Subject: Re: IDL's handling of LOGICAL quantities (WHERE)
Posted by Mark Fardal on Tue, 12 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

James Tappin <sjt@star.sr.bham.ac.uk> writes:

> \begin{rant}
> I've finally decided to have a public moan about one of the weaknesses of=
> IDL's
> handling of logical operations: to boot -- that the WHERE function follo=
> ws
> a C-like interpretation while most other things are Fortran-like.
>
...
> IDL> print, where(a eq 0)
> 0 2
> IDL> print, where(not (a ne 0))
> 0 1 2 3
>
> I guess the proper answer isto have aproper logical or boolean type and
> functions like FINITE and logical operations should return it, and of cou=
> rse
> WHERE should accept it.

Actually, it would be fine if IDL were more C-like in this case. C doesn't have a boolean type either. But it does maintain a consistent definition of "true", which helps a lot. In C, 2.0 and 2 are both true. In IDL, 2.0 is true but 2 is false. Even worse, IDL conflates the boolean with the bitwise operators. AND, OR, XOR, and NOT are boolean operators for floating types and bitwise operators for the integral types. In C you don't have this problem, because there are different symbols for the boolean and bitwise operators. (Did I get that explanation right? It only took me eight years with IDL to figure out how it works...)

This illogical logic has been around for so long that I think we're stuck with it. But if anyone out there is writing an IDL replacement (hey Arno, you can handle this right?), this is my nomination for the second thing to fix.

Mark Fardal
UMass

Subject: Re: IDL's handling of LOGICAL quantities (WHERE)
Posted by Liam Gumley on Tue, 12 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Liam Gumley wrote:
> To filter out non-finite values in an array, I'd use a function:

Now that I think about it, you might want to check an array subset. The
version below will handle an array subset.

```
FUNCTION CHECK_FINITE, DATA, VALUE=VALUE

;- Check arguments
if n_params() ne 1 then message, 'Usage: RESULT = CHECK_FINITE(DATA)'
if n_elements(data) eq 0 then message, 'DATA is undefined'
if n_elements(value) eq 0 then value = 0.0

;- Set any non-finite elements to VALUE
index = where(finite(data) eq 0, count)
result = data
if count gt 0 then result[index] = value

;- Return the result
return, result

END
```

For example:

```
IDL> a = findgen(5)
IDL> a[0:2] = 1.0/0.0
% Program caused arithmetic error: Floating divide by 0
IDL> print, a
         Inf         Inf         Inf     3.00000     4.00000
IDL> a[0:2] = check_finite(a[0:2])
% Compiled module: CHECK_FINITE.
IDL> print, a
     0.00000     0.00000     0.00000     3.00000     4.00000
```

Cheers,
Liam.

--
Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: IDL's handling of LOGICAL quantities (WHERE)
Posted by Liam Gumley on Tue, 12 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

James Tappin wrote:
>
> \begin{rant}
> I've finally decided to have a public moan about one of the weaknesses of IDL's
> handling of logical operations: to boot -- that the WHERE function follows
> a C-like interpretation while most other things are Fortran-like.
>
> for example suppose we have an array (m) some of whose values are NaN then the
> (inefficient) loop:
> for j=0, n_elements(m) do if not finite(m(j)) then m(j)=0
> will set all non-finite elements of m to 0.
> However:
> m(where(not finite(m))) = 0
> will zero out the whole array since where sees (not 1) as a Yes.
> [The correct solution is of course:
> m(where(finite(m) ne 1)) = 0
>]
>
> Or a simpler example:
> IDL> a = [0, 1, 0, 1]
> IDL> print, where(a eq 0)
> 0 2
> IDL> print, where(not (a ne 0))
> 0 1 2 3
>
> I guess the proper answer isto have aproper logical or boolean type and
> functions like FINITE and logical operations should return it, and of course
> WHERE should accept it.

I think it's useful to look at the output of NOT to understand what it's
doing. For example,

```
IDL> print, not 0
     -1
IDL> print, not 1
     -2
```

This shows that NOT is a bitwise operator for integer operands, which
sets each bit in the operand to it's complement. Funnily enough, if you
use a float operand, the results are what you'd expect of a logical
(rather than bitwise) operator, e.g.

```
IDL> print, not 0.0
     1.00000
IDL> print, not 1.0
     0.00000
```

To filter out non-finite values in an array, I'd use a function:

```
FUNCTION CHECK_FINITE, DATA, VALUE=VALUE

;- Check arguments
if n_params() ne 1 then message, 'Usage: RESULT = CHECK_FINITE(DATA)'
if n_elements(data) eq 0 then message, 'DATA is undefined'
if n_elements(value) eq 0 then value = 0.0

;- Set any non-finite elements to VALUE
index = where(finite(data) eq 0, count)
if count gt 0 then data[index] = value

;- Return the result
return, data

END
```

Example:

```
IDL> a = findgen(5)
IDL> a[0:1] = 1.0/0.0
% Program caused arithmetic error: Floating divide by 0
IDL> print, a
         Inf         Inf      2.00000      3.00000      4.00000
IDL> a = check_finite(a)
IDL> print, a
     0.00000      0.00000      2.00000      3.00000      4.00000
```

Cheers,
Liam.

--
Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

Subject: Re: IDL's handling of LOGICAL quantities (WHERE)
Posted by Pavel Romashkin on Tue, 12 Oct 1999 07:00:00 GMT

View Forum Message <> Reply to Message

Maybe NOT function in IDL is not exactly what some people are used to, but at least it is documented. NOT function performs a biwise conversion, and an expected result will only come if the tested value is truly boolean. Here is what Help says:
"NOT
The NOT operator is the Boolean inverse and is a unary operator (it has only one operand). In other words, "NOT true" is equal to "false" and "NOT false" is equal to "true." NOT complements each bit for integer operands.
Note Signed integers are expressed using the "2s complement" representation. This

means that to arrive at the decimal representation of a negative binary number (a string of binary digits with a one as the most significant bit), you must take the complement of each bit, add one, convert to decimal, and prepend a negative sign. This means that NOT 0 equals -1, NOT 1 equals -2, etc.
For floating-point operands, the result is 1.0 if the operand is zero; otherwise, the result is zero. The NOT operator is not valid for string or complex operands.
NOT 5 = -6

NOT 0101 = 1010"

It is easy tget used to it, especially because other methods of comparison are readily available. It is common in IDL that logical functions return integer values - we better get used to it.
Good luck,
Pavel

James Tappin wrote:

> \begin{rant}
> I've finally decided to have a public moan about one of the weaknesses of IDL's
> handling of logical operations: to boot -- that the WHERE function follows
> a C-like interpretation while most other things are Fortran-like.
>
> for example suppose we have an array (m) some of whose values are NaN then the
> (inefficient) loop:
> for j=0, n_elements(m) do if not finite(m(j)) then m(j)=0
> will set all non-finite elements of m to 0.
> However:
> m(where(not finite(m))) = 0
> will zero out the whole array since where sees (not 1) as a Yes.
> [The correct solution is of course:
> m(where(finite(m) ne 1)) = 0
>]
>
> Or a simpler example:
> IDL> a = [0, 1, 0, 1]
> IDL> print, where(a eq 0)
> 0 2
> IDL> print, where(not (a ne 0))
> 0 1 2 3
>
> I guess the proper answer isto have aproper logical or boolean type and
> functions like FINITE and logical operations should return it, and of course
> WHERE should accept it.

Subject: Re: IDL's handling of LOGICAL quantities (WHERE)

James Tappin (sjt@star.sr.bham.ac.uk) writes:

> \begin{rant}
> \end{rant}

Uh, right. Whatever.

Cheers,

David

P.S. I *think* it was IDL code. James must be a real
software developer. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155