Subject: Operator precedence
Posted by Harvey Rarback on Mon, 10 Jul 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Folks,

I have a couple of questions regarding operator precedence. From this newsgroup and some experimentation I believe the following statement is true:

Structure field extraction and array indexing have equal precedence, higher than pointer dereference but lower than parentheses to group expressions.

Is this statement true?

So for nested structures struct1.struct2.data produces the same result as (struct1.struct2).data as expected. However, for nested objects (example code appended) these rules don't seem to apply:

obj1.obj2.data produces an error
(obj1.obj2).data produces the expected result, along with the infamous
% Temporary variables are still checked out - cleaning up...

Can some kind soul enlighten me about this behavior?

Thanks.
--Harvey

_____Example Code_____

```
pro obj2__define
obj2 = {obj2, data:0}
end

;************************************************************ *****************

function obj1::init
self.obj2 = obj_new('obj2')
return,1
end

;************************************************************ *****************

pro obj1__define
obj1 = {obj1, obj2:obj_new()}
end

;************************************************************ *****************
```

```
pro obj1::test

obj1 = self
; next line produces % Expression must be a structure in this context: <No name>
;print, obj1.obj2.data

; next line prints data but produces
; % Temporary variables are still checked out - cleaning up...
print, (obj1.obj2).data

;next line produces % Expression must be a structure in this context: OBJ2.
;print, obj1.(obj2.data) ;

end

;********************************************************** *****************

pro test_has_a

obj1 = obj_new('obj1')
obj1->test

end
```