Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
21./22.12.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Extend the class `Fraction` from the lecture (slide 230) by the `public` method `void reduce()` that determines the reduced form of the fraction `numerator/denominator`. Use the *euclidean division algorithm*. Moreover, implement the method `setValue(string value)` that converts an arbitrary number, given as a string, into a fraction. For the implementation you can proceed as follows: First, find the decimal-point in the string and count the number of positions after the decimal-point. Then, erase the decimal-point from the string. The string now represents a natural number and can be converted into an `int` variable by use of the function `atoi`. This number is used for the numerator. Then, the denominator is set to 10^p, where $p \in \mathbb{N}$ is the number of positions after the decimal-point. Then, call the method `reduce()`. Finally, overload the method `setValue` in an appropiate way, so that `setValue(n)` for n of type `int` makes sense. How did you test your implementation? Save your source code as `fraction.{hpp,cpp}` into the directory `serie09`.
Hint: The method `find` of the class `string` allows you to find a specific character in the string, e.g., `int pos = value.find('.')` returns the position of the decimal-point in the string `value`. The call `value.erase(pos,k)`, erases `k` characters after the position `pos` in the string `value`. The function `atoi` from the standard library `cstdlib` converts a given string (in C-style) to an `int` variable. To get the string as `char *`, you can use the method `c_str()` of class `string`.

Aufgabe 9.2. Write a class `University`. This class should contain the members `numStudents`, `city`, and `name` as well as the methods `graduate`, and `newStudent`. If the method `graduate` is called, the number of students gets decreased by one, whereas if `newStudent` is called, the number of students increases by one. All data members should be declared as `private`. Therefore, you have to implement `get` and `set` methods. How did you test your implementation? Save your source code as `university.{hpp,cpp}` into the directory `serie09`.

Aufgabe 9.3. For the HR-department of the University it can be tedious to add and delete students one by one in their data. Therefore, overload the methods `graduate` and `newStudent` from the class `University` from Exercise 9.2, so that the number of graduating and beginning students can be a parameter of the methods. Moreover, write constructors which initialize your object with meaningful data. If the object is not initialized directly, then set `numStudents = 0`, `city = noWhere`, `name = noName`. Write a `plot`-routine to print the data of your object on screen. How did you test your implementation? Save your source code as `University.{hpp,cpp}` into the directory `serie09`.

Aufgabe 9.4. Write a structure `Matrix` to save quadratic $n \times n$ `double` matrices. Distinguish between fully-populated matrices (type `'F'`), lower triangle matrices (type `'L'`) and upper triangle matrices (type `'U'`). A lower triangular matrix L and an upper triangular matrix U have the following polulation structure:

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \qquad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

We thus have $u_{jk} = 0$, if $j > k$ and $\ell_{jk} = 0$, if $j < k$. A fully populated matrix should by stored in Fortran-Style- therefore columnwise in a dynamical vector with $n \cdot n$ entries. triangle-matrices should be stored in a vector with $\sum_{j=1}^{n} j = n(n+1)/2$ entries. Implement the following functionalities:

- Default constructor, which allocates a 0×0 matrix of the type `'F'`
- Constructor, which gets the type and the dimension as an input parameter
- Destructor
- `get` and `set`-methods for the matrix entries, the type and the dimension

The `get` and `set`-methods for the matrix entries depend on the type of the matrix. How did you test your implementation? Save your source code as `matrix.{hpp,cpp}` into the directory `serie09`.

Aufgabe 9.5. Extend the class `Matrix` from Exercise 9.4 by

- a method `scanMatrix(char typ, int n)` to read the type and the matrix $A \in \mathbb{R}^{n \times n}$ depending on the type from the keyboard,
- a method `printMatrix()`, which prints the matrix to the screen,
- a method `columnsumnorm()`, which computes and returns the column sum norm

$$\|A\| = \max_{k=0,\ldots,n-1} \sum_{j=0}^{n-1} |a_{jk}|,$$

- a method `rowsumnorm()`, which computes and returns the row sum norm

$$\|A\| = \max_{j=0,\ldots,n-1} \sum_{k=0}^{n-1} |a_{jk}|.$$

Note that for lower resp. upper triangular matrizes the methods can access only coefficients a_{jk} resp. a_{kj} with $0 \leq k \leq j \leq n-1$. How did you test your implementation? Save your source code as `matrix2.{hpp,cpp}` into the directory `serie09`.

Aufgabe 9.6. Let $U \in \mathbb{R}^{n \times n}$ be an upper triangular matrix such that $U_{jj} \neq 0$ for all $j = 0, \ldots, n-1$. Given $b \in \mathbb{R}^n$, there exists a unique $x \in \mathbb{R}^n$ such that $Ux = b$. Derive a formula to compute the solution $x \in \mathbb{R}^n$ of $Ux = b$ by using the formula for the matrix-vector product and the simplifications thereof which follows from the triangular structure of U. Implement a function which, given an upper triangular matrix $U \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$, computes the solution of the system $Ux = b$. Use the class `Matrix` from Exercise 9.4 for U and the class `Vector` from the lecture for b (cf. slides 249). Use `assert` to check that the dimensions match and that $U_{jj} \neq 0$ for all j. Then, write a main program to test your implementation accurately. How did you test your implementation? Speichern Sie den Source-Code unter `solveMatrixU.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.7. We consider the class `Matrix` from Exercise 9.4 and `Vector` from the lecture (cf. slides 249). Implement the method `solve` for the class `Matrix`, which solves the linear system $Ax = b$ by using the so-called *Gaussian elimination*. Consider a matrix $A \in \mathbb{R}^{n \times n}$ (type `Matrix`) and a right-hand side vector $b \in \mathbb{R}^n$ (typ `Vector`). The algorithm reads as follows:

- First of all, the matrix A is converted into an equivalent upper triangular matrix. Note that also the right-hand side vector b must be modified accordingly.
- The resulting system, characterized by an upper triangular matrix A, is then solved with Exercise 9.6.

In particular, during the first elimination step, an appropriate multiple of the first row of the matrix is subtracted from the remaining rows so to obtain a matrix of the form

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ 0 & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2} & \ldots & a_{nn} \end{pmatrix}.$$

In the second elimination step, an appropriate multiple of the second row of the matrix is subtracted from the remaining rows so to obtain a matrix of the form

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3} & \dots & a_{nn} \end{pmatrix}.$$

After $n-1$ elimination steps, one obtains an upper triangular matrix A. Use `assert` to ensure that, in the k-th elimination step, the condition $a_{kk} \neq 0$ is satisfied. Don't forget that also the right-hand side vector $b \in \mathbb{R}^n$ must be modified accordingly. Then you can use exercise 9.6 to solve the system $Ax = b$ where A is an upper triangular matrix. What is the computational cost of your implementation of the Gaussian elimination and why? To understand the algorithm, start with simple examples with $A \in \mathbb{R}^{2\times 2}$ and $A \in \mathbb{R}^{3\times 3}$. How did you test your implementation? Save your source code as `gauss.cpp` into the directory `serie09`.

Aufgabe 9.8. The Gaussian elimination algorithm from Exercise 9.7 fails when it happens that $a_{kk} = 0$ in the k-th elimination step. This can happen even when the linear system $Ax = b$ has a unique solution x. To avoid this, the algorithm is usually extended with the so-called *pivoting*:

- During the k-th step, choose amongst $a_{kk}, \dots, a_{nk}$ the element a_{pk} with the largest absolute value.
- Swap the k-th and the p-th row of A (and b).
- Perform the elimination step as before.

Implement for the class `Matrix` from Exercise 9.4 the method `gausspivot`, which computes the solution of the system $Ax = b$ following the aforementioned strategy. (It is possible to prove that the Gaussian elimination algorithm with pivoting can be successfully applied if and only if the linear system $Ax = b$ admits a unique solution. A proof of this result is given in the lecture on numerical mathematics.) How did you test your implementation? Save your source code as `gausspivot.cpp` into the directory `serie09`.