

Course C++, Exercise List 9

Deadline: 26.05.2017

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 $(v_1^{i_1}, \dots, v_n^{i_n})$ to the numerical values associated to the variables. In the example above, the map consists of

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

Each pair in the map represents a *monomial*. We need a class for the first element of the pair, and an order defined on it that can be used by `std::map`. According to wikipedia, the first element of the pair can be called **power product**, so I will use this word 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 power product. This means that **(1)** the chain is sorted by variable, **(2)** occurrences of $x^{i_1} \cdot x^{i_2}$ are merged into $x^{i_1+i_2}$, **(3)** all occurrences of form x^0 are removed. Implement the `normalize()` method of `powerproduct`. You can use `sort()`, defined in `algorithm` for sorting, and after that merge repeated variables. Make sure that performance of `normalize()` is not worse than $n \cdot \log(n)$.

2. Complete the `compare` function of `powerproduct`. Of course, you can use the fact that `powerproducts` are always sorted.

3. Once you have implemented `normalize()`, it is trivial to implement a multiplication operator for `powerproduct`. You can just merge the vectors and normalize the result. It is not theoretically 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?