CoCoALib - Feature #839

SparsePolyIter: make more compatible with STL

13 Jan 2016 14:11 - John Abbott

Status:	In Progress	**Start date:**	13 Jan 2016
Priority:	Low	**Due date:**	
Assignee:		**% Done:**	10%
Category:	Improving	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	0.90 hour

Description

Mario has suggesting making SparsePolyIter more compatible with C++ (and STL); for instance this would allow use of the "new" for ~~loop syntax, which Mario likes a lot :~~)

Related issues:

Related to CoCoALib - Design #1787: Iterator design: compatible with C++STL? ...	**In Progress**	**07 Mar 2024**

History

#1 - 13 Jan 2016 14:19 - John Abbott

As a quick reminder: the new for-loop syntax is like this:

```
vector<int> v; ...
for (const int& n : v)
{
  // loop body
}
```

The for-loop automatically calls the begin and end fns of the iterator type of v. It also automatically increments the iterator, and does an implicit operator* to get a reference to the value pointed to by the iterator.

Currently SparsePolyIter does not fit this model too well, but maybe it can be adapted. There should be no problem in providing begin and end fns. The hard part will be deciding what operator* will produce. Ideally it should produce something from which one can get either a coeff or a PP easily; this will not be entirely trivial since there is no existing object containing all relevant information (namely value of coeff and of PP, and also the owning ring/PPMonoid). It would be undesirable to have to create a new object for each term visited in the sparsepoly.

#2 - 13 Jan 2016 14:23 - John Abbott

A possible solution would be to return an "index" which can be converted into a usable value via a memfn of the polynomial. For instance it could look like this:

```
RingElem f; // belongs to a sparsepolyring
for (auto& term : f)
{
  cout << "Coeff is " << f.coeff(term) << endl;
  cout << "PP is "    << f.PP(term) << endl;
}
```

A disadvantage is that something nasty could happen if you pass the "index" to the wrong sparsepoly; moreover, detecting this at run-time would be non-trivial.

#3 - 13 Jan 2016 14:26 - John Abbott

Another idea is to have separate iterator types for the coeffs and the PPs; but the problem of not being able to refer to a pre-existing object remains.

Two separate iterators would also be awkward (*e.g.* both must be incremented in step with one another) in contexts where one wants to access both the coeff and PP -- probably most uses!

#4 - 13 Jan 2016 15:41 - Anna Maria Bigatti

interesting syntax :-)
looks quite horribly complicated to implement though :-(

#5 - 17 Nov 2016 13:36 - John Abbott

- Priority changed from Normal to Low

Here is a possible way to implement the idea in comment 2.

The "index" could be a pair of pointers: one pointer to the head of the polynomial (the head must contain the coeffring and PPM), the other a pointer to the internal "summand" structure. The first pointer would let us check whether the polynomial is the right one; the second pointer can then be "derefenced" to yield either the coeff or the PP.

A disadvantage of this approch (using raw pointer to internal summand object) is that something nasty (undefined behaviour, probably SEGV) could happen if the polynomial is modified while iterating over it.

This approach would also allow the following, simpler syntax:

```
RingElem f; // belongs to a sparsepolyring
for (auto& term : f)
{
  cout << "Coeff is " << coeff(term) << endl;
  cout << "PP is "    << PP(term) << endl;
}
```

#6 - 07 Mar 2024 19:51 - John Abbott

- Related to Design #1787: Iterator design: compatible with C++STL? Advancing beyond end? added

#7 - 07 Mar 2024 20:01 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

In comment 5 above, the disadvantage that an iterator could be invalidated by a structure-changing operation to the polynomial is already well-known in C++: even iterators on a std::vector can become invalid if certain operations are applied to the vector. So, it is indeed a disadvantage, but the same problem exists for most/all other structure iterators; therefore, I think it is not really grave.