Dirk Praetorius,
Michele Ruggeri

Wintersemester 2015/16
09.-10.12.2015

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. Write a structure (data-type) `polynomial` for the storage of polynomials that are represented as $p(x) = \sum_{j=0}^{n} a_j x^j$. Note that you have to store the degree $n \in \mathbb{N}_0$ as well as the coefficient vector $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$. Write all necessary functions to work with this structure (`newPoly`, `delPoly`, `getPolyDegree`, `getPolyCoefficient`, `setPolyCoefficient`). Save your source code as `polynomial.c` into the directory `serie09`.

Aufgabe 9.2. The sum $r = p + q$ of two polynomials p, q is again a polynomial. Write a function `addPolynomials` that computes the sum r. For the storage of polynomials use the structure from Exercise 9.1. Additionally, write a main program that reads in two polynomials and computes the sum thereof. Save your source code as `addPolynomials.c` into the directory `serie09`.

Aufgabe 9.3. The product $r = pq$ of two polynomials $p(x) = \sum_{j=0}^{m} a_j x^j$ and $q(x) = \sum_{j=0}^{n} b_j x^j$ is again a polynomial. Write a function `prodPoly` that computes the product r and stores it in the structure from Exercise 9.1. At first, think about the degree of the polynomial r. Additionally, write a main program that reads in two polynomials and computes the product thereof. Test your code on a suitable example. Save your source code as `prodPoly.c` into the directory `serie09`.

Aufgabe 9.4. The k-th derivative $p^{(k)}$ of a polynomial p is again a polynomial. Write a function `differentiatePolynomial` that computes the k-th derivative of a polynomial. For the storage of polynomials use the structure from Exercise 9.1. Additionally, write a main program that reads in p and k, and prints out $p^{(k)}$. Test your code on a suitable example. Save your source code as `differentiatePolynomial.c` into the directory `serie09`.

Aufgabe 9.5. Write a structure `Matrix` to save quadratic $n \times n$ `double` matrices. Distinguish between fully-populated matrices (type `0`), 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. Write all the necessary functions to work with this structure ((`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Save your source code as `matrix.c` into the directory `serie09`. (Hint: The functions `getMatrixEntry` and `setMatrixEntry` depend on the type of the matrix.)

Aufgabe 9.6. Write a function `columnsumnorm.c`, which, for a given matrix $A \in \mathbb{R}^{n \times n}$, calculates and returns the absolute column sum norm

$$\|A\|_S := \max_{j=1,\dots,n} \sum_{i=1}^{n} |A_{ij}|$$

A is stored in the structure from Exercise 9.5. if A is a triangular matrix, exploit the population structure of A. Save your source code as `columnsumnorm` into the directory `serie09`. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 9.7. 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. Save your source code as `University.cpp` into the directory `serie09`.

Aufgabe 9.8. Write a class `Deposit` with methods `accountNumber`, `assets`, and `ratePerCent`. Moreover, implement `set` and `get` methods for the members `accountNumber`, `assets`. To change the assets, write a method `drawMoney` and `placeOnDeposit`. Note that with this deposit you are not allowed to draw more money than is given, i.e., the member `assets` must be positive. The rate per cent as well as the account number must also be positive. Finally, implement the method `calculateAssets`. Save your source code as `Deposit.cpp` into the directory `serie09`.