Subject: strange GT and LT behavior
Posted by markb77 on Mon, 20 Oct 2014 15:41:50 GMT
View Forum Message <> Reply to Message

I've encountered a bizarre situation where IDL thinks that 0 is less than a negative number. Can anyone rationalize this? Is it really not ok to compare the value of an unsigned integer with a signed integer? Shouldn't the compiler handle this?

test case:

```
pro test_gt_lt

   a = ulong64(0)

   b = long(100)

   if (a lt b) then begin

      print, 'ZERO IS LESS THAN 100'

   endif else begin

      print, 'ZERO IS GREATER THAN 100'

   endelse


   c = ulong64(0)

   d = long(-100)

   if (c lt d) then begin

      print, 'ZERO IS LESS THAN -100'

   endif else begin

      print, 'ZERO IS GREATER THAN -100'

   endelse

end
```

OUTPUT:
ZERO IS LESS THAN 100
ZERO IS LESS THAN -100

IDL> print, !version

{ x86_64 Win32 Windows Microsoft Windows 8.3 Nov 15 2013 64 64}

thanks
Mark

Subject: Re: strange GT and LT behavior
Posted by Dick Jackson on Mon, 20 Oct 2014 16:35:46 GMT
View Forum Message <> Reply to Message

On Monday, 20 October 2014 08:41:52 UTC-7, superchromix wrote:
> I've encountered a bizarre situation where IDL thinks that 0 is less than a negative number. Can anyone rationalize this? Is it really not ok to compare the value of an unsigned integer with a signed integer? Shouldn't the compiler handle this?

Hi Mark,

Forgive me for boiling down your test case:

```
IDL> 0ULL LT -100L
  1
```

I think what's happening is that, to compare a 64-bit (unsigned) type to a 32-bit (signed) type, the 32-bit value is converted to the "higher precedence" type, even though it will no longer be able to represent a negative number

From Help on "Language > Operators > Relational Operators"
=====
Each operand is promoted to the data type of the operand with the greatest precedence or potential precision. (See Data Type and Structure of Expressions for details.)
=====

Here's what was happening

```
IDL> 0ULL LT ULong64(-100L)
  1
IDL> help, -100L
<Expression>    LONG      =       -100
IDL> help, ULong64(-100L)
<Expression>    ULONG64   =   18446744073709551516
```

If we're pushing the limits here, this is possibly even more troublesome:

=====
Note: Signed and unsigned integers of a given width have the same precedence. In an expression involving a combination of such types, the result is given the type of the leftmost operand.
=====

This leads to the following curiosity, where it seems that, with the same "level" of precision (64

bits, but one signed and one unsigned), a < b and b < a:

```
IDL> 0ULL LT -100LL
   1
IDL> -100LL LT 0ULL
   1
```

I suppose the lesson here is, if there's a chance of comparing positive and negative values, be sure to convert both expressions to a signed type, or a float type.

Cheers,
-Dick

Dick Jackson Software Consulting Inc.
Victoria, BC, Canada - www.d-jackson.com

Subject: Re: strange GT and LT behavior
Posted by markb77 on Mon, 20 Oct 2014 16:48:28 GMT
View Forum Message <> Reply to Message

On Monday, October 20, 2014 6:35:48 PM UTC+2, Dick Jackson wrote:
> On Monday, 20 October 2014 08:41:52 UTC-7, superchromix wrote:
>
>> I've encountered a bizarre situation where IDL thinks that 0 is less than a negative number. Can anyone rationalize this? Is it really not ok to compare the value of an unsigned integer with a signed integer? Shouldn't the compiler handle this?
>
>
>
> Hi Mark,
>
>
>
> Forgive me for boiling down your test case:
>
>
>
> IDL> 0ULL LT -100L
>
> 1
>
>
>
> I think what's happening is that, to compare a 64-bit (unsigned) type to a 32-bit (signed) type, the 32-bit value is converted to the "higher precedence" type, even though it will no longer be able to represent a negative number
>

>
>
> From Help on "Language > Operators > Relational Operators"
>
> =====
>
> Each operand is promoted to the data type of the operand with the greatest precedence or potential precision. (See Data Type and Structure of Expressions for details.)
>
> =====
>
>
>
> Here's what was happening
>
> IDL> 0ULL LT ULong64(-100L)
>
> 1
>
> IDL> help, -100L
>
> <Expression> LONG = -100
>
> IDL> help, ULong64(-100L)
>
> <Expression> ULONG64 = 18446744073709551516
>
>
>
> If we're pushing the limits here, this is possibly even more troublesome:
>
>
>
> =====
>
> Note: Signed and unsigned integers of a given width have the same precedence. In an expression involving a combination of such types, the result is given the type of the leftmost operand.
>
> =====
>
>
>
> This leads to the following curiosity, where it seems that, with the same "level" of precision (64 bits, but one signed and one unsigned), a < b and b < a:
>
>
>

> IDL> 0ULL LT -100LL
>
> 1
>
> IDL> -100LL LT 0ULL
>
> 1
>
>
>
> I suppose the lesson here is, if there's a chance of comparing positive and negative values, be sure to convert both expressions to a signed type, or a float type.
>
>
>
> Cheers,
>
> -Dick
>
>
>
> Dick Jackson Software Consulting Inc.
>
> Victoria, BC, Canada - www.d-jackson.com

hi Dick,

Thanks for the insights. The reasons for this behavior is clear.. it was just somewhat unexpected.

Mark

Subject: Re: strange GT and LT behavior
Posted by Craig Markwardt on Mon, 20 Oct 2014 21:28:15 GMT
View Forum Message <> Reply to Message

On Monday, October 20, 2014 12:48:30 PM UTC-4, superchromix wrote:
>
> Thanks for the insights. The reasons for this behavior is clear.. it was just somewhat unexpected.

When I was young I thought integer math on a computer was so simple and easy. After doing some moderately intensive integer calculations in C, I realized that integer math is the work of evil.

The interactions of signed vs unsigned, and short vs long data types, is very subtle and prone to error. One needs to pay very careful attention to compiler/interpreter conventions regarding

integer math. In my particular case I was using integer math in C to avoid the overhead of floating point, so it was "worth it."

Craig

Subject: Re: strange GT and LT behavior
Posted by markb77 on Tue, 21 Oct 2014 11:01:04 GMT
View Forum Message <> Reply to Message

On Monday, October 20, 2014 11:28:17 PM UTC+2, Craig Markwardt wrote:
> On Monday, October 20, 2014 12:48:30 PM UTC-4, superchromix wrote:
>
>>
>
>> Thanks for the insights. The reasons for this behavior is clear.. it was just somewhat unexpected.
>
>
>
> When I was young I thought integer math on a computer was so simple and easy. After doing some moderately intensive integer calculations in C, I realized that integer math is the work of evil.

>
>
>
> The interactions of signed vs unsigned, and short vs long data types, is very subtle and prone to error. One needs to pay very careful attention to compiler/interpreter conventions regarding integer math. In my particular case I was using integer math in C to avoid the overhead of floating point, so it was "worth it."
>
>
>
> Craig

If this was C, the programmer would see a compiler warning to flag the comparison of a signed with an unsigned variable. I wonder if such a "warning" would be possible in IDL...? Now that I think of it, probably not, since a function doesn't know what will be passed to it until runtime..
