

Re: How come Ada isn't more popular?

Source: <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2007-02/msg00375.html>

- *From:* Maciej Sobczak <no.spam@xxxxxxxxxxxxx>
 - *Date:* Fri, 09 Feb 2007 09:17:47 +0100
-

Dmitry A. Kazakov wrote:

For handling cycles there are weak pointers.

Not only. If you have cycles, then you'd better rethink the design.

The difference is between a) graph treated as a mesh (or mess) of nodes which "own" each other and b) graph treated as a collection of nodes.

The former might have ownership cycles between nodes, but not the latter, where ownership is an acyclic relation between graph and nodes.

I agree that this kind of restructuring is not always possible, but for me it is conceptually cleaner and worth trying from the beginning.

I would say that in all cases all references within it should be non-controlled. The argumentation could go as follows:

A controlled reference (subject of GC) expresses the life-time relation "not before me." [*] This relation is obviously transitive (due to nature of time). Ergo, there cannot be any cycles, per definition.

From this stand point I would claim that a cycle of controlled references manifests either a bug or a design problem. That a buggy program continues to function as-if there were no bug, barely should be attributed as an advantage of GC systems.

Amen!

Still, there is a strong argument is that for some class of algorithms it might be beneficial to be able to "drop on the floor" a bigger part of the graph altogether. Consider a situation where an algorithm breaks a connection between two nodes in a graph and just loses the interest in the part that was on the other side of the broken connection. It might be a single node, but it might be as well million, with mesh (mess) of connections between them. Reclaiming that abandoned part might require implementing the same tracking logic that GC already provides out of the box and therefore the argument goes that the use of off-the-shelf GC can be beneficial for the memory-management aspect of such an algorithm. (Any thoughts on this?)

Personally, I accept this reasoning and I'm happy that I *can* plug for example the Boehm collector if I find it useful – but at the same time I don't find this class of algorithms to be so widespread as to justify GC as a

Re: How come Ada isn't more popular?

general "paradigm", worth exposure as a driving language feature.

--

Maciej Sobczak : <http://www.msobczak.com/>

Programming : <http://www.msobczak.com/prog/>

.