IWR, Heidelberg University — Summer Semester 2017

Exercise Sheet 12 — 11. 07. 2017

Exercises for the Lecture Series

“Object-Oriented Programming for Scientific Computing”

Dr. Ole Klein
`ole.klein@iwr.uni-heidelberg.de`

To be handed in on 18. 07. 2017 before the lecture

This is the 13th and final exercise sheet of the lecture series. Please note that there is only one week to hand in your results, due to the exam in the following week (25.7., 16h c.t.). All exercises on this sheet are bonus exercises that don't count towards the total points needed for admission. Instead, these exercises provide bonus points and an opportunity to train for the exam. Please also note that the content of this sheet is nevertheless relevant for the exam.

Exercise 1 Type Traits for Matrices

In a previous exercise we implemented the Frobenius norm for matrices, and there the return type was always `double`. We now want to extend the algorithm in order to make it work for very different types of numbers, using the concept of type traits. There are a few things to note:

- The return type depends primarily on the number type of the matrix.
- The norm calculation should also work for `complex<T>` and `Rational` (see first exercise sheet).
- The return type is not always the number type of the matrix, e.g. for complex numbers it is the underlying real type.
- Changes may also be necessary for the calculation of the squared absolute value of the entries.

Answer the following questions and implement:

1. What kind of variability is necessary in the algorithm?
2. Which traits would you want to introduce because of this?
3. Design test problems for matrices with integer entries, real entries, rational entries and complex entries (3×3 matrices are sufficient).
4. Implement the corresponding traits classes and adapt the function `frobeniusnorm`. Write specializations of the traits class, so that the calculation of the norm works for `double`, `float`, `int`, `Rational` and `complex<T>`.
5. Test your code using the test problems for the built-in types `double`, `float`, and `int`, the user-defined class `Rational`, and finally the complex number types `complex<double>` and `complex<float>`.

Base your algorithm on the one you handed in for the previous exercise if applicable. If the abstract matrix interface using iterators isn't available to you, you may use direct access to the entries of the matrices.

10 Bonus Points

EXERCISE 2 MATRIX NORM VIA POLICY

Depending on the application, you might want to choose between different matrix norms, and there is a wide selection available.

Let $A \in \mathbb{R}^{n\times n}$, and consider the following subset of matrix norms:

- Frobenius norm
$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} |a_{ij}|^2},$$
- Row sum norm
$$\|A\|_{\infty} = \max_i \sum_j |a_{ij}|,$$
- Total norm
$$\|A\|_{\mathrm{G}} = n \cdot \max_{i,j} |a_{ij}|$$

Write a function

```
template<class M, class P = FrobeniusNorm>
MatrixTraits<M>::realType matrixNorm (const M& A);
```

which is parameterized with a policy class `P` and thus capable to calculate all three given norms (and possibly much more). Here `realType` is the name you used in the previous exercise.

One can formulate these norms in a common way so that one iterates over all rows and for each row iterates over all of its entries. For each entry an operation `EntryOp` is executed and afterwards for the whole row an operation `RowOp`.

Pseudo code:

```
matrixNorm<P> (Matrix A)
{
    result = 0;

    foreach (row r of A)
    {
        row_result = 0;

        foreach (entry x of r)
        {
            row_result = P::EntryOp(x, row_result);
        }

        result = P::RowOp(row_result, result);
    }

    return result;
}
```

The policy class `P` should supply all necessary operators and information to compute the norm.

1. What information / operators must be provided by the policy class for the three norms defined above?
2. Which template parameters (if any) do the policy classes need?

3. Implement the generic function `matrixNorm`.
4. Implement the policy classes for all three given norms.
5. Test the classes with the following example:

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & -4 \end{pmatrix}, \qquad \|A\|_{\mathrm{F}} = 6, \qquad \|A\|_\infty = 6, \qquad \|A\|_{\mathrm{G}} = 12$$

6. The algorithm above may be inefficient, e.g. in the case of the Frobenius norm. What change to the algorithm (and the policies) would allow more flexibility and a faster implementation? Please answer in words and pseudo code as above, an actual implementation is not required.

Try to reuse as much code as possible from previous exercises, so you don't need to reimplement everything.

10 Bonus Points