Subject: Re: simple question (I hope)
Posted by wlandsman@jhu.edu on Fri, 30 Mar 2007 15:28:12 GMT
View Forum Message <> Reply to Message

"> could get quite large. Here is an example of what I would like to do:
>
> A = [0,2,4,6,8,10,12,14,16,18,20]
> indices_to_remove = [3,5,9]
>
> to get a resulting array, B:
> B = [0,2,4,8,12,14,16,20]

You might look at http://idlastro.gsfc.nasa.gov/ftp/pro/misc/remove.pro
which is set up to do this using HISTOGRAM.

Subject: Re: simple question (I hope)
Posted by Ryan. on Fri, 30 Mar 2007 15:32:27 GMT
View Forum Message <> Reply to Message

> You might look athttp://idlastro.gsfc.nasa.gov/ftp/pro/misc/remove.pro
> which is set up to do this using HISTOGRAM.

Thanks Wayne!

That's exactly what I needed.

Ryan.

Subject: Re: simple question (I hope)
Posted by Fil. on Fri, 30 Mar 2007 15:38:41 GMT
View Forum Message <> Reply to Message

Ryan. wrote:
> Dear All,
>
> Do any of you know a fast way of removing elements from an array given
> an array of the indices? I know it is possible with a FOR loop but I
> would like to avoid that if possbile because the array to be searched
> could get quite large. Here is an example of what I would like to do:
>
> A = [0,2,4,6,8,10,12,14,16,18,20]
> indices_to_remove = [3,5,9]
>
> to get a resulting array, B:
> B = [0,2,4,8,12,14,16,20]

>
> Note: I don't find the indices to remove using the WHERE function so I
> am unable to use the COMPLEMENT option.
>
> I think the 2.5 hours at All-You-Can-Eat Sushi last night has affected
> my thinking because I'm still digesting.
>
> Thanks,
> Ryan.
>

What about:

A[indices_to_remove] = -454 ; or some other value different
than any value in A
ind = where(A ne -454, count)
if count then B = A(ind)

Fil.

Subject: Re: simple question (I hope)
Posted by Ryan. on Fri, 30 Mar 2007 15:54:07 GMT
View Forum Message <> Reply to Message

Hi Wayne,

I have one more question about it, but it is more about how IDL works
than the REMOVE routine.
Say for example I do this:

group_array = huge_array[groupidx]
indices_2_remove_in_group_array = [...]

And If I call the REMOVE routine
REMOVE, indices_2_remove_in_group_array, huge_array[groupidx]

Will this call remove the elements from the *huge_array* or will it
remove them from a temporary array created when calling the REMOVE
routine?

I know that IDL passes references as arguments, but in this will it
actually remove the elements from the original *huge_array* or not.

Thanks,
Ryan.

Subject: Re: simple question (I hope)
Posted by David Fanning on Fri, 30 Mar 2007 16:48:59 GMT
View Forum Message <> Reply to Message

Wayne Landsman writes:

> "> could get quite large. Here is an example of what I would like to do:
>>
>> A = [0,2,4,6,8,10,12,14,16,18,20]
>> indices_to_remove = [3,5,9]
>>
>> to get a resulting array, B:
>> B = [0,2,4,8,12,14,16,20]
>
> You might look at http://idlastro.gsfc.nasa.gov/ftp/pro/misc/remove.pro
> which is set up to do this using HISTOGRAM.

These, and other HISTOGRAM tricks, can always be found
in the infamous Histogram Tutorial:

http://www.dfanning.com/tips/histogram_tutorial.html

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: simple question (I hope)
Posted by David Fanning on Fri, 30 Mar 2007 17:00:25 GMT
View Forum Message <> Reply to Message

Ryan. writes:

> I have one more question about it, but it is more about how IDL works
> than the REMOVE routine.
> Say for example I do this:
>
> group_array = huge_array[groupidx]
> indices_2_remove_in_group_array = [...]
>
> And If I call the REMOVE routine
> REMOVE, indices_2_remove_in_group_array, huge_array[groupidx]
>

> Will this call remove the elements from the *huge_array* or will it
> remove them from a temporary array created when calling the REMOVE
> routine?
>
> I know that IDL passes references as arguments, but in this will it
> actually remove the elements from the original *huge_array* or not.

Actually, IDL passes *variables* by reference. Everything
else, including expressions like "huge_array[groupidx]", it
passes by value. So if you called REMOVE like this, you
would get no error messages, since it would work, but
you wouldn't know about it. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: simple question (I hope)
Posted by JD Smith on Fri, 30 Mar 2007 17:51:02 GMT
View Forum Message <> Reply to Message

On Fri, 30 Mar 2007 10:00:25 -0700, David Fanning wrote:

> Ryan. writes:
>
>> I have one more question about it, but it is more about how IDL works
>> than the REMOVE routine.
>> Say for example I do this:
>>
>> group_array = huge_array[groupidx]
>> indices_2_remove_in_group_array = [...]
>>
>> And If I call the REMOVE routine
>> REMOVE, indices_2_remove_in_group_array, huge_array[groupidx]
>>
>> Will this call remove the elements from the *huge_array* or will it
>> remove them from a temporary array created when calling the REMOVE
>> routine?
>>
>> I know that IDL passes references as arguments, but in this will it
>> actually remove the elements from the original *huge_array* or not.

>
> Actually, IDL passes *variables* by reference. Everything
> else, including expressions like "huge_array[groupidx]", it
> passes by value. So if you called REMOVE like this, you
> would get no error messages, since it would work, but
> you wouldn't know about it. :-)

This isn't quite correct. Everything, and I mean everything, in IDL is passed by reference. However, when IDL encounters a statement like `array[x]', or `struct.y', or total(array), it first creates a temporary variable to hold the results of the array indexing or structure de-reference, or function call. Other than the fact that this variable isn't accessible externally, it is just a regular old IDL variable (does this remind you of heap variables in the pointer tutorial?). This temporary variable is passed, just like all other variables in IDL, *by reference* into a calling procedure, e.g.:

```
  mypro, array[x]  ---> mypro, some_internal_idl_temp_var1234
```

Since you can't access that temporary variable explicitly, this is effectively the same as pass by value. You can now set some_internal_idl_temp_var1234 to your heart's content, but you'll never be able to recover the special value you put there:

```
pro mypro, arr
  arr[0]=42
end

IDL> a=randomu(sd,100,1000,100)
IDL> mypro, a[0:800,*,*]
IDL> help,a
A               FLOAT     = Array[1000, 1000, 100]
IDL> print,a[0]
     0.776156  ; wherefore art though, 42?
```

The one difference which makes this distinction more than pedantic is that true pass by value is very inefficient for large arrays. In a pass-by-value scheme, all of that data (801x1000x100) would be copied via the stack into the local address space of the routine MYPRO. It may sound like a subtle difference, but it does represent a real gain in efficiency, in particular when the temporary variable has a life outside the called routine. Eventually, all temporary variables are harvested, and their memory freed. So while you can't ever get at them yourself, they do offer advantages.

JD

Subject: Re: simple question (I hope)
Posted by JD Smith on Fri, 30 Mar 2007 18:30:01 GMT
View Forum Message <> Reply to Message

On Fri, 30 Mar 2007 12:20:28 -0700, David Fanning wrote:

> JD Smith writes:
>
>> The one difference which makes this distinction more than pedantic is
>> that true pass by value is very inefficient for large arrays. In a
>> pass-by-value scheme, all of that data (801x1000x100) would be copied
>> via the stack into the local address space of the routine MYPRO. It may
>> sound like a subtle difference, but it does represent a real gain in
>> efficiency, in particular when the temporary variable has a life outside
>> the called routine. Eventually, all temporary variables are harvested,
>> and their memory freed. So while you can't ever get at them yourself,
>> they do offer advantages.
>
> This is the kind of information I usually try to avoid,
> since it makes it VERY hard to teach IDL classes when
> you know it. I agree it is an important point, and I'll
> store it some place in the back of my head (or in an obscure
> corner of my web page), but I really think my explanation
> is a GREAT DEAL more useful in practice! :-)

You're probably right, but if you can make a mental model of IDL's operations in terms of temporary variables, many other issues relating to optimization of IDL memory usage, which have nothing to do with by-value or by-reference calling, become much clearer. You might also gain insight into those mysterious "temporary variables need cleaning up" messages which pop up from time to time ;).

JD

Subject: Re: simple question (I hope)
Posted by Foldy Lajos on Fri, 30 Mar 2007 18:39:10 GMT
View Forum Message <> Reply to Message

On Fri, 30 Mar 2007, JD Smith wrote:

> This isn't quite correct. Everything, and I mean everything, in IDL is
> passed by reference. However, when IDL encounters a statement like
> `array[x]', or `struct.y', or total(array), it first creates a temporary
> variable to hold the results of the array indexing or structure
> de-reference, or function call. Other than the fact that this variable
> isn't accessible externally, it is just a regular old IDL variable (does
> this remind you of heap variables in the pointer tutorial?). This

> temporary variable is passed, just like all other variables in IDL, *by
> reference* into a calling procedure, e.g.:
>
> mypro, array[x] ---> mypro, some_internal_idl_temp_var1234
>
> Since you can't access that temporary variable explicitly, this is
> effectively the same as pass by value. You can now set
> some_internal_idl_temp_var1234 to your heart's content, but you'll never
> be able to recover the special value you put there:
>
> pro mypro, arr
> arr[0]=42
> end
>
> IDL> a=randomu(sd,100,1000,100)
> IDL> mypro, a[0:800,*,*]
> IDL> help,a
> A   FLOAT   = Array[1000, 1000, 100]
> IDL> print,a[0]
>   0.776156 ; wherefore art though, 42?
>
> The one difference which makes this distinction more than pedantic is
> that true pass by value is very inefficient for large arrays. In a
> pass-by-value scheme, all of that data (801x1000x100) would be copied
> via the stack into the local address space of the routine MYPRO. It may
> sound like a subtle difference, but it does represent a real gain in
> efficiency, in particular when the temporary variable has a life outside
> the called routine. Eventually, all temporary variables are harvested,
> and their memory freed. So while you can't ever get at them yourself,
> they do offer advantages.
>
> JD
>

I don't know how IDL is implemented, but I use pass-by-value for temporary variables in FL. Here "pass" means move, not copy ("move semantics"). The original temporary does not exist after entering the called routine, it is undefined. The called routine gets values, not references. It is faster than pass-by-reference, since no de-referencing is needed for these variables.

regards,
lajos

Subject: Re: simple question (I hope)
Posted by JD Smith on Fri, 30 Mar 2007 18:53:47 GMT

On Fri, 30 Mar 2007 20:39:10 +0200, Fï¿½LDY Lajos wrote:

>
> On Fri, 30 Mar 2007, JD Smith wrote:
>
>> This isn't quite correct. Everything, and I mean everything, in IDL is
>> passed by reference. However, when IDL encounters a statement like
>> `array[x]', or `struct.y', or total(array), it first creates a temporary
>> variable to hold the results of the array indexing or structure
>> de-reference, or function call. Other than the fact that this variable
>> isn't accessible externally, it is just a regular old IDL variable (does
>> this remind you of heap variables in the pointer tutorial?). This
>> temporary variable is passed, just like all other variables in IDL, *by
>> reference* into a calling procedure, e.g.:
>>
>> mypro, array[x] ---> mypro, some_internal_idl_temp_var1234
>>
>> Since you can't access that temporary variable explicitly, this is
>> effectively the same as pass by value. You can now set
>> some_internal_idl_temp_var1234 to your heart's content, but you'll never
>> be able to recover the special value you put there:
>>
>> pro mypro, arr
>> arr[0]=42
>> end
>>
>> IDL> a=randomu(sd,100,1000,100)
>> IDL> mypro, a[0:800,*,*]
>> IDL> help,a
>> A FLOAT = Array[1000, 1000, 100]
>> IDL> print,a[0]
>> 0.776156 ; wherefore art though, 42?
>>
>> The one difference which makes this distinction more than pedantic is
>> that true pass by value is very inefficient for large arrays. In a
>> pass-by-value scheme, all of that data (801x1000x100) would be copied
>> via the stack into the local address space of the routine MYPRO. It may
>> sound like a subtle difference, but it does represent a real gain in
>> efficiency, in particular when the temporary variable has a life outside
>> the called routine. Eventually, all temporary variables are harvested,
>> and their memory freed. So while you can't ever get at them yourself,
>> they do offer advantages.
>>
>> JD
>>
>
> I don't know how IDL is implemented, but I use pass-by-value for temporary

> variables in FL. Here "pass" means move, not copy ("move semantics"). The
> original temporary does not exist after entering the called routine, it is
> undefined. The called routine gets values, not references. It is faster
> than pass-by-reference, since no de-referencing is needed for these
> variables.

I just speculate, but given that most IDL variable types use a pointer
to access their contained data (strings, arrays, etc), from an IDL user
point of view, this is pass by reference, no matter how you inject the
thin wrapper around the variable into a routine. That's what I mean by
by-value vs. by-reference, and I'd guess FL does it the same (?).

JD

Subject: Re: simple question (I hope)
Posted by David Fanning on Fri, 30 Mar 2007 19:20:28 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> The one difference which makes this distinction more than pedantic is
> that true pass by value is very inefficient for large arrays. In a
> pass-by-value scheme, all of that data (801x1000x100) would be copied
> via the stack into the local address space of the routine MYPRO. It may
> sound like a subtle difference, but it does represent a real gain in
> efficiency, in particular when the temporary variable has a life outside
> the called routine. Eventually, all temporary variables are harvested,
> and their memory freed. So while you can't ever get at them yourself,
> they do offer advantages.

This is the kind of information I usually try to avoid,
since it makes it VERY hard to teach IDL classes when
you know it. I agree it is an important point, and I'll
store it some place in the back of my head (or in an obscure
corner of my web page), but I really think my explanation
is a GREAT DEAL more useful in practice! :-)

Cheers,

David

P.S. You should see the eyes glaze over when I start in
on CONTOUR plots. I wish I never knew there was such a
think as a "hole" in a filled contour plot! Or that
NLEVELS=15 gives you no such thing. :-(

--

David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: simple question (I hope)
Posted by Foldy Lajos on Fri, 30 Mar 2007 19:46:27 GMT
View Forum Message <> Reply to Message

On Fri, 30 Mar 2007, JD Smith wrote:

>> I don't know how IDL is implemented, but I use pass-by-value for temporary
>> variables in FL. Here "pass" means move, not copy ("move semantics"). The
>> original temporary does not exist after entering the called routine, it is
>> undefined. The called routine gets values, not references. It is faster
>> than pass-by-reference, since no de-referencing is needed for these
>> variables.
>
> I just speculate, but given that most IDL variable types use a pointer
> to access their contained data (strings, arrays, etc), from an IDL user
> point of view, this is pass by reference, no matter how you inject the
> thin wrapper around the variable into a routine. That's what I mean by
> by-value vs. by-reference, and I'd guess FL does it the same (?).
>

For scalar numeric values, it is exact pass-by-value. For more complicated
data (strings, arrays) it is pass-by-reference for the internal pointer,
if you like. But there is one big difference: the number of references.
For pass-by-reference, there are more than one valid reference. For
pass-by-value (move), there is always one valid reference. This is very
useful for garbage collecting.

IDL users can not access all the references, but IDL internally can, and
managing data with multiple references is difficult, that's why sometimes
temporary variables are not handled correctly.

regards,
lajos

ps: the "small string optimization" is on my TODO list. After that, small
strings will be passed by value, too.

Subject: Re: simple question (I hope)
Posted by Ryan. on Fri, 30 Mar 2007 19:56:19 GMT
View Forum Message <> Reply to Message

Thanks everyone for responding.

It seems it turned out to be not as simple as I thought but I have settled on a method. Without testing it yet, I have settled on the following code:

```
values_2_remove = (huge_array[groupidx])
[indices_2_remove_in_group_array]
nvalues = N_ELEMENTS(values_2_remove)
full_idx = INTARR(nvalues)

FOR k=0, nvalues-1 DO full_idx[k] = WHERE(huge_array EQ
values_2_remove[k])

REMOVE, full_idx, huge_array
```

In general, the items to remove is quite small (~30) so I am content with using the for-loop for cleanliness. And for my purposes the WHERE function will always return only 1 value. If anyone has any further suggestions on how to do this better, feel free to post, I'd love to know =)

I didn't know that through all of this I would be able to learn so much about the magic of IDL =)

Thanks Again,
Ryan.

Subject: Re: simple question (I hope)
Posted by David Fanning on Fri, 30 Mar 2007 21:06:30 GMT
View Forum Message <> Reply to Message

Ryan. writes:

> I didn't know that through all of this I would be able to learn so
> much about the magic of IDL =)

Just about the only time we learn anything new around here is when I newbie jumps in and asks an "easy" question.

Cheers,

David

P.S. It's almost gotten to the point where if I open up the newsgroup and find the works "newbie" and "easy" in the same article, I just turn off the computer and go back

to bed. I'm getting too old for it. :-(

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: simple question (I hope)
Posted by William Daffer on Sun, 01 Apr 2007 17:23:17 GMT
View Forum Message <> Reply to Message

"Ryan." <rchughes@brutus.uwaterloo.ca> writes:

> Dear All,
>
> Do any of you know a fast way of removing elements from an array given
> an array of the indices? I know it is possible with a FOR loop but I
> would like to avoid that if possbile because the array to be searched
> could get quite large. Here is an example of what I would like to do:
>
> A = [0,2,4,6,8,10,12,14,16,18,20]
> indices_to_remove = [3,5,9]
>
> to get a resulting array, B:
> B = [0,2,4,8,12,14,16,20]

```
IDL> A = [0,2,4,6,8,10,12,14,16,18,20]
IDL> indices_to_remove = [3,5,9]
IDL> Good = replicate(1,n_elements(a))
IDL> good[indices_to_remove]=0
IDL> good=where(good)
IDL> a=a[good]
IDL> print,a
      0       2       4       8      12      14      16      20
IDL>
```

whd

--
OWE, v. To have (and to hold) a debt. The word formerly signified
not indebtedness, but possession; it meant "own," and in the minds of
debtors there is still a good deal of confusion between assets and
liabilities.
-- Ambrose Bierce: _The Devil's Dictionary_
