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