Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
07.12.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. Write a structure `cdouble` to store the real part $a \in \mathbb{R}$ and the imaginary part $b \in \mathbb{R}$ of a complex number $a + bi \in \mathbb{C}$ as `double` variables. The imaginary unit i satisfies the identity $i^2 = -1$; see

`https://en.wikipedia.org/wiki/Complex_number`.

Implement the functions

- `cDouble* newCDouble(double a, double b)`,
- `cDouble* delCDouble(cDouble* z)`

as well as the mutator functions

- `void setCDoubleReal(cDouble* z, double a)`,
- `double getCDoubleReal(cDouble* z)`,
- `void setCDoubleImag(cDouble* z, double b)`,
- sowie `double getCDoubleImag(cDouble* z)`.

How did you test your implementation? Save the source code, split into a header file `cdouble.h` and `cdouble.c`, into the directory `serie07`.

Aufgabe 7.2. Write the functions

- `cDouble* cAdd(cDouble* z, cDouble* w)`,
- `cDouble* cSub(cDouble* z, cDouble* w)`,
- `cDouble* cMult(cDouble* z, cDouble* w)`,
- `cDouble* cDiv(cDouble* z, cDouble* w)`,

which realize addition, subtraction, multiplication, and division of complex numbers. Moreover, implement the functions

- `double cNorm(cDouble* z)`, which computes and returns the modulus $|z| = \sqrt{a^2 + b^2}$ of a complex number $z = a + ib \in \mathbb{C}$,
- `cDouble* cConj(cDouble* z)`, which computes and returns the conjugate $\overline{z} = a - ib \in \mathbb{C}$ of a complex number $z = a + ib \in \mathbb{C}$.

Use the structure `cDouble` from Exercise 7.1. In particular, access the elements of the structure by using the appropriate functions. Write a main program, which reads two complex numbers $w, z \in \mathbb{C}$ from the keyboard and prints to the screen the quantities $|w|$, $|z|$, $w + z$, $w - z$, wz, and w/z (provided that $z \neq 0$). How did you test your implementation? Save your source code as `carithmetik.c` into the directory `serie07`.

Aufgabe 7.3. Write a structure `CPoly` for the storage of polynomials, where the coefficients are complex numbers, i.e., $p(x) = \sum_{j=0}^{n} a_j x^j$ with $a_j \in \mathbb{C}$. The structure should contain the degree $n \in \mathbb{N}$ and the coefficients $(a_0, \dots, a_n) \in \mathbb{C}^{n+1}$. Use the structure from Exercise 7.1. Moreover, implement the functions `newCPoly`, `delCPoly`, `getCPolyDegree`, `getCPolyCoefficient`, and `setCPolyCoefficient`. How did you test your implementation? Save your source code as `cpoly.c` into the directory `serie07`.

Aufgabe 7.4. Write a function `addCpolynomials` that computes the sum $r = p + q$ of two complex polynomials p, q and returns r. Use the structure from Exercise 7.3. Moreover, write a main program that reads in two polynomials p, q and prints out the sum $r = p + q$. How did you test your implementation? Save your source code as `addcpoly.c` into the directory `serie07`.

Aufgabe 7.5. Write a structure data-type `SquareMatrix` for the storage of quadratic matrices $A \in \mathbb{R}^{n\times n}$. The structure contains the dimension $n \in \mathbb{N}$ and the entries given as `double*`, i.e., the entries of the matrix have to be stored columnwise. In contrast to the usual indexing in C (e.g., the indexing considered in the lecture), let the indices for the matrix entries a_{jk} of your structure `SquareMatrix` go from $j, k = 1$ to n (as it is common in mathematics). Moreover, implement the necessary functions to work with this structure, i.e., `newSquareMatrix`, `delSquareMatrix`, `getSquareMatrixN`, `getSquareMatrixEntry`, and `setSquareMatrixEntry`. How did you test your implementation? Save the source code, split into a header file `squarematrix.h` and `squarematrix.c`, into the directory `serie07`.

Aufgabe 7.6. The Laplace formula states that, for each $j \in \{1, \dots, n\}$, it holds that

$$\det A = \sum_{k=1}^{n} (-1)^{j+k} \cdot a_{jk} \cdot \det A_{jk},$$

where a_{jk} are the entries of A and A_{jk} is the $(n-1)\times(n-1)$-submatrix of A obtained by removing the j-th row and the k-th column from A. Note that the determinant of a 1×1-matrix ($\in \mathbb{R}$) is the number itself. Write a recursive function `double detlaplace(SquareMatrix* A)`, which applies the Laplace formula to compute the determinant $\det(A)$ of a matrix $A \in \mathbb{R}^{n\times n}$. Use the structure `SquareMatrix` from Exercise 7.5. How did you test your implementation? Save your source code as `detlaplace.c` into the directory `serie07`.

Aufgabe 7.7. A matrix $A \in \mathbb{R}^{n\times n}$ admits a normalized LU-factorization $A = LU$ if

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

If A admits a normalized LU-factorization, it holds that

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{for } i = 1, \dots, n, \quad k = i, \dots, n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{for } i = 1, \dots, n, \quad k = i+1, \dots, n, \\ \ell_{ii} &= 1 \quad \text{for } i = 1, \dots, n. \end{aligned}$$

The remaining coefficients of $L, U \in \mathbb{R}^{n\times n}$ are zero. This can be easily shown by using the formula for the matrix-matrix product. Write a function `SquareMatrix* computeLU(SquareMatrix* A)`, which computes and returns the LU-factorization of A. To use the above formulae, compute the coefficients of L and U in an appropriate order (i.e., what you need must already be computed). Use the structure `SquareMatrix` from Exercise 7.5. Write a main program to test the function `computeLU` on a suitable example. How did you test your implementation? Save your source code as `computeLU.c` into the directory `serie07`.

Aufgabe 7.8. What is the system of floating-point numbers? Which parts does a floating-point number consist of? How can you determine its value? What is the meaning of `Inf`, `-Inf`, and `NaN`? What is a normalized floating-point number? What is an implicit leading bit? Which are the values of the largest and the smallest positive normalized floating point number in the `float`-system $\mathbb{F}(2, 24, -126, 127)$?