Smith schrieb:
>>> 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:
>>
>> How about putting all those parameters into a (named or
>> anonymous) struct? Then you can have different types for each
>> parameter and you're still able to loop over the elements.
>>

```
>> pro doit, param_struct
>>   for i=0, n_tags(param_struct) -1 do begin
>>     arg = param_struct.(i) ;->this way you can even store result values
>>     myproc, arg
>>     param_struct.(i) = arg
>>   endfor
>> end
```

>>
>> Would that be a possibility, or am I missing something?
>
> That works OK, and if you used TEMPORARY() you could cut down the
> data copying penalty. There are two main problems with this
> approach: 1) the user has to create a potentially large struct as
> input and then unpack it as output, which is not convenient from
> the command line, especially for output arguments, and 2) the type
> and size of each argument cannot be changed, thanks to the nature
> of structs.

The user doesn't need to build the structure if you build it for
them. I've taken this approach to allow a small subset of
command-line interactivity with the IDL VM. It allows you to run
procedures that only use keywords. That is, no functions, and no
positional arguments. The only IDL procedure I know of like this is
  ERASE [, COLOR=value]

It is quite a hack, and at this point _very_ inefficient (each
keyword copies the struct with e = CREATE_STRUCT(name,value,e). And
I wouldn't deploy it in a large codebase. But it works for small
stuff...

Another major limitation is everything gets turned into a string.
Most of the time this is OK... For example,
  erase, color='128'
works ok... But every once in a while it causes problems.

The code can be called like this:

```
IDL> .com eextra
IDL> eextra_test
IDLVM> erase, color=128 ; example command with only keyword args
IDLVM> quit
IDL>
```

And here is the eextra (and helper) functions:

   -k.

--
http://spacebit.dyndns.org/

```
function eextra_build, arr
arr = STRTRIM(arr,2)
e = CREATE_STRUCT("IDLVM_NIL",0B) ; filler
if SIZE( arr, /TNAME ) NE "STRING" then return, e
if SIZE( arr, /N_ELEMENTS ) EQ 1 then begin ; not an array, just 1 string
    if ( STREGEX( arr,"^/", /BOOLEAN ) ) then begin ; /foo
       e = CREATE_STRUCT( STRMID( arr[0], 1 ), 1, e )
    endif else if STREGEX( arr, "=", /BOOLEAN ) then begin ; foo = 42
       nv = STRTRIM(STRSPLIT( arr, "=", /extract ),2)

       ; this line so that x=0 makes x=0b, not x='0' (which is 1)
       if (byte(nv[1]))[0] eq 48 then vv = 0B else vv = nv[1]
       e = CREATE_STRUCT( nv[0], vv, e )
    endif else begin
       ; nothing here. String was 'foo', it is an ARG, not an _EXTRA=e
    endelse
    return, e
endif
extras = stregex(arr,"(.*)[=/](.*)",/extract)
blank = where( extras eq '', nblank, ncomp=ncomp )
if ncomp eq 0 then return, CREATE_STRUCT("IDLVM_NIL",0B)
if nblank ne 0 then extras = extras[where(extras ne '')]

; convert "/foo" to "foo=1"
one = STREGEX(extras,"^/",/BOOLEAN)
;one = STRCMP(arr,"/",1)
if total(one) ne 0 then begin ; some /keywords were set
    bool = STREGEX(extras,"/(.*)", /extract)
    bool = STRMID(bool,1) + "=1"; + STRING(1)
    extras[where(one eq 1)]=bool[where(one eq 1)]
endif

; name/value pairs from form foo=42;
; nv[ 1, * ] = foo & nv[ 2, * ] = 42
;nv = STREGEX(extras,"(.*)=([0-9a-zA-Z-]*)",/subexpr,/extract)
nv = STRTRIM(STREGEX(extras,"(.+)=(.+)",/subexpr,/extract),2)
n = STRTRIM(reform( nv[ 1, * ] ),2) ; allow spaces i.e. foo =  42
v = STRTRIM(reform( nv[ 2, * ] ),2)
zero = where( v eq '', nz )
if nz gt 0 then v[zero]='0'

;stop
for i = 0, n_elements( n )-1 do begin
; attempt to convert numbers to numbers (right now everything is strings)
    if (byte(v[i]))[0] eq 48 then vv = 0B else vv = v[i]
    e = CREATE_STRUCT(n[i],vv,e)
;   if ( v[i] eq FLOAT( v[i] ) ) then $
;      e = CREATE_STRUCT(n[i],FLOAT(v[i]),e) else $
;        e = CREATE_STRUCT(n[i],v[i],e)
endfor
return, e
```

```
end



function eextra_clean, e
for i = 0, n_tags(e)-1 do begin
    s = e.(i)
    if size(s,/tname) eq 'STRING' then begin
       fix = STREGEX(s,'"(.*)"',/SUBEXPR,/EXTRACT)
       if fix[0] NE '' then e.(i) = fix[1]
       fix = STREGEX(s,'"(.*)"',/SUBEXPR,/EXTRACT)
       if fix[0] NE '' then e.(i) = fix[1]
    endif
endfor
return, e
end

pro eEXTRA_test
input = ''
while (1) do begin
    read, input, PROMPT="IDLVM> " ; get the proc and all keyword arguments
    cmds = STRTRIM(STRSPLIT( input, ",", /EXTRACT ),2)
    cmd = cmds[0]
    if ( n_elements(cmds) GE 2 ) then $
      e = eEXTRA( cmds[1:*] ) else $
       e = CREATE_STRUCT("IDLVM_NIL",0B) ; null filler
    case cmd of
       'exit': exit
       'stop': stop
       'quit': goto, done
       else: CALL_PROCEDURE, cmd, _EXTRA=e
    endcase
endwhile
done:
end

function eEXTRA, arr
e = eextra_build( arr )
e = eextra_clean( e )
return, e
end
```