Course C^{++}, Exercise List 10

Deadline: 02.06.2016

This exercise is about computer algebra and template classes. We will implement *multivariate polynomials*. These are polynomials over more than one variable, e.g.

$$1 + x + 3y + 4xy - 7x^2y.$$

We will implement them using a map. The map maps chains of variables of form $(v_1^{i_1}, \ldots, v_n^{i_n})$ to the numerical values associated to the variables. In the example above, we would have a map containing

$$() \rightarrow 1, \quad (x) \rightarrow 1, \quad (y) \rightarrow 3, \quad (x, y) \rightarrow 4, \quad (x^2, y) \rightarrow -7.$$

This requires a class for variable chains, with an order defined on it that can be used by `std::map`. Since there doesn't seem to exist a mathematical term, I just call them **varchains** in the rest of the exercise. You need some additional code from the course homepage.

1. The task of method `normalize( )` is to normalize the varchain. This means that **(1)** the chain is sorted by variable, **(2)** occurrences of $x^{i_1}.x^{i_2}$ are merged into $x^{i_1+i_2}$, **(3)** all occurrences of form x^0 are removed.

   Implement the `normalize()` method of `varchain`. You can use `sort( )`, defined in `algorithm` for sorting.

2. Complete the `compare` function of `varchain`. Of course, you can use the fact that **varchains** are always sorted.

3. One you have implemented `normalize( )`, it is trivial to implement a multiplication operator for `varchain`. You can just merge the vectors and normalize the result. It is not theorically optimal, but good enough for this exercise.

4. Now we can turn our attention to class `polynomial`.

   Class polynomial is implemented as a template `template<typename N> polynomial`, where `N` can be an arbitrary number type. I tried it with **int**, **double**, **bigint** and **rational**.

5. Implement addition and subtraction operators for `polynomial`. This is not difficult, because you can use `+=` and `-=` as starting point.

6. Implement polynomial multiplication

```
template< typename N >
polynomial<N>
operator * ( const polynomial<N> & pol1, const polynomial<N> & pol2 )
```

   This is easier than you probably think. My implementation is 6 lines long.

7. Test your implementation over a few of the given number types. How much is $(1 + x)^5$? How much is $(1 + x^2.y.z^3)^4$? And $(3 + x.y)^6$.

   There are mathematicians, who believe that $(1 + \frac{x}{N})^N$ converges to e^x for large N.

   You can test this by computing $(1 + \frac{x}{N})^N$ for some big N, and comparing the result to the Taylor expansion. You can use function `exptaylor< >` and subtract the result.

   You can use `rational` or `double`.

   Does the statement appear to be true?