Subject: Re: Functions and arrays
Posted by steinhh on Fri, 06 Dec 1996 08:00:00 GMT
View Forum Message <> Reply to Message

In article <5877p7$6gp@post.gsfc.nasa.gov>, thompson@orpheus.nascom.nasa.gov (William Thompson) writes:
|>
|>On 4 Dec 1996, Stein Vidar Hagfors Haugan wrote:
|> >> Or at least get a compile time error about the possible mixup,
|> >> something like "Error: test1 interpreted as a function in line 5,
|> >> but as a variable in line 10".
|>
|> Peter Mason <peterm@demsyd.syd.dem.csiro.au> writes:
|> >I think that this would be a good idea.   It wouldn't bust any code that
|> >wasn't already hovering on the edges of "busthood", and it would catch many
|> >of the ambiguities.   When requested to compile a function, IDL could stop with
|> >an error if a variable of the same name already existed.   ...
|>
|> There seems to be a assumption here that the *PROGRAMMER* is forming an
|> ambiguity by trying to use the same name for both a variable and a function in
|> the same routine. I argue that it's *IDL* which is responsible for the
|> ambiguity. The situation I ran into was when the software was written in a
|> self-consistent manner--the name was intended to refer to a variable throughout
|> the routine. However, IDL on its own decided to sometimes interpret the call
|> as a variable (correct) and sometimes as a function (incorrect), depending on
|> what applications were started first in the IDL session.

Just to clear up any misunderstandings: I do not see this in any way as a
programmer error. It is a *compiler* error. My preferred solution was exactly the
same as you state, Bill:

|> The correction to this is quite simple. If a name is used for a variable in a
|> subroutine, then it refers to that variable throughout that subroutine. The

The reason I included other options for minimizing (though not eliminating) the
problem is that a rewrite of IDL's compiler procedures might be too complicated
to appear any time soon (version 6? or 7?). Consider the following:

```
pro compiler_trouble

  while n_elements(value) eq 0 do begin
    if n_elements(dummy) eq 0 then begin
       value = arr(0)
    end

    dummy = 1
    arr = fltarr(5)
  endwhile
```

end

and couple that with the possible existence of a function "arr".

Although the definition (or lack thereof) of IDL as a language does
not create any problems in resolving the conflict in the only reasonable
fashion (always interpreting arr as a variable inside pro compiler_trouble),
I fear that the current IDL compiler is not up to that task without
at least some medium-sized rewrite.

So, in case the only decent thing cannot be done soon, I'd like to see *something*
done about the problem, as a temporary solution.

Including both a "forward_variable" statement *and* a compiler message about a
potential *compiler* error would at least cut the time spent searching for
mysterious bugs down to almost zero. Granted, it would still allow a lot of
perfectly fine routines to suddenly become non-functional simply because someone
happened to add some function, and I agree, that's not acceptable. But a temporary
solution is better than *no* solution.

A "forward_variable" statement could of course be implemented with no additions
to the language, with a statement like "if 0 then arr = 0". That shouldn't
leave much room for doubt if the compiler would listen.....

Stein Vidar