Subject: Is this a bug?
Posted by David Foster on Wed, 30 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Platform: Sun Sparc2, Solaris 2.5 Patched
IDL Version: 5.0.3

Can someone please explain to me why the following is happening?

```
p = ptr_new( {a:1, b:2, s:{x:0,y:0, a:[256,256,48]} } )

print, ((*p).s.a)[2]
      48

((*p).s.a)[2] = ((*p).s.a)[2] * 4
% Temporary variables are still checked out - cleaning up...

print, ((*p).s.a)[2]
      48                        ; Value was not adjusted

help, ((*p).s.a)[2]
<Expression>    INT       =      48

print, ((*p).s.a)
     256     256      48

(*p).s.a[2] = (*p).s.a[2] * 4

print, ((*p).s.a)[2]
     192                        ; Value *was* adjusted
```

Is ((*p).s.a)[2] an invalid construct? IDL doesn't seem to have problems with it in the PRINT command, and it sure seems ok to me. The message about temporary variables seems to be a clue, but I need some help on this one.

Thanks!

Dave
--

~~~~~~ ~~~~~~
David S. Foster Univ. of California, San Diego
Programmer/Analyst Brain Image Analysis Laboratory
foster@bial1.ucsd.edu Department of Psychiatry
(619) 622-5892 8950 Via La Jolla Drive, Suite 2240
La Jolla, CA 92037
~~~~~~ ~~~~~~

Subject: Re: Is this a bug?
Posted by steinhh on Thu, 01 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <6uuvndail1@nnrp1.dejanews.com>
menakkis@my-dejanews.com writes:

> David Foster <foster@bial1.ucsd.edu> wrote:
> <...>
>> ((*p).s.a)[2] = ((*p).s.a)[2] * 4
>> % Temporary variables are still checked out - cleaning up...

[..and (*p).s.a[2] is not adjusted..]

> I think something like this came up in the newsgroup a while back, and the
> problem was that the brackets on the LHS effectively turned it into a RHS,
> with the REAL LHS becoming a temporary variable.

I think this is the correct "explanation": The left hand side is
not an "lvalue" (from C terminology - "left hand side value").

The bug here is (IMO) not that (*p).s.a[2] is not adjusted, it is
the fact that you don't get an error!.

Whenever you put brackets around anything *but* a simple variable,
IDL makes a temporary variable (a "right hand value" - rvalue) out
of it, and *then* decides how to do stuff outside the
brackets. This is *why* putting brackets around e.g., function
calls will allow you to do this: print,(size(a))[0].

You may not index a function [call], but you may index a temporary
variable. But the line ((*p).s.a)[2] = ((*p).s.a)[2] * 4 is thus
analogous to:

IDL> a=fltarr(5)
IDL> (a*5) = 6
% Attempt to store into an expression: <FLOAT Array[5]>.

...but not *quite*, since IDL doesn't give the proper error!

Come to think of it, it may be more like:

IDL> (a[0:2])[2] = 6
% Temporary variables are still checked out - cleaning up...

Yep - that's it.

Note that the *pointer* has nothing to do with it:

```
IDL> st={a:1, b:2, s:{x:0,y:0, a:[256,256,48]} }
IDL> (st.s.a)[2]=2
% Temporary variables are still checked out - cleaning up...
```

I guess it's the fact that you're first picking out a *part* of
the structure, then putting brackets around it, then storing into
part of it.

> [....]
> I guess an interpretation of what (*p) means is: "make a
> temporary variable *control* structure to get at the contents of p";
> you can then use the "thing" (*p) as if it was a variable name.
> It takes some getting used-to, but it *is* very efficient for
> dereferencing just an array element or structure member.
> (I was going to sound forth on how I dislike this syntax but had
> second thoughts.)

My shot: The sequence of letters "*p" is syntactically identical to
a variable name. The value of the variable called "*p" is what is
pointed to by the pointer "p" (obvious, but somehow I had to say
it).

The "problem" (not really, read on!) is that structure dereference
binds harder than the pointer dereference.

(Indeed, RSI doesn't seem to think that "." is an operator at all!
That must be the reason why we didn't get the "p->element"
notation as a shorthand for "(*p).element" as in C. I would
still like to see it!).

So to overcome the binding problem we use the (*p) construct. But,
according to "my" rule above, this is analogous to "(variable)".
And lo and behold, the parenthesis is *not* a problem:

```
IDL> a=fltarr(3)
IDL> (a)[1]=55
IDL> print,a
      0.00000      55.0000      0.00000
```

So to sum up:

1. Brackets around anything *except* a simple variable name gives
   an rvalue (expression).
2. Dereferenced pointers -- "*p" behave as a simple variable name,
   -> see exception in first point
3. Rvalues should *not* be allowed on the left hand side of
   assignments (fix it, RSI).

Finally, I guess that Peter Mason changing his mind in the middle
of a paragraph caused him to say things about the

    (<expression>)[index]

construct whilst appearing to talk about the (*p) construct..?

Regards,

Stein Vidar

Subject: Re: Is this a bug?
Posted by rmlongfield on Thu, 01 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <36129C40.297A@bial1.ucsd.edu>,
  David Foster <foster@bial1.ucsd.edu> wrote:
> Platform: Sun Sparc2, Solaris 2.5 Patched
> IDL Version: 5.0.3
>
> Can someone please explain to me why the following is happening?
>
>   p = ptr_new({a:1, b:2, s:{x:0,y:0, a:[256,256,48]} })
>
>   print, ((*p).s.a)[2]
>   48
>
>   ((*p).s.a)[2] = ((*p).s.a)[2] * 4
>   % Temporary variables are still checked out - cleaning up...
>
>   print, ((*p).s.a)[2]
>   48   ; Value was not adjusted
>
>   help, ((*p).s.a)[2]
>   <Expression>   INT   =   48
>
>   print, ((*p).s.a)
>   256   256   48
>
>   (*p).s.a[2] = (*p).s.a[2] * 4
>
>   print, ((*p).s.a)[2]
>   192   ; Value *was* adjusted
>
> Is ((*p).s.a)[2] an invalid construct? IDL doesn't seem to have
> problems with it in the PRINT command, and it sure seems ok to
> me. The message about temporary variables seems to be a clue,

> but I need some help on this one.
>
> Thanks!
>
> Dave
> --
>
> ~~~~~~ ~~~~~~
> David S. Foster Univ. of California, San Diego
> Programmer/Analyst Brain Image Analysis Laboratory
> foster@bial1.ucsd.edu Department of Psychiatry
> (619) 622-5892 8950 Via La Jolla Drive, Suite 2240
> La Jolla, CA 92037
> ~~~~~~ ~~~~~~
>

Hi Dave, That's a pretty interesting (and disturbing) bug. I get the same results on my SGI. The temporary variable message is a puzzle because I ran your program just after entering IDL. Usually I get this message after a program crashes and I forget to type RETALL. (I don't know what this does internally, just that the book says that I should do it)

Based on my experience with pointers to structures with pointers to arrays (ugh!), this construction: (*p).s.a[2] looks more correct than this one: ((*p).s.a)[2] , although I have never made this particular construction. I've also noticed that IDL does not discuss all variations. However, I use PRINT statements to check whether something is working properly. The fact that it doesn't in your case is worrisome.

I hope someone has an answer.

Rose

Subject: Re: Is this a bug?
Posted by menakkis on Thu, 01 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

David Foster <foster@bial1.ucsd.edu> wrote:
<...>
> ((*p).s.a)[2] = ((*p).s.a)[2] * 4
> % Temporary variables are still checked out - cleaning up...
>
> print, ((*p).s.a)[2]
> 48 ; Value was not adjusted

I think something like this came up in the newsgroup a while back, and the problem was that the brackets on the LHS effectively turned it into a RHS, with the REAL LHS becoming a temporary variable.

mmm. I didn't do too well there. How about... You want ((*p).s.a)[2] to receive a value. But IDL sees those outer brackets and decides: "I've got to calculate something here; I'll stick the result in a temporary variable, like I usually do." So you get a temporary variable containing (*p).s.a (or maybe even ((*p).s or *p ???). Finally, IDL assigns your (actual) RHS result to element 2 of the "a" member of this temporary variable, and then chucks away the temporary variable because, as far as it's concerned, the temporary variable has nowhere to go.

Perhaps like me you're a bit uneasy about having to put brackets around dereferenced pointers to get at structure members or array elements therein. It really does go against the grain, doesn't it? Any superfluous brackets and you get this "calculation mode" thing kicking in. I must say that I can't see a way around this, though, given what I gather about how IDL's variables work. I guess an interpretation of what (*p) means is: "make a temporary variable *control* structure to get at the contents of p"; you can then use the "thing" (*p) as if it was a variable name. It takes some getting used-to, but it *is* very efficient for dereferencing just an array element or structure member. (I was going to sound forth on how I dislike this syntax but had second thoughts.)

Peter Mason

Subject: Re: Is this a bug?
Posted by J.D. Smith on Fri, 02 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan wrote:
> Whenever you put brackets around anything *but* a simple variable,
> IDL makes a temporary variable (a "right hand value" - rvalue) out
> of it, and *then* decides how to do stuff outside the
> brackets. This is *why* putting brackets around e.g., function
> calls will allow you to do this: print,(size(a))[0].
>

As a side note, it is this same behavior which allows you to perform another useful trick, often overlooked:

b=(a=val)

This will assign val to b and a in one shot. Furthermore,

d=(c=(b=(a=val)))

will work just as well.

See, rvalues aren't all bad.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Is this a bug?
Posted by J.D. Smith on Fri, 02 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

menakkis@my-dejanews.com wrote:
>
> David Foster <foster@bial1.ucsd.edu> wrote:
> <...>
>> ((*p).s.a)[2] = ((*p).s.a)[2] * 4
>> % Temporary variables are still checked out - cleaning up...
>>
>> print, ((*p).s.a)[2]
>> 48 ; Value was not adjusted
>
 < SNIP >
> Perhaps like me you're a bit uneasy about having to put brackets around
> dereferenced pointers to get at structure members or array elements therein.
> It really does go against the grain, doesn't it? Any superfluous brackets
> and you get this "calculation mode" thing kicking in. I must say that I
> can't see a way around this, though, given what I gather about how IDL's
> variables work. I guess an interpretation of what (*p) means is: "make a
> temporary variable *control* structure to get at the contents of p"; you can
> then use the "thing" (*p) as if it was a variable name. It takes some
> getting used-to, but it *is* very efficient for dereferencing just an array
> element or structure member. (I was going to sound forth on how I dislike
> this syntax but had second thoughts.)

I would interpret (*p) as "lookup and use the heap variable pointed to

by p". This is for all purposes the same meaning as *p, because a temporary copy of a heap variable is not well-defined, since heap variables are global and long-lived by definition. That is, even if IDL is making a temporary copy of whatever internal variable (likely a C pointer) is tied to <PtrHeapVar1>, this is just as valid a reference to the global heap as *p. It is unclear if *p vs. (*p) actually elicits different internal handling by IDL (with perhaps more overhead for the latter). Even if so, however, it will not make any difference to our usage. In this context the ambiguity is not as alarming.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Is this a bug?
Posted by menakkis on Fri, 02 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

steinhh@ulrik.uio.no (Stein Vidar Hagfors Haugan) wrote:
<...>
> 1. Brackets around anything *except* a simple variable name gives
> an rvalue (expression).
> 2. Dereferenced pointers -- "*p" behave as a simple variable name,
> -> see exception in first point
> 3. Rvalues should *not* be allowed on the left hand side of
> assignments (fix it, RSI).

It looks like you have it. I'd go so far as to add that: brakets around a simple variable name are effectively ignored / discarded. e.g., A=INDGEN(4) &(((A)))[2]=20 works.

> Finally, I guess that Peter Mason changing his mind in the middle
> of a paragraph caused him to say things about the
> (<expression>)[index]
> construct whilst appearing to talk about the (*p) construct..?

I guess I should have thrown in a <NL> there for your paragraph parser :-)

Peter Mason