Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
07.12.2017

**Übungen zur Vorlesung**
**Einführung in das Programmieren für TM**

**Serie 7**

**Aufgabe 7.1.** Schreiben Sie einen Strukturdatentyp `cDouble`, in dem Realteil $a \in \mathbb{R}$ und Imaginärteil $b \in \mathbb{R}$ einer komplexen Zahl $z = a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Die imaginäre Einheit $i$ erfüllt die Eigenschaft $i^2 = -1$, siehe

`https://de.wikipedia.org/wiki/Komplexe_Zahl`.

Schreiben Sie Funktionen

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

sowie die vier Zugriffsfunktionen

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

Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, in das Verzeichnis `serie07`.

**Aufgabe 7.2.** Schreiben Sie Funktionen

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

die die Addition, die Subtraktion, die Multiplikation und die Division für komplexe Zahlen realisieren. Weiters schreiben Sie

- eine Funktion `double cNorm(cDouble* z)`, die den Betrag $|z| = \sqrt{a^2 + b^2}$ von $z = a + ib \in \mathbb{C}$ berechnet und zurückgibt,
- eine Funktion `cDouble* cConj(cDouble* z)`, die die Konjugierte $\overline{z} = a - ib \in \mathbb{C}$ von $z = a + ib \in \mathbb{C}$ berechnet und zurückgibt.

Verwenden Sie zur Speicherung die Struktur `cDouble` aus Aufgabe 7.1, und benutzen Sie beim Strukturzugriff nur die entsprechenden Zugriffsfunktionen. Schreiben Sie ein aufrufendes Hauptprogramm, in dem zwei komplexe Zahlen $w, z \in \mathbb{C}$ eingelesen werden und $|w|$, $|z|$, $w + z$, $w - z$, $wz$ sowie $w/z$ (falls $z \neq 0$) ausgegeben werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `carithmetik.c` in das Verzeichnis `serie07`.

**Aufgabe 7.3.** Schreiben Sie eine Struktur `CPoly` zur Speicherung von Polynomen mit komplexwertigen Koeffizienten, die bezüglich der Monombasis dargestellt sind, d.h. $p(x) = \sum_{j=0}^{n} a_j x^j$. Es sind also der Grad $n \in \mathbb{N}_0$ sowie der Koeffizientenvektor $(a_0, \dots, a_n) \in \mathbb{C}^{n+1}$ zu speichern. Verwenden Sie für die Darstellung der komplexwertigen Koeffizienten den Strukturdatentyp aus Aufgabe 7.1. Schreiben Sie die ferner die nötigen Zugriffsfunktionen `newCPoly`, `delCPoly`, `getCPolyDegree`, `getCPolyCoefficient` und `setCPolyCoefficient`. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `cpoly.c` in das Verzeichnis `serie07`.

**Aufgabe 7.4.** Schreiben Sie eine Funktion `addCpolynomials`, die die Summe $r = p + q$ zweier komplexer Polynome $p$ und $q$ (auch unterschiedlichen Grades) berechnet und zurückgibt. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 7.3. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem zwei Polynome $p, q$ eingelesen und die Summe $r = p + q$ ausgegeben werden. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `addcpoly.c` in das Verzeichnis `serie07`.

**Aufgabe 7.5.** Schreiben Sie einen Strukturdatentyp `SquareMatrix` zur Speicherung quadratischer Matrizen $A \in \mathbb{R}^{n\times n}$. Hierbei sollen die Einträge der Matrix spaltenweise als `double*`, sowie die Größe $n \in \mathbb{N}$ abgespeichert werden. Im Gegensatz zur üblichen Indizierung in C (bzw. zur Indizierung in der Vorlegung) soll die Indizierung der Matrixeinträge $a_{jk}$ in Ihrer Struktur `SquareMatrix` von $j, k = 1$ bis $n$ laufen (wie in der Mathematik üblich). Schreiben Sie außerdem alle nötigen Funktionen um mit dieser Struktur arbeiten zu können, d.h. implementieren Sie `newSquareMatrix`, `delSquareMatrix`, `getSquareMatrixN`, `getSquareMatrixEntry` und `setSquareMatrixEntry`. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code, aufgeteilt in Header-Datei `squarematrix.h` und `squarematrix.c`, in das Verzeichnis `serie07`.

**Aufgabe 7.6.** Der Laplacesche Entwicklungssatz für Determinanten besagt, dass für ein beliebiges $j \in \{1, \ldots, n\}$ gilt, dass

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

wobei $a_{jk}$ der $jk$-te Eintrag von $A$ und $A_{jk}$ die $(n-1) \times (n-1)$-Untermatrix von $A$ ist, die durch Streichen der $j$-ten Zeile und $k$-ten Spalte entsteht. Schreiben Sie eine rekursive Funktion `double detlaplace(SquareMatrix* A)`, die die Determinante $\det(A)$ einer Matrix $A \in \mathbb{R}^{n\times n}$ mit Hilfe des Laplaceschen Entwicklungssatzes berechnet und zurückgibt. Hierbei ist zu beachten, dass die Determinante einer $1 \times 1$-Matrix ($\in \mathbb{R}$) die Zahl selbst ist. Verwenden Sie die Struktur `SquareMatrix` aus Ausgabe 7.5. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `detlaplace.c` in das Verzeichnis `serie07`.

**Aufgabe 7.7.** Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

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

Wenn aber $A$ eine normalisierte LU-Zerlegung besitzt, so gilt

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \ldots, n, \quad k = i, \ldots, n,$$

$$\ell_{ki} = \frac{1}{u_{ii}} \left( a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \right) \quad \text{für } i = 1, \ldots, n, \quad k = i+1, \ldots, n,$$

$$\ell_{ii} = 1 \quad \text{für } i = 1, \ldots, n,$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Schreiben Sie eine Funktion `SquareMatrix* computeLU(SquareMatrix* A)`, die die LU-Zerlegung von $A$ berechnet und zurückgibt. Dazu überlege man, in welcher Reihenfolge man die Einträge von $L$ und $U$ berechnen muss, damit die angegebenen Formeln wohldefiniert sind (d.h. alles was benötigt wird, ist bereits zuvor berechnet worden). Verwenden Sie die Struktur `SquareMatrix` aus Ausgabe 7.5. Schreiben Sie ein aufrufendes Hauptprogramm, in dem Sie die Funktion `computeLU` an einen geeigneten Beispiel testen. Wie haben Sie Ihren Code auf Korrektheit getestet? Speichern Sie den Source-Code unter `computeLU.c` in das Verzeichnis `serie07`.

**Aufgabe 7.8.** Was ist ein Gleitkommazahlsystem? Aus welchen Bestandteilen setzt sich eine Gleitkommazahl zusammen? Wie bestimmt man daraus ihren Wert? Was verbirgt sich hinter den Symbolen `Inf`, `-Inf` und `NaN`? Was ist eine normalisierte Gleitkommazahl? Was ist ein implizites erstes

Bit? Welchen Wert haben die größte und die kleinste positive normalisierte Gleitkommazahl im `float`-Gleitkommazahlsystem $\mathbb{F}(2, 24, -126, 127)$?