CoCoALib - Feature #180

GlobalManager: registration of global variables

07 Jun 2012 23:11 - John Abbott

Status:	Closed	**Start date:**	07 Jun 2012
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Safety	**Estimated time:**	15.00 hours
Target version:	CoCoALib-0.99560	**Spent time:**	5.60 hours

Description

Add a feature to **GlobalManager** which mimics the Unix fn **atexit**.

This allows users to register pseudo-dtors for their global variables, so that everything is cleaned up at the end of the session. What is registered is a procedure without args which empties (or destroys) a global variable; ideally the procedure should not throw.

Note: the registered pseudo-dtors are executed in reverse order (*i.e.* the last registered is the first called). We can probably register the dtors for RingZZ and RingQQ using this mechanism.

Related issues:

Related to CoCoALib - Feature #762: ExternalLib-GFan: first prototype	**Closed**	**28 Aug 2015**
Related to CoCoALib - Design #785: finite fields: global register of fields a...	**New**	**12 Oct 2015**

History

#1 - 01 Aug 2014 08:59 - Anna Maria Bigatti

- Target version set to CoCoALib-1.0

#2 - 01 Sep 2015 11:21 - John Abbott

[during meeting in Aarhus with Anders, Anna and Christof]
It was pointed out that external libs may want to store CoCoALib values in global variables; this proposal would allow such values to be properly cleaned up at the end of program execution.

#3 - 16 Oct 2015 12:11 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I discovered yesterday that this has already been implemented (at least partly).

Furthermore the implementation was not what I expected. Here is a brief description.

- **(A)** the pseudo-dtor to be registered is simply a void (*dtor)()
- **(B)** the pseudo-dtor to be registered is a void (*dtor)(void* ptr) which expects a pointer to the object to be destroyed.

Case **(A)** mimics what atexit does, but case **(B)** could be useful if there are several global objects of the same type.

Case **(B)** has already been implemented (by whom? when?), and also used in RingQQt (in file PolyRing.C:125).

What to do?

The way the pseudo-dtors are registered in RingQQt would not be possible if only interface **(A)** were available.

A pseudo-dtor of type **(A)** could easily be "passed" using the interface **(B)** *e.g.* by supplying a NULL pointer (or any pointer, as the value is ignored). However, it is "annoying" to have supply an unused value.

It would also be possible to offer both interfaces. How to implement this? Internally there must be a single stack, and a stack entry must be capable of representing both a void (*dtor)() and a pair comprising void* and void (*dtor)(void*).

#4 - 16 Oct 2015 14:20 - John Abbott

Here is a "trick" which could work:
define a function void CallDtor(void* dtor) { dtor(); } and then
to register an argumentless pseudo-dtor dtor0 we could register the pointer
&dtor0 and the function CallDtor (which has the correct signature).

This would allow all entries in the pseudo-dtor stack to be pairs of void* and void (*fn)(void* ptr) where the fn is either CallDtor which expects its arg to be a pointer to an argumentless pseudo-dtor, or fn is a 1-arg pseudo-dtor whose arg is a pointer to the object to be destroyed.

Is this a "neat trick" or an "evil hack"? I'm unsure. It uses void* to hide a type ambiguity...

A clean implementation would have space for 3 pointer in each entry:

- void* a point to some object to be destroyed
- void (*dtor1)(void *ptr) pseudo-dtor for passed in object
- void (*dtor0)() argumentless pseudo-dtor

Only one of dtor1 and dtor0 would be non-NULL.
The disadvantage of the clean impl is that it wastes some space: 1 or 2 pointers in each entry would be unused.

#5 - 16 Oct 2015 14:34 - John Abbott

It seems that C++ STL provides no guarantees about the order of destruction of elements in a container: see
http://stackoverflow.com/questions/6169125/order-of-destruction-of-elements-of-an-stdvector

Thus, for instance, a pseudo-dtor for a vector<ring> must go through the rings explicitly one at a time: the following code might work

```
const long n = len(v);
for (long k=n-1; k >= 0; --k)
{
  v.back().RunDtor();
  v.resize(k);
}
```

or maybe the even simpler

```
while (!v.empty())
{
  v.top().RunDtor();
  v.pop();
}
```

NOTE (2015-10-19) my earlier suggestion to make the dtor for the entries in the stack actually run the registered dtors was **A BAD IDEA** because internally the stack operations copy and delete the stack entries, so rings were destroyed too early! Ouch!

#6 - 19 Oct 2015 14:54 - John Abbott

- Assignee set to John Abbott

- % Done changed from 10 to 50

I have implemented the clean version; the "nifty" version can be kept for later if it turns out that the stack of dtors to call becomes enormous... At least the clean version has semantics which are easier to describe.

Wasted lots of time debugging because an implicit hint in comment 5 was **BAD**!!!
The STL stack impl internally calls the dtor for elements of the stack... this made my ring disappear mysteriously 8-0

#7 - 19 Oct 2015 15:08 - John Abbott

I wonder whether the possibility to register the pair ptr+dtor should be removed later on. At the moment it is used only in RingQQt (in file PolyRing.C:125), and that use will probably disappear if/when we implement a global register for rings (assuming that any rings left in the register at the end will be automatically destroyed by GlobalManager).

Perhaps I'll word the documentation to discourage using it...

#8 - 19 Oct 2015 16:53 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 50 to 80

I've added some doc to GlobalManager.txt; it's probably not the best place, but no other place springs to mind.

I've cleaned up the prototype code (almost all out-of-line as there's no need for great speed... you shouldn't be using globals anyway!)

All tests pass. I'll check in when I can get VPN going.

NOTE: if I recall well, C5-GUI sometimes triggers an "imminent disaster" warning when exiting; I was hoping to spot a global variable whose pseudo-dtor can be registered (but I've not yet seen it).

#9 - 22 May 2017 13:38 - Redmine Admin

- Target version changed from CoCoALib-1.0 to CoCoALib-0.99560

#10 - 08 Nov 2017 17:16 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 80 to 100