Dirk Praetorius
Michele Ruggeri

Summer semester 2019
May 24, 2019

Scientific programming in mathematics

Exercise sheet 10

References, keyword `const`, operator overloading, dynamic memory allocation

Exercise 10.1. Write a class `Fraction` for the representation of fractions of the form $x = p/q$, where $p \in \mathbb{Z}$ and $q \in \mathbb{N}$. The numerator and the (positive) denominator are stored as `int`. Implement the following features:

- the standard constructor (without parameters), which sets $p = 0$ and $q = 1$,
- a constructor, which takes $p, q \in \mathbb{Z}$ as input parameters and stores the fraction p/q (use `assert` to ensure a valid input, i.e, $q \neq 0$, and, for the case $q < 0$, store the fraction $(-p)/|q|$),
- the destructor,
- the copy constructor,
- the assignment operator.

Templates for the structure of your implementation are provided by the classes `Complex` and `Vector` presented in the lecture notes (slides 276–279 and slides 292–296, respectively). Test your implementation appropriately!

Exercise 10.2. Extend the class `Fraction` from Exercise 10.1 by the method `void reduce()`, which reduces a fraction p/q to lowest terms, i.e., in the form p_0/q_0, where $p_0 \in \mathbb{Z}$ and $q_0 \in \mathbb{N}$ are coprime and satisfy $p_0/q_0 = p/q$. Use an algorithm of your choice to compute the greatest common divisor of numerator and denominator, e.g., the Euclidean algorithm (see slides 99–101 from the lecture notes) or an algorithm which computes the prime factorization of integers (see, e.g., Exercises 6.6–6.8). Test your implementation appropriately!

Exercise 10.3. Implement the type casting from `Fraction` (the class from Exercise 10.1) to `double`. Implement also the type casting from `double` to `Fraction`. For the second case, consider only the first 9 decimals and note that the return value should be in reduced form. Test your implementation appropriately!

Exercise 10.4. Overload the operators $+$, $-$, $*$, and $/$, in order to be able to calculate the sum, the difference, the product, and the quotient of two fractions stored as objects of type `Fraction` from Exercise 10.1. For $/$, use `assert` to avoid dividing by 0. For all cases, the return value should be in reduced form. Test your implementation appropriately!

Exercise 10.5. Overload the sign operator $-$, which, given $x \in \mathbb{Q}$ stored as `Fraction` from Exercise 10.1, returns the fraction $-x$. Overload the operator `<<` in order to be able to print to the screen a fraction p/q stored in an object `x` of type `Fraction` by typing `cout << x`. Test your implementation accurately!

Exercise 10.6. Overload the comparison operators `==`, `!=`, `<`, `<=`, `>`, and `>=` in order to be able to compare two fractions stored as objects of type `Fraction` from Exercise 10.1. Compare directly the numerator and the denominator of the fractions (you might need to determine a common denominator). Test your implementation appropriately!

Exercise 10.7. Write a class `FractionVector` for the storage of vectors in $\mathbb{Q}^n$. The class contains the length n (`int`) and the dynamic vector of the entries (`Fraction*`). Implement the following features:

- the constructor,
- the copy constructor,
- the assignment operator,
- the destructor,
- mutator methods to work with the class.

Test your implementation appropriately!

Exercise 10.8. Implement the method `void sort()` for the class `FractionVector` from Exercise 10.7, which sorts a vector in ascending order and overwrites the input vector with the sorted one. For example, the vector

$$x = \left(-\frac{1}{2}, \frac{5}{7}, \frac{1}{3}, \frac{0}{1}, \frac{11}{2}, -\frac{7}{8}\right) \in \mathbb{Q}^6$$

should be overwritten by the vector

$$x = \left(-\frac{7}{8}, -\frac{1}{2}, \frac{0}{1}, \frac{1}{3}, \frac{5}{7}, \frac{11}{2}\right).$$

Use the comparison operators from Exercise 10.6 and a sorting algorithm of your choice (see, e.g., one that you have already implemented to solve Exercise sheet 5). Test your implementation appropriately! What is the computational complexity of your implementation in terms of the vector length n? If sorting a vector of length $n = 10^2$ has a runtime of 0.5 seconds with your implementation, which runtime do you expect for $n = 10^3$? Justify your answers!