IWR, Heidelberg University | Summer Semester 2015
Exercise Sheet 7 | 26. 05. 2015

Exercises for the Lecture Series

"Object-Oriented Programming for Scientific Computing"

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

To be handed in on 02. 06. 2015 before the lecture

Exercise 1 Matrices and Templates

In the lecture templates were presented as a technique of generic programming. They allow the development of algorithms regardless of underlying data structures.

In this exercise we will extend an existing implementation of a matrix class to a template class. Matrices are needed again and again in numerical software. In the lecture, an implementation for the number type `double` has already been presented, and on the lecture homepage you will find the class `MatrixClass`. Depending on the application you want, however, it may be useful to have matrices based on other types of numbers, for example `complex`, `float` or `int`.

Templates may reduce code redundancy, so that one has an implementation of `MatrixClass` that can be used for all the different types.

MatrixClass for `double` — Basic features:

- Matrix entries are stored as a vector of vectors:
  `std::vector<std::vector<double> > a_;`
- The parentheses operator allows access to individual entries:
  ```
  double& operator()(int i, int j);
  double operator()(int i, int j) const;
  ```
- The matrix provides the arithmetic operations of scaling and addition:
  ```
  MatrixClass& operator*=(double x);
  MatrixClass& operator+=(const MatrixClass& b);
  ```

Additionally there are several free functions implementing arithmetic operatons based on `*=` and `+=`:

- `operator*(const MatrixClass& a, double x);`
- `operator*(double x,const MatrixClass& a);`
- `operator+(const MatrixClass& a, const MatrixClass& b);`

The current version can be found on the lecture website in the following files:

- `matrix_double.h`
- `matrix_double.cc`
- `test_matrix_double.cc`

The first two contain the definition and implementation of the `MatrixClass` for `double`, while the third file contains a test program.

Exercises:

1. **Template classes and functions:** Change the implementation of `MatrixClass` that was provided to that of a template class, so that it can be used for different number types. Change the main function of the test program, so that the template variants are tested. The program should use all data types mentioned above, with `complex` referring to `std::complex<double>` and the required header being `<complex>`. Also test the matrix with your `Rational` class. For the `Print` functionality you have to define a free function `std::ostream& operator<< (std::ostream& str, const Rational& r)` that prints the rational number by first printing its numerator, then a "/" and then its denominator.

   Note: The free functions

   - `operator*(const MatrixClass& a, double x)`
   - `operator*(double x,const MatrixClass& a)`
   - `operator+(const MatrixClass& a, const MatrixClass& b)`

   have to be modified as well. Also note that the argument `double x` of the functions need not be of the same type as the matrix entries after templatization. Introduce further template parameters to allow for this. Your test program should also take this into account.

2. **Templates and inheritance:** Separate the templated class `MatrixClass` into two classes:

   - a class `BaseMatrixClass`, without numerical operators. This represents a matrix containing an arbitrary data type and has the following methods:
     - Constructor (as in `MatrixClass`)
     - `operator()(int i, int j)` for access to individual entries
     - `Resize` and `Print` methods (as in `MatrixClass`)
   - a numerical matrix class `NumMatrixClass` inheriting from `MatrixClass`. the numerical matrix class additionally provides the arithmetic operations mentioned above.

   Also modify the free functions accordingly. Your second test program does not need to test each number type again, an arithmetic test with `double` is enough. Instead, test your `BaseMatrixClass` with `string`s as data, and check wether you can create and use objects of type `BaseMatrixClass<NumMatrixClass<int> >`. Either add a `operator<<` as for `Rational` to `BaseMatrixClass`, or avoid calling its `Print()` method in this case.

Note: The first part of this exercise is worth $3/5$ of the total points. Please hand in two distinct test programs, one for the first part of the exercise, with name `test_matrix_temp.cc`, and one for the second part, with name `test_matrix_inher.cc`, and also use separate header files for the two exercise parts.

20 Points