Dirk Praetorius
Marcus Page

Sommersemester 2011
11. April 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Aufgabe 6.1*. Schreiben Sie einen Strukturdatentyp `polynomial` zur Speicherung von Polynomen, die bezüglich der Monombasis dargestellt sind, d.h. $p(x) = \sum_{j=0}^{n} a_j x^j$. Es ist also der Grad $n \in \mathbb{N}_0$ sowie der Koeffizientenvektor $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$ zu speichern. Schreiben Sie alle nötigen Funktionen, um mit dieser Struktur arbeiten zu können (`newPoly`, `delPoly`, `getPolyDegree`, `getPolyCoefficient`, `setPolyCoefficient`). Speichern Sie den Source-Code unter `polynomial.c` in das Verzeichnis `serie06`.

Aufgabe 6.2*. Die Summe $r = p + q$ zweier Polynome p, q ist wieder ein Polynom. Schreiben Sie eine Funktion `addPolynomials`, die die Summe r berechnet. Zur Speicherung verwende man die Struktur aus Aufgabe 6.1. Binden Sie den Code aus Aufgabe 6.1 mittels `#include "cdouble.c"` ein. Zum Test schreibe man eine Funktion, die zwei Polynome einliest und deren Summe ausgibt. Speichern Sie den Source-Code unter `addPolynomials.c` in das Verzeichnis `serie06`.

Aufgabe 6.3*. Schreiben Sie einen Strukturdatentyp `cdouble`, in dem Realteil und Imaginärteil einer Zahl $a + bi \in \mathbb{C}$ jeweils als `double` gespeichert werden. Schreiben Sie Funktionen `newCDouble`, `delCDouble` sowie die vier Zugriffsfunktionen `setCDoubleReal`, `getCDoubleReal`, `setCDoubleImag` sowie `getCDoubleImag`. Speichern Sie den Source-Code, aufgeteilt in Header-Datei `cdouble.h` und `cdouble.c`, ins Verzeichnis `serie06`.

Aufgabe 6.4*. Schreiben Sie Funktionen, die die Addition, die Subtraktion, die Multiplikation und die Division für komplexe Zahlen $a + bi \in \mathbb{C}$ realisieren. Verwenden Sie zur Speicherung die Struktur aus Aufgabe 6.3, 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 $w + z$, $w - z$, $w \cdot z$ sowie w/z ausgegeben werden. Binden Sie den Code aus Aufgabe 6.3 mittels `#include "cdouble.c"` ein. Die Signaturen der Funktionen sollen von der Form `cdouble* addCDouble(cdouble* w, cdouble* z, cdouble* output)` sein. Falls `output = NULL` ist, soll neuer Speicher für `z+w` allokiert werden, andernfalls ist `z+w` int `output` zu speichern. Speichern Sie den Source-Code unter `carithmetik.c` in das Verzeichnis `serie06`.

Aufgabe 6.5. Schreiben Sie eine Struktur `Matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `'F'`) auch untere (Typ `'L'`) und obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

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

als obere bzw. untere Dreiecksmatrix. Mathematisch formuliert, gilt also $u_{jk} = 0$ für $j > k$ bzw. $\ell_{jk} = 0$ für $j < k$. Eine vollbesetzte Matrix werde als dynamische Matrix der Größe $n \times n$ gespeichert. Dreiecksmatrizen sollen entsprechend effizient gespeichert werden (orientieren Sie sich hierbei an Aufgabe 5.3). Schreiben Sie die Funktionen, um mit dieser Struktur arbeiten zu können (`newMatrix`, `delMatrix`, `getMatrixDimension`, `getMatrixType`, `getMatrixEntry`, `setMatrixEntry`). Speichern Sie den Source-Code unter `matrix.c` in das Verzeichnis `serie06`. Achten Sie darauf, dass viele dieser Funktionen vom Matrixtyp abhängen.

Aufgabe 6.6. Die *Spaltensummennorm* einer Matrix $M \in \mathbb{R}^{n \times n}$ ist definiert als

$$\|M\|_1 := \max_j \sum_{i=1}^{n} |m_{ij}|.$$

Schreiben Sie eine Funktion `matrixNorm`, welche die Spaltensummennorm realisiert. Verwenden Sie hierzu Die Datenstruktur aus Aufgabe 6.5. Orientieren Sie sich an der Zeilensummennorm aus der Vorlesung. Speichern Sie den Source-Code unter `matrixNorm.c` in das Verzeichnis `serie06`.

Aufgabe 6.7. Schreiben Sie eine Funktion `solve`, die Gleichungssysteme der Form

$$Lx = b \quad \text{bzw.} \quad Ux = b$$

für Dreiecksmatrizen löst. Hierbei soll die Datenstruktur aus Aufgabe 6.5 verwendet werden. Die Funktion soll sowohl untere, als auch obere Dreieckssysteme lösen können und den Typ jeweils aus der Struktur auslesen. Die Ausgabe soll ein `Vektor` im Format aus der Vorlesung sein. Speichern Sie den Source-Code unter `solveMe.c` in das Verzeichnis `serie06`.

Aufgabe 6.8. Schreiben Sie eine Funktion `matrixvector` zur Berechnung des Matrix-Vektor-Produkts $Ax \in \mathbb{R}^n$, wobei die Matrix $A \in \mathbb{R}^{n \times n}$ in der Datenstruktur aus Aufgabe 6.5 gespeichert werde. Der Vektor $x \in \mathbb{R}^n$ sei in der Datenstruktur aus der Vorlesung gespeichert. Schreiben Sie die Funktion möglichst effizient, d.h. eventuelle Struktur (Dreiecksmatrix!) von A soll ausgenutzt werden. Speichern Sie den Source-Code unter `matrixvector.c` in das Verzeichnis `serie06`.

Aufgabe 6.9. Schreiben Sie ein Funktion `whatAmI`, die eine Matrix aus einer Datei einliest, den Typ (`'F', 'L', 'U'`) überprüft und schließlich entsprechend abspeichert. Sie können davon ausgehen, dass die Größe der Matrix vorab bekannt ist und daher als Variable im Programm übergeben werden kann. Orientieren Sie sich hierbei an der Funktion `loadMatrix` aus der Vorlesung. Speichern Sie den Source-Code unter `whatAmI.c` in das Verzeichnis `serie06`.

Aufgabe 6.10. Schreiben Sie Ihre Funktion `dec2bin` aus Aufgabe 5.2 so um, dass nun die gekürzte Binärdarstellung zurückgegeben wird. Beispielsweise würde also für $n = 77$ die Zahlenfolge `1 0 0 1 1 0 1` ausgegeben. Verwenden Sie hierfür die Struktur `Vektor` aus der Vorlesung. Speichern Sie den Source-Code unter `dec2bin2.c` in das Verzeichnis `serie06`.