CoCoALib - Design #1389

myZeroPtr and myOnePtr

09 Jan 2020 16:11 - Anna Maria Bigatti

Status:	Rejected	**Start date:**	09 Jan 2020
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Data Structures	**Estimated time:**	0.55 hour
Target version:	CoCoALib-0.99700	**Spent time:**	0.55 hour

Description

every concrete ring have these lines (in .C file)

```
    mutable MemPool myMemMgr;       // MemPool must come *BEFORE* myZeroPtr and myOnePtr
    unique_ptr<RingElem> myZeroPtr;  ///< Every ring stores its own zero.
    unique_ptr<RingElem> myOnePtr;   ///< Every ring stores its own one.
```

why aren't these in the abstract class?
I think there was a reason, but I can't remember why.
Cache? to have other fields in top position (e.g. myModulus), before myMemMgr?

Related issues:

Related to CoCoA-5 - Support #1387: John's visit Feb 2020	**Closed**	**07 Jan 2020**

History

#1 - 09 Jan 2020 17:01 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I think the clue is in the line above about MemPool.

In the dtor for the ring the fields are destroyed in reverse order to their appearance in the class definition.
To take advantage of the automatic destruction of the values pointed at by the unique_ptr, it is necessary
that the unique_ptr fields come **after** the MemPool field.

I suppose we could move the MemPool into the abstract base class, but it could only be a "fake" MemPool since it does not know what size blocks it has to manage (unless this info is passed in as a ctor arg...).   Hmmm, this might be do-able; not sure yet if it is wise/clever. Need to think.

#2 - 09 Jan 2020 20:59 - John Abbott

- Related to Support #1387: John's visit Feb 2020 added

#3 - 12 Jan 2020 19:00 - John Abbott

- Status changed from In Progress to Rejected

- Assignee set to John Abbott

- % Done changed from 10 to 50

JAA thinks this proposal should be **REJECTED** because it is unsafe.

The crucial point is that if the data fields myOnePtr and myZeroPtr belonged to the base (ring) class then the compiler (and the programmer) expect that these fields are constructed and destroyed by the base class: this cannot happen, because the values can be constructed and destroyed only by the derived (concrete) class.

One could argue that since the values are pointers true construction ca be delayed until the body of the derived class ctor is being executed; this is true (though it does mean that these pointers are in an anomalous state (presumably null) for a little while). **The real risk** is that at destruction time; the dtor for the derived class would have to explicitly destroy the values pointed to by myOnePtr and myZeroPtr, because otherwise the base class will attempt to destroy them **after** the true owning ring itself has been "partly" destroyed. **THIS IS UNSAFE**.

Marking as **rejected**, but done is only 50 until Anna confirms that my arguments are valid.

#4 - 13 Jan 2020 09:36 - Anna Maria Bigatti

- % Done changed from 50 to 100

Thanks for the explanation. Now I will remember why.

#5 - 13 Jan 2020 12:07 - John Abbott

- Target version changed from CoCoALib-0.99800 to CoCoALib-0.99700

#6 - 13 Jan 2020 12:07 - John Abbott

- Estimated time set to 0.55 h