
Subject: Re: Ptr improve

Posted by [mirek](#) on Tue, 24 May 2011 15:46:17 GMT

[View Forum Message](#) <> [Reply to Message](#)

cbpporter wrote on Tue, 24 May 2011 03:25mirek wrote on Mon, 23 May 2011 23:37cbpporter wrote on Mon, 23 May 2011 07:52Here is an idea that has been going around in my head for a while: what if we combine the U++ way with GC.

GC is great, but the cost of mark & sweep can be too much for some cases.

Traditional memory management is problematic, and we have a relatively big cost of allocation/deallocation.

Well, that is relative, average allocation/deallocation is about pretty fast in U++.... (about the same as link/unlink in double-linked list + a couple of simple loads)

Quote:

With the U++ we have everything belongs somewhere.

GC is quite incompatible with the concept of destructors.

Quote:

But what if we kept the principle intact for non heap allocated objects, but for heap allocated one, the destructor would only mark the object for deletion, and it would actually get deleted later.

IMO results in unmaintainable mess.

Besides, GC AFAIK is still not a part of C++. And you cannot realistically add it by library - the best you can do is stochastic approach (Boehm), which works fine, unless you are processing white noise

Still, what is unclear to me is why to complicate your code with heap if you do not have to? The whole mission of U++ is to eliminate universal shared heap as much as possible.

Mirek

Destructors are not generally incompatible with GC, just the normal GC scenarios and implementation we have now. What I am proposing is reversing the order of the GC flow.

Another additional advantage would be that it would eliminate one of the big disadvantages of conservative GC: false positives and inability to deal with extremely large heaps or data that looks like pointers.

With what I am proposing, only objects whose destructors have been called will be collected, or those that were marked by the API. It is not GC, it is the U++ memory management flow, with one

single difference: data is not deleted immediately. It is delayed.

This would apply to all containers that allocate memory, so it is not about eliminating universal shared heap or not so it wouldn't contrast with our mission statement.

Well, very unlikely to happen, but for the fun of theoretical debate:

How do you plan to implement GC? are you going to write your C++ compiler? How would it handle typical C++ inconsistencies like pointer casts

In the past, I was experimenting with such library based GC approach, where only specific objects were subjects of GC. I was able to get it working, and it was quite optimal, however the whole thing collapsed the moment you have started mixing such objects and regular ones. I have not found a way out. Since then I consider the whole point of GC in C++ moot - perhaps Boehm or is good for fixing leaky code, but my conclusion is that you can either have reliable destructors or GC. There is nothing in between.

Mirek
