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 <6uuvnd$ail$1@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