Subject: strange problem: Alarm Clock????....
Posted by stl on Tue, 09 Aug 1994 16:05:25 GMT
View Forum Message <> Reply to Message

hi all,

this is very weird. after I use a routine thats in a shared object, called with call_external, about 2-5 minutes later I get the following message on my prompt:

IDL> Alarm Clock

And then get thrown out of IDL. Any ideas? I am running 3.6.1a on a Sparc station 10 running Solaris 2.x.

I have been at this for a while, and cannot replicate the problem. I have reduced the routine that is in the shared object file to just a return(-1). I have some more testing to do, but I am getting really baffeled. Is this a 3.6.1a thing? (doesn't seem like it because none of the test call externals do this)

aaaaaaaaaaaaaaarghhhhh...

thanks a bunch for any help. This is almost funny.

-stephen

--
Stephen C Strebel / SKI TO DIE
strebel@sma.ch / and
Swiss Meteorological Institute, Zuerich / LIVE TO TELL ABOUT IT
01 256 93 85 / (and pray for snow)

Subject: Re: Strange problem
Posted by Bhautik Joshi on Mon, 26 Nov 2001 01:38:05 GMT
View Forum Message <> Reply to Message

> Anybody know what's going on?

My spin on it:

Well, I think it may be a problem that goes right down to the core, and is not just restricted to FOR loops.

Example:

MOO>a=replicate(0.1,100)

```
MOO>print, total(a)
      10.0000
MOO>print, total(a) - 0.1*100
  1.90735e-06
```

argh! where I think the problem may lie is with the binary representation of floating point numbers. Now, its been a looooooooooooooooooooooong while since I did computer architecture (yes, it is a real subject at uni) but as far as I remember, in floating point binary, there is no 'exact' representation for 0.1.

However, lets take 0.5 - which is a quarter of 2; something nice and (pardon the expression) base-twoish.

```
MOO>a=replicate(0.5,100)
MOO>print, total(a) - 0.5*100
      0.00000
```

Lets change the game slightly again:

```
MOO>a=replicate(0.51,100)
MOO>print, total(a) - 0.51*100
 -4.95911e-05
```

should i be worried?

--
```
 /-------------------------------------------------(__)----- ----\
| nbj@imag.wsahs.nsw.gov.au | phone: 0404032617     |..|--\ -moo |
| ICQ #: 2464537            | http://cow.mooh.org  |  |--|      |
\-------------------------------------------------\OO/|| ------/
```

Subject: Re: Strange problem
Posted by Bhautik Joshi on Mon, 26 Nov 2001 03:55:17 GMT
View Forum Message <> Reply to Message

> type: for i=0., 0.8., 0.1 do print,i
> That's bad isn't it?
OK lets drag this out a little further :)

Lets translate this into a WHILE loop and try it again, using INTS.

```
MOO>i=0
MOO>while i le 8 do begin print, i & i=i+1
       0
       1
```

```
       2
       3
       4
       5
       6
       7
       8
```

Now again, using FLOAT:

```
MOO>i=0.0
MOO>while i le 0.8 do begin print, i & i=i+0.1
      0.00000
     0.100000
     0.200000
     0.300000
     0.400000
     0.500000
     0.600000
     0.700000
```

What's going on? Shouldn't it count up to 0.8?

```
MOO>print, i-0.8
  5.96046e-08
```

Basically what has happened is that when 0.1 gets added to 0.7, the result is > 0.8, so the loop gets terminated early!

This is because what you see in the screen is rounded to a couple of decimal places; where it says 0.7 it should really be something like 0.7000000001. As the addition occurs, accuracy is lost.

To demonstrate, try this routine:

```
pro foo, l
 loopvar=FLOAT(0.0)
 lim=FLOAT(l)
 inc=FLOAT(0.1)
 i=FIX(0)

 while (loopvar le l) do begin
  actvar=FLOAT(i)*inc
  print, 'loopvar: ',loopvar,' actvar: ',actvar,'
diff:',(loopvar-actvar)
  loopvar=loopvar+inc
  i=i+1
 end
```

end

So, trying it out:

```
MOO>foo, 0.8
loopvar:       0.00000 actvar:       0.00000 diff:      0.00000
loopvar:      0.100000 actvar:      0.100000 diff:      0.00000
loopvar:      0.200000 actvar:      0.200000 diff:      0.00000
loopvar:      0.300000 actvar:      0.300000 diff:      0.00000
loopvar:      0.400000 actvar:      0.400000 diff:      0.00000
loopvar:      0.500000 actvar:      0.500000 diff:      0.00000
loopvar:      0.600000 actvar:      0.600000 diff:      0.00000
loopvar:      0.700000 actvar:      0.700000 diff:  5.96046e-08
```

So whats printed on the screen is 0.700000, but the actual number is 0.70000006 (roughly - remember that on a computer, only exact multiples or um.. whats the word - numbers that have been divided exactly by - 2 can be exactly represented. ie. ..2.0,1.0,0.5,0.25.. are prescise; however, ..2.1,1.1,0.51,0.251.. have a degree of inaccuracy).

The exact same thing occurs when you write the same thing in C, so its not just something limited to IDL.

The morals of this story:
* don't let your loop variable=data variable
* floats have only 7 digits of prescision (on most platforms); where you are trying to be exact (and loop variables have to be exact) either use a DOUBLE (which has 15 digits of prescision on most systems) or use an INT to count the number of times the loop goes around
* The 29,249th digit of pi is a seven.

Cheers,
Bhautik

--

```
 /-------------------------------------------------(__)----- ----\
| nbj@imag.wsahs.nsw.gov.au | phone: 0404032617     |..|--\ -moo |
| ICQ #: 2464537            | http://cow.mooh.org   |  |--|      |
\-------------------------------------------------\OO/|| ------/
```

Subject: Re: Strange problem
Posted by Martin Downing on Mon, 26 Nov 2001 15:00:17 GMT
View Forum Message <> Reply to Message

Andre,

Joshi is right, this behaviour is due to the lack of precision in

floating point number representation. With your for loop

```
for i=0., 0.801, 0.1 do print,i
```

The code execution can more easily be visualised as

```
i = 0.
while i LE 0.8 do begin
   print, i
   i  = i + 0.1
endwhile
```

Thus:

```
IDL> for i=0., 0.8, 0.1 do print,i
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.700000
```

So what was the final value of i?

```
IDL> print, i
0.800000
```

Oh, isnt that the value of the upper bound?

```
IDL> print, i EQ 0.8
0
IDL> print, i - 0.8
5.96046e-008
```

Clearly not! Slightly more than 0.1 was added each time, so there was a small excess to i when representing 0.8

So the moral is that you have to be very careful when applying comparison operators to floating point numbers, one of which is implicitly applied in the FOR statement. Now you realise the problem, the answer is to be a little less strict with your comparisons. With FOR loops you can add a small excess, relative to the increment, to the upper bound:

```
IDL> for i=0., 0.8001, 0.1 do print,i
0.000000
0.100000
```

0.200000
0.300000
0.400000
0.500000
0.600000
0.700000
0.800000

Out of interest notice that the final value of "i" is now 0.9:
IDL> print, i
0.900000

> should i be worried?
Well if you write code which depends on floating point numbers having perfect precision then yes!
If you wanted to compare two floats for equality, you have to rethink what you mean by "equal", i.e. how exact does this application need the variables to be?
Relying on doubles is not a robust solution, so instead of writing:

```
IF a EQ b THEN ...
```

write

```
myPrecision = 0.001
IF abs(a-b) LT myPrecision THEN ......
```

I hope this helps

Martin

"Bhautik Joshi" <nbj@imag.wsahs.nsw.gov.au> wrote in message news:3C019CFD.B20DB925@imag.wsahs.nsw.gov.au...
>> Anybody know what's going on?
>
> My spin on it:
>
> Well, I think it may be a problem that goes right down to the core, and
> is not just restricted to FOR loops.
>
> Example:
>
> MOO>a=replicate(0.1,100)
> MOO>print, total(a)
> 10.0000
> MOO>print, total(a) - 0.1*100

> 1.90735e-06
>
> argh! where I think the problem may lie is with the binary
> representation of floating point numbers. Now, its been a
> looooooooooooooooooooooong while since I did computer architecture
> (yes, it is a real subject at uni) but as far as I remember, in floating
> point binary, there is no 'exact' representation for 0.1.
>
> However, lets take 0.5 - which is a quarter of 2; something nice and
> (pardon the expression) base-twoish.
>
> MOO>a=replicate(0.5,100)
> MOO>print, total(a) - 0.5*100
>   0.00000
>
> Lets change the game slightly again:
>
> MOO>a=replicate(0.51,100)
> MOO>print, total(a) - 0.51*100
> -4.95911e-05
>
> should i be worried?
>
>
> --
> /---(__)----- ----\
> | nbj@imag.wsahs.nsw.gov.au | phone: 0404032617 |..|--\ -moo |
> | ICQ #: 2464537 | http://cow.mooh.org | |--| |
> \---\OO/|| ------/

Subject: Re: Strange problem
Posted by Pavel A. Romashkin on Mon, 26 Nov 2001 18:15:57 GMT
View Forum Message <> Reply to Message

Please enlighten me why would one want to use a floating point value as
a counter?

Pavel

Andre Kyme wrote:
>
> Hi everyone,
>
> type: for i=0., 0.8., 0.1 do print,i
> That's bad isn't it?i

Subject: Re: Strange problem
Posted by Andre Kyme on Mon, 26 Nov 2001 22:04:49 GMT
View Forum Message <> Reply to Message

"Pavel A. Romashkin" wrote:

> Please enlighten me why would one want to use a floating point value as
> a counter?
>
> Pavel
>
> Andre Kyme wrote:
>>
>> Hi everyone,
>>
>> type: for i=0., 0.8., 0.1 do print,i
>> That's bad isn't it?i

Pavel,
Floating point counters are so roomy you can put you're PC, lunch, coffee,
and notes on them and still have room to move!
I thought that was obvious.

Andre

Subject: Re: Strange problem
Posted by Andre Kyme on Mon, 26 Nov 2001 22:10:31 GMT
View Forum Message <> Reply to Message

Martin Downing wrote:

> Andre,
>
> Joshi is right, this behaviour is due to the lack of precision in
> floating point number representation. With your for loop
>
> for i=0., 0.801, 0.1 do print,i
>
> The code execution can more easily be visualised as
>
> i = 0.
> while i LE 0.8 do begin
> print, i
> i = i + 0.1
> endwhile
>
> Thus:

>
>   IDL> for i=0., 0.8, 0.1 do print,i
>   0.000000
>   0.100000
>   0.200000
>   0.300000
>   0.400000
>   0.500000
>   0.600000
>   0.700000
>
> So what was the final value of i?
>
>   IDL> print, i
>   0.800000
>
> Oh, isnt that the value of the upper bound?
>
>   IDL> print, i EQ 0.8
>   0
>   IDL> print, i - 0.8
>   5.96046e-008
>
> Clearly not! Slightly more than 0.1 was added each time, so there was a
> small excess to i when representing 0.8
>
> So the moral is that you have to be very careful when applying comparison
> operators to floating point numbers, one of which is implicitly applied in
> the FOR statement. Now you realise the problem, the answer is to be a little
> less strict with your comparisons. With FOR loops you can add a small
> excess, relative to the increment, to the upper bound:
>
>   IDL> for i=0., 0.8001, 0.1 do print,i
>   0.000000
>   0.100000
>   0.200000
>   0.300000
>   0.400000
>   0.500000
>   0.600000
>   0.700000
>   0.800000
>
> Out of interest notice that the final value of "i" is now 0.9:
>   IDL> print, i
>   0.900000
>
>> should i be worried?

> Well if you write code which depends on floating point numbers having
> perfect precision then yes!
> If you wanted to compare two floats for equality, you have to rethink what
> you mean by "equal", i.e. how exact does this application need the variables
> to be?
> Relying on doubles is not a robust solution, so instead of writing:
>
> IF a EQ b THEN ...
>
> write
>
> myPrecision = 0.001
> IF abs(a-b) LT myPrecision THEN .....
>
> I hope this helps
>
> Martin

Thanks Martin, that makes good sense. I can see the good reason for always keeping your loop variable as an integer, so I'll make sure I do this from now on.
Andre

Subject: Re: Strange problem
Posted by Jeff Hester on Wed, 05 Dec 2001 15:40:57 GMT
View Forum Message <> Reply to Message

The danger of IDL is that it allows people access to tools about which they have no knowledge.
If the fact that floating point representations of numbers have limited precision comes as a
shock, one can only wonder...

Is there an emoticon for "shudder in abject fear"?

"Pavel A. Romashkin" wrote:

> Please enlighten me why would one want to use a floating point value as
> a counter?
>
> Pavel
>
> Andre Kyme wrote:
>>
>> Hi everyone,
>>

>> type: for i=0., 0.8., 0.1 do print,i
>> That's bad isn't it?i

--
Jeff Hester
Professor
Dept. of Physics & Astronomy
Arizona State University
jhester@asu.edu
