CoCoALib - Design #411

design of factorization template class

23 Oct 2013 12:54 - John Abbott

Status:	Closed	**Start date:**	23 Oct 2013
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Tidying	**Estimated time:**	8.00 hours
Target version:	CoCoALib-0.99532	**Spent time:**	7.25 hours

Description

Which is the better of the following two ways of implementing the factorization class?

(A) the current impl

- record with 3 fields myRemainingFactor, myFactors = list of RingElem, myMultiplicities = list of integers

(B) alternative impl

- record with 2 fields myRemainingFactor, myFacMults = list of "facpows"
- a "facpow" is a record of myFactor and an integer

Approach (A) is probably easier to use, but approach (B) is probably conceptually cleaner.

Related issues:

Related to CoCoALib - Support #391: Check consistency of template class facto...	**Closed**	**23 Jul 2013**
Related to CoCoALib - Design #429: factorization: public data fields, or mem ...	**Closed**	**28 Jan 2014**

History

#1 - 23 Oct 2013 14:28 - John Abbott

This question arose because I have just implemented SqfreeFactor, and thought that it might be nice to guarantee that the multiplicities are in (weakly) increasing order. With the current impl (two separate "lists" for factors and multiplicities) it is not possible to use a built-in STL algorithm for reordering both lists; instead I would have to implement explicitly an ordering algorithm which moves elements coherently in both lists (facs & mults).

Advantages of method (A) over (B):

- the impl is already complete
- it is "simpler" for the implementor (no need to define a type for containing a factor and its multiplicity)
- it is (probably) simpler to use, *e.g.* facs.myFactors[i] and facs.myMultiplicities[i]

Advantages of method (B) over (A):

- the design is conceptually cleaner, a factorization is a "list of factor & multiplicity pairs"
- (sort of) closer to impl used in CoCoA-4
- easier to apply built-in STL algms to a single "list" (rather then two "parallel lists")
- but **harder** to use, *e.g.* facs.myFacMults.myFactors[i] and facs.myFacMults.myMultiplicities[i]

Currently JAA prefers method (A) because:

1. it's already impl'ed, so no extra work needed
2. it is easier to use (think of KISS paradigm)
3. the greater ease of applying STL algms will be appreciated only quite rarely

Other comments, opinions, etc?

#2 - 23 Oct 2013 14:42 - John Abbott

- Category set to Tidying

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 20

Once this matter is resolved, we can change the file TmpFactor into factor; strictly the Tmp prefix should have been applied to factorization.

#3 - 24 Oct 2013 14:56 - Anna Maria Bigatti

Currently JAA prefers method (A) because:

1. it's already impl'ed, so no extra work needed
2. it is easier to use (think of KISS paradigm)
3. the greater ease of applying STL algms will be appreciated only quite rarely

I agree that (B) is cleaner, but if we had chosen to break backward compatibility with CoCoA-4 there was a good reason ;-)

I see this point:

1. sometimes (often?) when we ask for a factorization we actually want to access the factors, and ignore the multiplicities. Method (A) makes this use easy and natural (just a simple vector of RingElem).
2. On the other hand (B) make it easy to access, one "factor+multiplicity", but I find it unlikely that we want to access the first f+m and ignore the others.

So I vote (again ;-) for (A)

#4 - 29 Oct 2013 12:28 - Anna Maria Bigatti

- Target version set to CoCoALib-0.99531

#5 - 29 Oct 2013 13:28 - Anna Maria Bigatti

- Status changed from In Progress to Feedback

#6 - 21 Mar 2014 14:18 - John Abbott

- Status changed from Feedback to In Progress

Bite the bullet!
I know what needs to be done; I just have to find the courage to do it (I keep hearing "if it ain't broke, don't fix it!")

#7 - 21 Mar 2014 14:31 - Anna Maria Bigatti

- Target version changed from CoCoALib-0.99531 to CoCoALib-0.99532

#8 - 24 Mar 2014 12:52 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 20 to 80

I have revised the interface of factorization.

The 3 main fields are now private, so I have implemented (read-only) accessor fns.

I have added a fn for "appending" a new factor-multiplicity pair, and another for updating the "remaining factor". These new fns check that their args are sensible.

I had implemented, but then deleted the following two mem fns:

```
const RingElem& myFactor(int i);
const long& myMultiplicity(int i);
```

I did this because in existing code there were calls such as facs.myFactors()[j] which is ugly/hard-to-read. However, the improvement was only slight: facs.myFactor(j). So in the end I returned to the previous interface, but changed the examples to use const references: for instance

```
const vector<RingElem>& facs = FactorInfo.myFactors();
...
cout << facs[j] << endl;
```

It is a bit of a nuisance to have to write the const-ref declarations, but the rest of the code gains considerably in terms of readability. I think this is a fair compromise.

I've updated documentation, tests, examples and C5 interpreter.

Waiting for Anna's approval to pass to "feedback".

#9 - 24 Mar 2014 13:08 - John Abbott

I've checked in everything.
For me all tests pass (even with max debugging). Also OK on my linux box (a netbook, >45 mins **without** debugging)

#10 - 02 Apr 2014 13:12 - John Abbott

- Status changed from Resolved to Feedback

- % Done changed from 80 to 90

I'm pretty happy with the revised design; changing state to "feedback".

After speaking to Bruns about the design, I have added some comments to the "Bug, Shortcomings, etc" section of the documentation.

#11 - 02 Apr 2014 16:52 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

As Anna says, we've released the code, we're convinced by the new design.
So closing.

#12 - 03 Apr 2014 11:54 - Anna Maria Bigatti

- Estimated time set to 8.00 h