Subject: Re: Commutativity of multiplication
Posted by K. Bowman on Wed, 25 Oct 2006 20:42:36 GMT
View Forum Message <> Reply to Message

In article <ehoh6g$asc$1@naig.caltech.edu>, Sven Geier <sZgeier@calteZch.edZu>
wrote:

```
> There's something odd afoot with ULong variables (in 6.3) that isn't clear
> to me:
>
> IDL> help,f,g,h
> F              LONG      =         500
> G              ULONG     =           1
> H              FLOAT     =      1.00000
> IDL> help,f*h,h*f
> <Expression>   FLOAT     =      500.000
> <Expression>   FLOAT     =      500.000
> IDL> help,f*g,g*f
> <Expression>   LONG      =         500
> <Expression>   ULONG     =         500
>
>
> Is this a bug? A feature? Well-known gotcha? Obscure property? Recently
> added functionality? Vestigial peculiarity?
>
>
> - S
```

A little poking around in the docs revealed this:

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. (Emphasis in original.)

Ken Bowman

Subject: Re: Commutativity of multiplication
Posted by David Fanning on Wed, 25 Oct 2006 20:54:20 GMT
View Forum Message <> Reply to Message

Sven Geier writes:

```
> There's something odd afoot with ULong variables (in 6.3) that isn't clear
> to me:
>
> IDL> help,f,g,h
> F              LONG      =         500
>  G              ULONG     =           1
>  H              FLOAT     =      1.00000
>  IDL> help,f*h,h*f
>  <Expression>    FLOAT     =      500.000
>  <Expression>    FLOAT     =      500.000
>  IDL> help,f*g,g*f
>  <Expression>    LONG      =          500
>  <Expression>    ULONG     =          500
>
>
>  Is this a bug? A feature? Well-known gotcha? Obscure property? Recently
>  added functionality? Vestigial peculiarity?
```

I'm pretty sure it's not a bug. I don't know about all
the rest of the possibilities. -)

I think the operative rules here are:

1. Maintain the type of data that maintains the most accuracy
2. If things are the same, process from left to right.

A ULONG and a LONG are both four bytes, and "accuracy" is
nebulous. A long could be negative. Is that "more accurate"
than treating the highest byte as a number rather than a
sign? I think you could argue both ways. So let's call it
even. Then, falling back on rule 2 explains the results.

It also explains this:

```
IDL> f = 500L
IDL> g = 1UL
IDL> help, g*(-f)
<Expression>    ULONG     =   4294966796
IDL> help, (-f)*g
<Expression>    LONG      =         -500
IDL> print, long(g*(-f))
        -500
```

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: Commutativity of multiplication
Posted by Sven Geier on Wed, 25 Oct 2006 21:54:59 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
[...]
```
>  It also explains this:
>
>  IDL> f = 500L
>  IDL> g = 1UL
>  IDL> help, g*(-f)
>  <Expression>    ULONG     =   4294966796
>  IDL> help, (-f)*g
```

Wow. This is ... uhm ... "more interesting than I thought". There's whole
realms of oddity here that I never knew existed:

```
IDL> help,f,g
F               LONG      =        -500
G               ULONG     =           1
IDL> print,f*g
        -500
IDL> print,g*f
  4294966796
IDL> if f*g eq g*f then print,f*g," equals ",g*f
        -500 equals   4294966796
```

So for the sake of multiplication preservation of a sign is considered
important enough to break commutativity, but for the sake of comparison,
type conversion is performed that changes the value of an expression by
four billion.

Some days I can get all my exercise just by shaking my head over these kinds
of oddities...

Thanks for the reply, tho ...

- S

--
http://www.sgeier.net
My real email address does not contain any "Z"s.

Subject: Re: Commutativity of multiplication
Posted by David Fanning on Wed, 25 Oct 2006 22:33:29 GMT
View Forum Message <> Reply to Message

Sven Geier writes:

> Some days I can get all my exercise just by shaking my head over these kinds
> of oddities...

When you get tired of this, you can work on why JAVA
thinks a byte goes from plus or minus 127, instead of
from 0 to 255. :-)

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: Commutativity of multiplication
Posted by JD Smith on Wed, 25 Oct 2006 23:09:03 GMT
View Forum Message <> Reply to Message

On Wed, 25 Oct 2006 14:54:59 -0700, Sven Geier wrote:

```
> David Fanning wrote:
> [...]
>> It also explains this:
>>
>> IDL> f = 500L
>> IDL> g = 1UL
>> IDL> help, g*(-f)
>> <Expression>    ULONG     =   4294966796
>> IDL> help, (-f)*g
>
> Wow. This is ... uhm ... "more interesting than I thought". There's whole
> realms of oddity here that I never knew existed:
>
> IDL> help,f,g
> F               LONG      =        -500
> G               ULONG     =           1
> IDL> print,f*g
>         -500
> IDL> print,g*f
>   4294966796
> IDL> if f*g eq g*f then print,f*g," equals ",g*f
>         -500 equals   4294966796
```

The bit pattern for these two numbers is *exactly* the same. The only difference is the "type" that IDL assigns them, which affects how the numbers are printed (and only how they are printed). That type (for IDL) is determined by the "left-most" variable or constant in the expression, with the exception that if anything to the right has a larger range, it will be used as the type (e.g. promotion to float, double, ULONG64, etc.), hence:

```
IDL> f=1.0
IDL> l=100UL
IDL> help,l+f
<Expression>    FLOAT     =       101.000
```

When the range is the same, e.g. UL and L, leftmost always wins.

Commutation hasn't been broken, only "type commutation", which doesn't really exist. For all purposes, given the limitations of integer representation in computers, -500 and 4294966796 *are* the same. I could just as easily claim that "adding and subtracting 1 is broken":

```
IDL> print, 4294967295UL + 1UL
          0

IDL> print,0b - 1b
 255
```

JD

Subject: Re: Commutativity of multiplication
Posted by Foldy Lajos on Thu, 26 Oct 2006 08:26:12 GMT
View Forum Message <> Reply to Message

On Wed, 25 Oct 2006, JD Smith wrote:

```
>  Commutation hasn't been broken, only "type commutation", which doesn't
>  really exist.  For all purposes, given the limitations of integer
>  representation in computers, -500 and 4294966796 *are* the same.  I
>  could just as easily claim that "adding and subtracting 1 is broken":
>
>  IDL> print, 4294967295UL + 1UL
>            0
>
>  IDL> print,0b - 1b
>  255
>
>  JD
```

>

If multiplication is commutative, then a*b should be equal to b*a.

```
IDL> a=-1l
IDL> b= 1ul
IDL> print, a*b eq b*a
1
```

Fine. If a*b is equal to b*a, then 1.0*(a*b) should be equal to 1.0*(b*a),
too.

```
IDL> print, 1.0*(a*b) eq 1.0*(b*a)
0
```

I tend to say that IDL's multiplication is not commutative in
the mathematical sense.

Other languages, like C are "more commutative":

```
    signed int a=-1;
  unsigned int b= 1;

  printf("%d %d\n", a*b==b*a, 1.0*(a*b)==1.0*(b*a));
```

prints 1 1 (the result is signed int both for a*b and b*a).

regards,
lajos

Subject: Re: Commutativity of multiplication
Posted by Braedley on Thu, 26 Oct 2006 17:24:32 GMT

View Forum Message <> Reply to Message

JD Smith wrote:

```
>  On Wed, 25 Oct 2006 14:54:59 -0700, Sven Geier wrote:
>
>>  David Fanning wrote:
>>  [...]
>>>  It also explains this:
>>>
>>>  IDL> f = 500L
>>>  IDL> g = 1UL
>>>  IDL> help, g*(-f)
>>>  <Expression>    ULONG     =   4294966796
>>>  IDL> help, (-f)*g
>>
```

```
>>  Wow. This is ... uhm ... "more interesting than I thought". There's whole
>>  realms of oddity here that I never knew existed:
>>
>>  IDL> help,f,g
>>  F               LONG      =         -500
>>  G               ULONG     =            1
>>  IDL> print,f*g
>>         -500
>>  IDL> print,g*f
>>    4294966796
>>  IDL> if f*g eq g*f then print,f*g," equals ",g*f
>>         -500 equals   4294966796
>
>  The bit pattern for these two numbers is *exactly* the same.  The only
>  difference is the "type" that IDL assigns them, which affects how the
>  numbers are printed (and only how they are printed).  That type (for
>  IDL) is determined by the "left-most" variable or constant in the
>  expression, with the exception that if anything to the right has a
>  larger range, it will be used as the type (e.g. promotion to float,
>  double, ULONG64, etc.), hence:
>
>  IDL> f=1.0
>  IDL> l=100UL
>  IDL> help,l+f
>  <Expression>    FLOAT     =       101.000
>
>  When the range is the same, e.g. UL and L, leftmost always wins.
>
>  Commutation hasn't been broken, only "type commutation", which doesn't
>  really exist.  For all purposes, given the limitations of integer
>  representation in computers, -500 and 4294966796 *are* the same.  I
>  could just as easily claim that "adding and subtracting 1 is broken":
>
>  IDL> print, 4294967295UL + 1UL
>            0
>
>  IDL> print,0b - 1b
>   255
>
>  JD
```

IDL must make a choice as to which type to use, since the length of ranges of LONG and ULONG are exactly the same, but there is only 50% overlap. The result from the multiplication may be within the range of both, in which case everything is fine. However, if the result is negative, the result will be outside the range of ULONG. Likewise, if the two numbers are sufficiently large, the result will be outside the range of LONG. IDL doesn't know beforehand what the result will be, and therefore assigns the type of the leftmost variable.

Subject: Re: Commutativity of multiplication
Posted by JD Smith on Thu, 26 Oct 2006 17:37:34 GMT
View Forum Message <> Reply to Message

On Thu, 26 Oct 2006 10:26:12 +0200, Fï¿½LDY Lajos wrote:

```
>
> On Wed, 25 Oct 2006, JD Smith wrote:
>
>> Commutation hasn't been broken, only "type commutation", which doesn't
>> really exist.  For all purposes, given the limitations of integer
>> representation in computers, -500 and 4294966796 *are* the same.  I
>> could just as easily claim that "adding and subtracting 1 is broken":
>>
>> IDL> print, 4294967295UL + 1UL
>>           0
>>
>> IDL> print,0b - 1b
>> 255
>>
>> JD
>>
>>
> If multiplication is commutative, then a*b should be equal to b*a.
>
> IDL> a=-1l
> IDL> b= 1ul
> IDL> print, a*b eq b*a
> 1
>
> Fine. If a*b is equal to b*a, then 1.0*(a*b) should be equal to 1.0*(b*a),
> too.
>
> IDL> print, 1.0*(a*b) eq 1.0*(b*a)
> 0
>
> I tend to say that IDL's multiplication is not commutative in the
> mathematical sense.
>
> Other languages, like C are "more commutative":
>
>     signed int a=-1;
>    unsigned int b= 1;
>
>    printf("%d %d\n", a*b==b*a, 1.0*(a*b)==1.0*(b*a));
```

>
> prints 1 1 (the result is signed int both for a*b and b*a).

There's nothing "more commutative" about this. It's just a reflection of the interaction of type casting and upconversion to float. The real equivalency in C is:

```
  int a=-1;
  unsigned int b=1;

  printf("%d %d\n", (int)(a*b)==(unsigned int)(b*a),
  1.0*(int)(a*b)==1.0*(unsigned int)(b*a));
```

prints 1 0

You could get your "more commutative" behavior out of IDL too:

```
IDL> print,1.0*long(a*b) eq 1.0*long(b*a)
   1
```

So when converting an integer and its equivalent unsigned int to float, C (and IDL), do care about its type, despite the fact that the bit pattern of the two is precisely the same. This is because a float is encoded differently from an integer (i.e. without twos complement wraparound), so the distinction between -500 and 4294966796 is significant for up-conversion.

The difference between the two is, C forces you to keep track of the desired output integer type (with no "leftmost type wins" convention), and make an explicit cast, whereas IDL implicitly does the cast for you. Which behavior is more convenient depends on usage.

JD

Subject: Re: Commutativity of multiplication
Posted by JD Smith on Thu, 26 Oct 2006 17:47:50 GMT

View Forum Message <> Reply to Message

On Thu, 26 Oct 2006 10:24:32 -0700, Braedley wrote:

```
>>  IDL> print, 4294967295UL + 1UL
>>            0
>>
>>  IDL> print,0b - 1b
>>   255
>>
>>  JD
>
```

> IDL must make a choice as to which type to use, since the length of ranges
> of LONG and ULONG are exactly the same, but there is only 50% overlap.
> The result from the multiplication may be within the range of both, in
> which case everything is fine. However, if the result is negative, the
> result will be outside the range of ULONG. Likewise, if the two numbers
> are sufficiently large, the result will be outside the range of LONG. IDL
> doesn't know beforehand what the result will be, and therefore assigns the
> type of the leftmost variable.

It's actually much simpler than that, with no real "decision" involved. The result of multiplying two integers is exactly the same whether they are interpreted as signed or unsigned, including if you overflow the size of the integer (this is the chief reason the two-complement system is so prevalent). It is up to you (or, in this case, IDL) to decide how to interpret the resulting number.

```
IDL> print,2L^30 * 2L
 -2147483648
IDL> print,2UL^30 * 2UL
  2147483648
IDL> print,long(2UL^30 * 2UL)
 -2147483648
```

The only case where this isn't so is up-conversion to non-equivalent types (like floating point), but that's a separate matter, and it is up to the environment (C compiler, IDL, etc.) as to how to handle that.

JD