Subject: Re: Using bitwise OR with non-integers; and using !NULL in expressions
Posted by Mark Piper on Thu, 29 Nov 2012 17:17:44 GMT
View Forum Message <> Reply to Message

On Wednesday, November 28, 2012 3:13:09 AM UTC-7, tom.gr...@gmail.com wrote:
> Hello all,
>
>
>
> I found to my delight that bitwise OR on strings evaluates left-to-right, with
>
> empty strings counting as FALSE, which lets me (partially) adapt a useful
>
> idiom for default values from some other language:
>
>
>
>
> my_value = table_value OR default_value
>
>
>
> which to my mind is much more readable than (and avoids the repetition of) the
>
> equivalent:
>
>
>
>
> my_value = table_value ? table_value : default_value
>
>
>
> or (horror of horrors):
>
>
>
>
> IF table_value NE '' THEN BEGIN
>
> my_value = table_value
>
> ENDIF ELSE BEGIN
>
> my_value = default_value
>
> ENDELSE
>
>
>
> It is perhaps strange that it should work, but I was happy to discover this.

> Trying to see how far this useful idiom could be stretched, I tried a couple
> of other variants:
>
>
>
> IDL> print, 0.0 or 42.0
>
> 42.0000
>
> IDL> print, 1.2 or 42.0
>
> 1.20000
>
>
>
> So the idiom carries over to floating-point numbers also. Excellent.
>
>
>
> IDL> print, 1 or 42.0
>
> 1.00000
>
>
>
> Huh? Why did my integer become a float?
>
>
>
> IDL> print, complex(0,0) or 42
>
> (42.0000, 0.00000)
>
>
>
> Okay, so IDL converts both sides to the narrowest type wide enough to
> accomodate both sides. If you're going to do bitwise OR, this makes sense.
> If you're going to return one side or the other, it is less useful, IMO.
>
>
>
> IDL> print, 0 or 'foo'
>
> % Type conversion error: Unable to convert given STRING to Long.

>
>   % Detected at: $MAIN$
>
>   0
>
>
>
> Well, for an operator advertised to do bitwise OR on integers, trying to
>
> convert string to integer is perhaps not unreasonable.
>
>
>
>   IDL> print, !NULL or 'foo'
>
>   % Variable is undefined: <UNDEFINED>.
>
>   % Execution halted at: $MAIN$
>
>
>
> But in this case converting the type is a bad idea: !NULL is false, just like
>
> the empty string is, and I would have hoped that this should evaluate to 'foo'
>
> without error. The same goes for this case:
>
>
>
>   IDL> print, 'foo' or !NULL
>
>   % Variable is undefined: <UNDEFINED>.
>
>   % Execution halted at: $MAIN$
>
>
>
> What I'd like to see (as a minimal change from today's behaviour) is logic
>
> something like this:
>
>
>
>   IF <both sides are integer types> THEN BEGIN
>
>   RETURN, <bitwise or in widest type necessary>
>
>   ENDIF ELSE BEGIN

>
>   RETURN, LHS ? LHS : RHS
>
>   ENDELSE
>
>
>
> In this case, type conversion occurs only to match up the width of integer
>
> arguments. In all other cases, type conversion is unnecessary and only the
>
> truth or falsity of the arguments counts. (This would break >> 0 OR '12' <<
>
> relative to today's behaviour -- but that is fixable.)
>
>
>
> On the other hand, the whole behaviour of OR for non-integers has little to do
>
> with BITWISE operations, and it makes integers behave very differently from
>
> other types for this operator, which violates the principle of least surprise.
>
> The consistent and logical behaviour would be to attempt conversion of ANY LHS
>
> or RHS to an integral type and do bitwise operations on the results. This
>
> would certainly break the idiom for strings and floating-point types.
>
>
>
> If this were to happen, I would _really_ like to have the logical || work for
>
> this idiom, instead. I.e. it would evaluate to its left hand side if it were
>
> (logially) true, and its right hand side otherwise. Today, it returns 0 or 1
>
> which is useful only for tests, not for values. This could break existing
>
> code if the code relied on || raising exceptions when presented with values
>
> without obvious truth or falsity (e.g. arrays)
>
>
>
> That way, the idiom on the first line could work without regard for the type
>
> of the RHS:

>
>
>
>   my_value = compute(arg) || 42
>
>   my_value = compute(arg) || 42.0
>
>
>
> would both behave the same, i.e. the returned value is used if it is true, and
>
> 42 is used otherwise. The operator should short-circuit, i.e. the RHS is
>
> evaluated only if the LHS is false:
>
>
>
>   my_value = cheap_approximation(x) || expensive_approximation(x)
>
>
>
> where expensive_approximation() is called only if cheap_approximation()
>
> produced a FALSE value: 0, 0.0, '', or !NULL.
>
>
>
> Finally, this mind-blowing exercise for the reader (see if you can guess the
>
> result before asking IDL):
>
>
>
>   print, indgen(5) or complex(3,1)
>
>
>
>
>
>
> --T

Hi Tom,

This is a neat idea (and I say this as an avid user of the ternary operator for setting default values); I'll talk it over with Chris.

mp
