Subject: -32768
Posted by greg.addr on Wed, 19 Mar 2014 04:43:59 GMT
View Forum Message <> Reply to Message

I'd like to set an integer variable to the value -32768, but this doesn't work...

```
IDL> a=-32768
IDL> help,a
A               LONG      =       -32768
```

...it gives a long. This works:

```
IDL> a=-32767-1
IDL> help,a
A               INT       =   -32768
```

but it's a strange thing to have to do.

Greg

Subject: Re: -32768
Posted by skymaxwell@gmail.com on Wed, 19 Mar 2014 05:02:12 GMT
View Forum Message <> Reply to Message

> I'd like to set an integer variable to the value -32768, but this doesn't work...
>
>
>
> IDL> a=-32768
>
> IDL> help,a
>
> A   LONG   =   -32768
>
>
>
> ...it gives a long. This works:
>
>
>
> IDL> a=-32767-1
>
> IDL> help,a
>
> A   INT   =   -32768

>
>
>
> but it's a strange thing to have to do.
>
>
>
> Greg

You can use FIX function

IDL> a=FIX(-32768)
IDL> help,a
A INT = -32768

Subject: Re: -32768
Posted by Craig Markwardt on Wed, 19 Mar 2014 19:17:31 GMT
View Forum Message <> Reply to Message

On Wednesday, March 19, 2014 12:43:59 AM UTC-4, greg...@googlemail.com wrote:
> I'd like to set an integer variable to the value -32768, but this doesn't work...
>
>
>
> IDL> a=-32768

Try
A = -32768L

The "L" indicates "L"ong integer. By default, IDL is stuck in the early 1990s when 16-bit integers were the standard. The maximum positive value representable by a 16-bit number is 32767.

Craig

Subject: Re: -32768
Posted by Rob Klooster on Thu, 20 Mar 2014 09:24:59 GMT
View Forum Message <> Reply to Message

The reason it does not work, is that you first try to define +32768 and then negate the result. Since +32768 can only be expressed as a long, you end up with a long variable. Using fix is a solution, but you could also use the bitwise not operator, if you feel adventurous:
IDL> a = not 32767
IDL> help, a
A INT = -32768

Subject: Re: -32768
Posted by Karl[1] on Thu, 20 Mar 2014 17:01:01 GMT
View Forum Message <> Reply to Message

On Thursday, March 20, 2014 3:24:59 AM UTC-6, Rob Klooster wrote:
> The reason it does not work, is that you first try to define +32768 and then negate the result. Since +32768 can only be expressed as a long, you end up with a long variable. Using fix is a solution, but you could also use the bitwise not operator, if you feel adventurous:
>
> IDL> a = not 32767
>
> IDL> help, a
>
> A INT = -32768

And whether this is a "bug" in IDL or not depends on how IDL defines literal constants.

If an IDL (integral) constant has the form (excuse my abuse of regex):

[+-][0-9]*

then one could argue this is be a bug.

If a constant can be only:

[0-9]*

then, the '-' is taken as an operator and this would not be considered an IDL bug.

C allows:

short a = -32768;

but warns about:

short a = -32769;

on a machine where a short is 16 bits.

Subject: Re: -32768
Posted by Jim Pendleton on Thu, 20 Mar 2014 21:11:07 GMT
View Forum Message <> Reply to Message

On Thursday, March 20, 2014 11:01:01 AM UTC-6, Karl wrote:
> On Thursday, March 20, 2014 3:24:59 AM UTC-6, Rob Klooster wrote:
>
>> The reason it does not work, is that you first try to define +32768 and then negate the result.

Since +32768 can only be expressed as a long, you end up with a long variable. Using fix is a solution, but you could also use the bitwise not operator, if you feel adventurous:
>
>>
>
>> IDL> a = not 32767
>
>>
>
>> IDL> help, a
>
>>
>
>> A INT = -32768
>
>
>
> And whether this is a "bug" in IDL or not depends on how IDL defines literal constants.
>
>
>
> If an IDL (integral) constant has the form (excuse my abuse of regex):
>
>
>
> [+-][0-9]*
>
>
>
> then one could argue this is be a bug.
>
>
>
> If a constant can be only:
>
>
>
> [0-9]*
>
>
>
> then, the '-' is taken as an operator and this would not be considered an IDL bug.
>
>
>
> C allows:
>
>

>
> short a = -32768;
>
>
>
> but warns about:
>
>
>
> short a = -32769;
>
>
>
> on a machine where a short is 16 bits.

The docs on numeric constants imply that the "+" and "-" are considered operators.

http://www.exelisvis.com/docs/Defining_and_Using_Const.html

This can also be tested experimentally by writing a DLM that checks if an input IDL_VARIABLE is an expression or a constant.

Subject: Re: -32768
Posted by Yngvar Larsen on Thu, 20 Mar 2014 21:30:49 GMT
View Forum Message <> Reply to Message

On Thursday, 20 March 2014 18:01:01 UTC+1, Karl wrote:
> On Thursday, March 20, 2014 3:24:59 AM UTC-6, Rob Klooster wrote:
>
>> The reason it does not work, is that you first try to define +32768 and then negate the result. Since +32768 can only be expressed as a long, you end up with a long variable. Using fix is a solution, but you could also use the bitwise not operator, if you feel adventurous:
>
>> IDL> a = not 32767
>> IDL> help, a
>> A INT = -32768

Just for completeness of the thread.

(1) A literal short fails here:

```
IDL> help, -32768S
help, -32768
      ^
% Integer constant must be less than 32768.
```

I consider this a bug.

(2) The following silly tricks work as expected:

```
IDL> help, -32767S-1
<Expression>    INT       =   -32768
IDL> help, 32767S+1
<Expression>    INT       =   -32768
```

(3) Generalizing to 32- and 64-bits integers:

```
IDL> help, -2147483648L

help, -2147483648
      ^
% Long integer constant must be less than 2147483648.

IDL> help, -9223372036854775808LL
<Expression>    LONG64    =   -9223372036854775808
IDL> help, -9223372036854775809LL
<Expression>    LONG64    =    9223372036854775807
```

Thus, we have the same bug for 32-bit integers, but not for 64-bit!!

```
IDL> print, !version
{ x86_64 linux unix linux 8.2.2 Jan 23 2013      64      64}
```

--
Yngvar

Subject: Re: -32768
Posted by chris_torrence@NOSPAM on Thu, 20 Mar 2014 23:44:17 GMT
View Forum Message <> Reply to Message

On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:
>
>
> IDL> help, -9223372036854775808LL
>
> <Expression>   LONG64   =   -9223372036854775808
>
> IDL> help, -9223372036854775809LL
>
> <Expression>   LONG64   =   9223372036854775807
>
>
>

> Thus, we have the same bug for 32-bit integers, but not for 64-bit!!
>
>

Not to start a flame war, but I would argue that 64-bit has the bug, and that 16-bit and 32-bit are behaving correctly. It seems bad that the 64-bit case is quietly returning what could arguably be the wrong value.

As an aside, in the original post, the user stated that they want to "set an integer variable..." But in IDL, there is no such thing as an "integer variable" because IDL is loosely typed. There are just variables that happen to have integer type, but only after they've been created.

Jim P. is correct that the minus sign is really an operator. So it really is a runtime error to write "a = 32768s", regardless of whether you are then going to take the negative of it.

Cheers,
Chris
ExelisVIS

Subject: Re: -32768
Posted by Yngvar Larsen on Fri, 21 Mar 2014 18:02:08 GMT
View Forum Message <> Reply to Message

On Friday, 21 March 2014 00:44:17 UTC+1, Chris Torrence wrote:
> On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:
>
>> IDL> help, -9223372036854775808LL
>> <Expression> LONG64 = -9223372036854775808
>> IDL> help, -9223372036854775809LL
>> <Expression> LONG64 = 9223372036854775807
>>
>> Thus, we have the same bug for 32-bit integers, but not for 64-bit!!
>
> Not to start a flame war, but I would argue that 64-bit has the bug, and that 16-bit and 32-bit are behaving correctly. It seems bad that the 64-bit case is quietly returning what could arguably be the wrong value.

For literals, I agree that this might be regarded as dubious. But in general, the fact that N-bit integer arithmetic is silently wrapped modulo 2^N should be regarded as semantically well defined. If not, I believe a serious performance hit would result.

> As an aside, in the original post, the user stated that they want to "set an integer variable..." But in IDL, there is no such thing as an "integer variable" because IDL is loosely typed. There are just variables that happen to have integer type, but only after they've been created.
>
>
> Jim P. is correct that the minus sign is really an operator. So it really is a runtime error to write

"a = 32768s", regardless of whether you are then going to take the negative of it.

Fair enough. In that case, what is missing is a way to enter a literal negative number. "-376768" _is_ a valid 16-bit signed integer after all! Not really a problem, since negation of a single number hardly is a big performance hit... Also, the OPs problem hasn't hit me at all during the 15+ years I've been using IDL. And there are at least 3 simple (bit silly) workarounds, already mentioned in this thread.

--
Yngvar

Subject: Re: -32768
Posted by Yngvar Larsen on Fri, 21 Mar 2014 18:04:02 GMT
View Forum Message <> Reply to Message

>> On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:

> Fair enough. In that case, what is missing is a way to enter a literal negative number. "-376768" _is_ a valid 16-bit signed integer after all!

... maybe not! That should obviously read "-32768" :)

--
Yngvar

Subject: Re: -32768
Posted by Lajos Foldy on Fri, 21 Mar 2014 18:16:40 GMT
View Forum Message <> Reply to Message

On Friday, March 21, 2014 7:02:08 PM UTC+1, Yngvar Larsen wrote:

> For literals, I agree that this might be regarded as dubious. But in general, the fact that N-bit integer arithmetic is silently wrapped modulo 2^N should be regarded as semantically well defined. If not, I believe a serious performance hit would result.

The C standard guarantees modulo 2^N arithmetic for unsigned integers only. For signed integers, the behavior is undefined.

-32768 is a funny number. Did you know that its absolute value is negative?

IDL> help, abs(-32767-1)
<Expression> INT = -32768

This is the consequence of the asymmetric representation of integers, 32768 can not be represented as an int and there is no NaN for integers.

regards,
Lajos

Subject: Re: -32768
Posted by Yngvar Larsen on Fri, 21 Mar 2014 18:39:45 GMT
View Forum Message <> Reply to Message

On Friday, 21 March 2014 19:16:40 UTC+1, fawltyl...@gmail.com wrote:
> On Friday, March 21, 2014 7:02:08 PM UTC+1, Yngvar Larsen wrote:
>
>
>
>> For literals, I agree that this might be regarded as dubious. But in general, the fact that N-bit integer arithmetic is silently wrapped modulo 2^N should be regarded as semantically well defined. If not, I believe a serious performance hit would result.
>
> The C standard guarantees modulo 2^N arithmetic for unsigned integers only. For signed integers, the behavior is undefined.

Thanks for the info! I didn't know that. Fortunately, I haven't had to do much numerical work in C lately.

Do you by any chance know the rationale behind that?

> -32768 is a funny number. Did you know that its absolute value is negative?
>
> IDL> help, abs(-32767-1)
> <Expression> INT = -32768

Yes! In fact, we've been internally misusing that fact in an IDL routine at work :) I guess we probably shouldn't, but it worked to well in the context to leave it alone!

--
Yngvar

Subject: Re: -32768
Posted by Lajos Foldy on Fri, 21 Mar 2014 19:55:37 GMT
View Forum Message <> Reply to Message

On Friday, March 21, 2014 7:39:45 PM UTC+1, Yngvar Larsen wrote:

>> The C standard guarantees modulo 2^N arithmetic for unsigned integers only. For signed integers, the behavior is undefined.
>

> Thanks for the info! I didn't know that. Fortunately, I haven't had to do much numerical work in C lately.
>
> Do you by any chance know the rationale behind that?

Optimization. There is an excellent blog post about undefined behavior at:

http://blog.llvm.org/2011/05/what-every-c-programmer-should- know.html

regards,
Lajos

Subject: Re: -32768
Posted by Dick Jackson on Sat, 22 Mar 2014 20:20:36 GMT
View Forum Message <> Reply to Message

On Friday, March 21, 2014 11:02:08 AM UTC-7, Yngvar Larsen wrote:
> On Friday, 21 March 2014 00:44:17 UTC+1, Chris Torrence wrote:
>> On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:
>
>> Jim P. is correct that the minus sign is really an operator. So it really is a runtime error to write "a = 32768s", regardless of whether you are then going to take the negative of it.
>
>
>
> Fair enough. In that case, what is missing is a way to enter a literal negative number. "-376768" _is_ a valid 16-bit signed integer after all! Not really a problem, since negation of a single number hardly is a big performance hit... Also, the OPs problem hasn't hit me at all during the 15+ years I've been using IDL. And there are at least 3 simple (bit silly) workarounds, already mentioned in this thread.

And for our further pedantry, if you really need to know which is most time efficient (at least on my MacBook, IDL 8.2):

IDL> tic & for i=1,1e7 do a=FIX(-32768) & toc
% Time elapsed: 1.7954290 seconds.

IDL> tic & for i=1,1e7 do a=FIX('8000'X) & toc ; Hex code depends on hardware "endian-ness"
% Time elapsed: 1.4319592 seconds.

IDL> tic & for i=1,1e7 do a=not 32767S & toc
% Time elapsed: 1.1942451 seconds.

IDL> tic & for i=1,1e7 do a=-32767S-1S & toc ; The winner!
% Time elapsed: 1.1435649 seconds.

Notice how much longer this takes compared to a simple, positive literal:

IDL> tic & for i=1,1e7 do a=32767S & toc
% Time elapsed: 0.39454412 seconds.

Cheers,
-Dick

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

Subject: Re: -32768
Posted by Lajos Foldy on Sat, 22 Mar 2014 20:58:16 GMT
View Forum Message <> Reply to Message

On Saturday, March 22, 2014 9:20:36 PM UTC+1, Dick Jackson wrote:
> On Friday, March 21, 2014 11:02:08 AM UTC-7, Yngvar Larsen wrote:
>
>> On Friday, 21 March 2014 00:44:17 UTC+1, Chris Torrence wrote:
>
>>> On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:
>
>>
>
>>> Jim P. is correct that the minus sign is really an operator. So it really is a runtime error to write "a = 32768s", regardless of whether you are then going to take the negative of it.
>
>>
>
>>
>
>>
>
>> Fair enough. In that case, what is missing is a way to enter a literal negative number. "-376768" _is_ a valid 16-bit signed integer after all! Not really a problem, since negation of a single number hardly is a big performance hit... Also, the OPs problem hasn't hit me at all during the 15+ years I've been using IDL. And there are at least 3 simple (bit silly) workarounds, already mentioned in this thread.
>
>
>
> And for our further pedantry, if you really need to know which is most time efficient (at least on my MacBook, IDL 8.2):
>
>
>
> IDL> tic & for i=1,1e7 do a=FIX(-32768) & toc
>

> % Time elapsed: 1.7954290 seconds.
>
>
>
> IDL> tic & for i=1,1e7 do a=FIX('8000'X) & toc ; Hex code depends on hardware "endian-ness"
>
> % Time elapsed: 1.4319592 seconds.
>
>
>
> IDL> tic & for i=1,1e7 do a=not 32767S & toc
>
> % Time elapsed: 1.1942451 seconds.
>
>
>
> IDL> tic & for i=1,1e7 do a=-32767S-1S & toc ; The winner!
>
> % Time elapsed: 1.1435649 seconds.
>
>
>
> Notice how much longer this takes compared to a simple, positive literal:
>
>
>
> IDL> tic & for i=1,1e7 do a=32767S & toc
>
> % Time elapsed: 0.39454412 seconds.
>
>
>
> Cheers,
>
> -Dick
>
>
>
> Dick Jackson Software Consulting Inc.
>
> Victoria, BC, Canada --- www.d-jackson.com

I think the winner is: a='8000'XS

regards,
Lajos

Subject: Re: -32768
Posted by Dick Jackson on Sat, 22 Mar 2014 23:56:29 GMT
View Forum Message <> Reply to Message

fawltylanguage@gmail.com wrote, On 2014-03-22, 1:58pm:
> On Saturday, March 22, 2014 9:20:36 PM UTC+1, Dick Jackson wrote:
>> On Friday, March 21, 2014 11:02:08 AM UTC-7, Yngvar Larsen wrote:
>>> On Friday, 21 March 2014 00:44:17 UTC+1, Chris Torrence wrote:
>>>> On Thursday, March 20, 2014 3:30:49 PM UTC-6, Yngvar Larsen wrote:
>>
>>>> Jim P. is correct that the minus sign is really an operator. So it really is a runtime error to write "a = 32768s", regardless of whether you are then going to take the negative of it.
>>
>>> Fair enough. In that case, what is missing is a way to enter a literal negative number. "-376768" _is_ a valid 16-bit signed integer after all! Not really a problem, since negation of a single number hardly is a big performance hit... Also, the OPs problem hasn't hit me at all during the 15+ years I've been using IDL. And there are at least 3 simple (bit silly) workarounds, already mentioned in this thread.
>>
>> And for our further pedantry, if you really need to know which is most time efficient (at least on my MacBook, IDL 8.2):
>>
>> IDL> tic & for i=1,1e7 do a=FIX(-32768) & toc
>> % Time elapsed: 1.7954290 seconds.
>>
>> IDL> tic & for i=1,1e7 do a=FIX('8000'X) & toc ; Hex code depends on hardware "endian-ness"
>> % Time elapsed: 1.4319592 seconds.
>>
>> IDL> tic & for i=1,1e7 do a=not 32767S & toc
>> % Time elapsed: 1.1942451 seconds.
>>
>> IDL> tic & for i=1,1e7 do a=-32767S-1S & toc ; The winner!
>> % Time elapsed: 1.1435649 seconds.
>>
>> Notice how much longer this takes compared to a simple, positive literal:
>>
>> IDL> tic & for i=1,1e7 do a=32767S & toc
>> % Time elapsed: 0.39454412 seconds.
>
>
> I think the winner is: a='8000'XS
>
> regards,
> Lajos

Ooh, right you are! (I missed that one) Excellent, thank you.

Again, though, the hex code depends on hardware "endian-ness".

```
IDL> tic & for i=1,1e7 do a='8000'XS & toc
% Time elapsed: 0.38863897 seconds.
```

--

Cheers,
-Dick

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