Subject: Re: rounding errors
Posted by Jaco van Gorkom on Fri, 04 May 2001 13:17:54 GMT
View Forum Message <> Reply to Message

Paul van Delst wrote:
> James Kuyper wrote:
>> Paul van Delst wrote:
>>> Randall Skelton wrote:
>>>> On Fri, 27 Apr 2001, Liam E. Gumley wrote:
>>>>
>>>> > This is a subtle but important point. DOUBLE() is a type conversion
>>>> > function, and
>>>> >
>>>> > a = double(2.348339)
>>>> >
>>>> > shows a FLOAT argument being converted to a DOUBLE. The safest way
to
>>>> > 'cast' a double variable is
>>>> >
>>>> > a = 2.348339d
>>>> [snip]
>>>>
>>>> Wow... I am glad that I have now learned that particular 'IDL
feature'
>>>> early on in my PhD.

>> [snip] the "feature" of IDL that surprised me
>> as much as it surprised Randall and Liam. This has everything to do with
>> IDL, and nothing to do with expecting exact representation of a
>> finite-length decimal fraction. By default, in C 2.348339 represents a
>> double precision number, not a single precision one, and I'd never
>> realized that the IDL convention was different.
>
> Ahh, I see. I guess it depends on what you started with. I code in Fortran
mostly so when
> I think "default floating point number" I think single-precision. In
Fortran at least (and
> by association IDL??), that convention _probably_ grew out of memory
limitations of
> computers and whatnot back in olden day. Nowadays it (mostly) doesn't
matter I guess.
>
> I wonder what other languages use as a default? (e.g. Matlab sticks
everything in double
> doesn't it? Probably strings as well.... :o)

I was thinking, wouldn't the logical IDL behaviour be to take whatever data

type is necessary to hold a constant? I mean, it works that way with integer
constants:

```
IDL> help, 17, 33000, 2150000000
<Expression>    INT       =       17
<Expression>    LONG      =       33000
<Expression>    LONG64    =           2150000000
```

Obviously IDL does not do the same type of overflow checking for
floating-point constants:

```
IDL> help,1e38, 5e38, 5d38
<Expression>    FLOAT     =  1.00000e+038
<Expression>    FLOAT     =           Inf
<Expression>    DOUBLE    =   5.0000000e+038
```

(note that no arithmetic error is thrown in this case, since no arithmetic
is being done, really.)

When specifying more decimal places for a constant, the choice between
integer and float data types is made automagically:

```
IDL> help, 2, 2.0
<Expression>    INT       =       2
<Expression>    FLOAT     =      2.00000
```

Logical behaviour then would seem that constants with more than, say, six
decimal places are interpreted as double-precision floats. 'Logical' in an
IDL sort of way: if you don't explicitly state the data type you want for a
constant, then you'll get something which is at least capable of holding it.

Just my thoughts on what could have been... It would have made life for
especially inexperienced users less tricky (together with smarter data
typing of FOR-loop counters, easier printing, and what have we). One
practical survival strategy in IDL: always explicitly declare the data type
of constants (by appending L, LL, D, ..).

Jaco