Subject: Re: Recursion in IDL
Posted by thompson on Tue, 06 Apr 1993 13:39:40 GMT
View Forum Message <> Reply to Message

alans@ll.mit.edu (Alan Jay Stein) writes:

> Recursion only works on procedures, though. It would be far more useful
> for IDL to support recursive *functions*. Unfortunately, this is more
> difficult for IDL's parser, I guess, because it has to distinguish between a
> function call and an array subscript operation which share the same syntax.
> E.G.,

> function foo, bar
> if (bar eq 1) then $
> return, 1 else $
> return, bar * foo (bar-1)
> end

> IDL> print,foo(2)
> % Variable is undefined: FOO.
> % Execution halted at FOO </dev/tty(1)> .
> % Called from FOO </dev/tty(2)>.
> % Called from $MAIN$.

> Yecch! Does anyone know a way around this? Compile it twice? (Yes,
> actually I just tried and it *does* work if you ".run" it twice. (*WOW*) So
> I take it all back; IDL *does* support recursive functions, albeit with an
> ugly hack. For the record, I'm running IDL 3.0.0 on Sparc SunOs 4.1.1...

> RSI, I understand the difficulty involved, but can something be done about
> that? I never explicitly ".run" anything; I put all procedures & functions
> in my IDL path, and have encouraged my colleagues to do the same...Gee,
> think of the CPU-hogging Object-Oriented possibilities with this - functions
> which walk structures of structures looking for a particular field names,
> recursive "sizeof"-type operations, etc. It boggles the imagination! ;-)
> --
> Alan J.Stein MIT/Lincoln Laboratory alans@ll.mit.edu

Actually, I can't make your example fail. If I enter "print,foo(2)" then I get the answer "2". I'm using IDL v3.0.0 on SunOS 4.1.2.

Sometimes I've found that IDL can get confused about whether something's a variable or a function when it tries to compile a routine and fails. Then an extra .run seems to solve the problem.

In any case, what you might try is to put a dummy keyword in your function definition. Then the IDL shouldn't have any problems determining that it's a

function and not a variable. For example

```
function foo, bar, recursive=recursive
  if (bar eq 1) then $
    return, 1 else $
    return, bar * foo (bar-1,/recursive)
end
```

The recursive keyword above doesn't actually do anything, but it makes it obvious that foo is a function.

Bill Thompson
