Samuel Ferraz-Leite, Dirk Praetorius
Christoph Überhuber

Sommersemester 2008
30. April 2008

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Die Aufgaben mit Stern (*) sind bis zur nächsten Übung vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung und mir als zusätzliche Grundlage für den Prüfungsstoff in den schriftlichen Tests. Kopieren Sie bitte den Source-Code in ein Unterverzeichnis `serie06` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit dem* `gcc` *compiliert werden können.*

Aufgabe 51*. Man schreibe eine Struktur `matrix` zur Speicherung von quadratischen Matrizen $A \in \mathbb{R}^{n\times n}$ mit Einträgen vom Typ `double`. In der Struktur sollen die Einträge A_{ij}, die Dimension n sowie der Typ der Matrix gespeichert werden. Berücksichtigen Sie die folgenden beiden Typen: (i) vollbesetzte Matrix (`type=0`) und (ii) symmetrische Matrix (`type=1`), d.h. $A_{ij} = A_{ji}$ für alle $i, j = 0, \ldots, n-1$.

- Eine vollbesetzte Matrix soll spaltenweise in Form eines (dynamischen) Vektors der Länge n^2 gespeichert werden, d.h. wie in Fortran.
- Bei einer symmetrischen Matrix werden nur die oberen Einträge A_{ij} mit $i \le j$ in einem (dynamischen) Vektor der Länge $\frac{n(n+1)}{2} = \sum_{k=1}^{n} k$ gespeichert.

Ferner schreibe man Funktionen `allocMatrix(n,type)` und `freeMatrix(A)`, die eine Matrix allokieren/initialisieren bzw. freigeben, sowie Funktionen `getMatrixDimension(A)` und `getMatrixType(A)`, um Dimension und Typ der Matrix A auszulesen.

Aufgabe 52*. Überlegen Sie sich, wie eine Matrix A im Datenformat aus Aufgabe 51 im Speicher abgelegt ist, d.h. an welcher Stelle des Datenvektors A_{ij} steht. Schreiben Sie Funktionen`getMatrixEntry(A,i,j)` und `setMatrixEntry(A,i,j,Aij)`, die den Eintrag A_{ij} zurückliefern bzw. setzen. Testen Sie die Struktur und die Zugriffsfunktionen mithilfe eines geeigneten Hauptprogramms.

Aufgabe 53*. Man schreibe eine Struktur `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 $a_0, \ldots, a_n \in \mathbb{R}$ zu speichern. Schreiben Sie alle nötigen Funktionen, um mit dieser Struktur arbeiten zu können (`newPoly`, `delPoly`, `getPolyDegree`, `getPolyCoefficient`, `setPolyCoefficient`). Speichern Sie den Source-Code, aufgeteilt in Header-File `polynomial.h` und `polynomial.c`, ins Verzeichnis `serie06`.

Aufgabe 54*. Die k-te Ableitung $p^{(k)}$ eines Polynoms p ist wieder ein Polynom. Man programmiere eine Funktion `differentiatePolynomial`, die zu gegebenem p und $k \in \mathbb{N}$ die Ableitung $p^{(k)}$ berechnet. Das Ergebnispolynom $p^{(k)}$ soll minimalen Grad $n \in \mathbb{N}$ haben, d.h. es gilt $a_n \neq 0$ für den höchsten Koeffizienten von $p^{(k)}$. Zur Speicherung verwende man die Struktur aus Aufgabe 53. Zum Test schreibe man eine Funktion, die p und k einliest und $p^{(k)}$ ausgibt.

Aufgabe 55. Gegeben seien eine Matrix $A \in \mathbb{R}^{n\times n}$,

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix},$$

und eine rechte Seite $b \in \mathbb{R}^n$. Man löse das Gleichungssystem $Ax = b$ mit dem *Gauß'schen Eliminationsverfahren*. Dies ist gerade das Vorgehen, wenn man ein lineares Gleichungssystem händisch löst:

- Zunächst bringt man die Matrix A auf obere Dreiecksform, in dem man die Unbekannten eliminiert. Gleichzeitig modifiziert man die rechte Seite b.
- Das entstandene Gleichungssystem mit oberer Dreiecksmatrix A löst man wie in Aufgabe 42.

Im ersten Eliminationsschritt zieht man geeignete Vielfache der ersten Zeile von den übrigen Zeilen ab und erhält dadurch eine Matrix der Form

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

Im zweiten Eliminationsschritt zieht man nun geeignete Vielfache der zweiten Zeile von den übrigen Zeilen ab und erhält eine Matrix der Form

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

Nach $n-1$ Eliminationsschritten erhält man also eine obere Dreiecksmatrix A. Man berücksichtige, dass auch die rechte Seite $b \in \mathbb{R}^n$ geeignet modifiziert werden muss und mache sich das Vorgehen zunächst an einem Beispiel mit $A \in \mathbb{R}^{2\times 2}$ sowie $A \in \mathbb{R}^{3\times 3}$ klar. Man schreibe eine Funktion `gauss`, die die Lösung von $Ax = b$ berechnet. Verwenden Sie die Strukturdatentypen aus den Aufgaben 51 und 43.Man schreibe ferner ein Hauptprogramm in dem die Dimension $n \in \mathbb{N}$, die Matrix A und die rechte Seite b eingelesen und die Lösung x ausgegeben werden.

Aufgabe 56. Das Gauß'sche Eliminationsverfahren scheitert, falls im k-ten Schritt $a_{kk} = 0$ gilt, auch wenn das Gleichungssystgem $Ax = b$ eine eindeutige Lösung x besitzt. Deshalb kann man das Verfahren um eine sogenannte *Pivot-Suche* erweitern:

- Im k-ten Schritt wählt man aus $a_{kk}, \dots, a_{nk}$ das betragsgrößte Element a_{pk}.
- Dann vertauscht man die k-te und die p-te Zeile von A (und b).
- Schließlich führt man den Eliminationsschritt aus wie zuvor.

Wenn man im Gauß-Verfahren mit Pivot-Suche die Zeilen im Eliminationsschritt *wirklich* vertauscht (d.h. Speicher kopiert), führt dies auf unnötig viele Operationen und entsprechend lange Laufzeit des Programms. Es empfiehlt sich daher, die Vertauschung nur *virtuell* durchzuführen: Man startet mit einem Buchhaltervektor $\pi = (1, \dots, n)$. Im Vertauschungsschritt vertauscht man lediglich $\pi(p)$ mit $\pi(k)$. Im Source-Code sind jetzt die Zeilenindizes, d.h. der erste Index von a_{jk} sowie der Index von b_j geeignet zu modifizieren. Man schreibe eine Funktion `gausspivot`, die das eben beschriebene Verfahren implementiert. Ferner schreibe man ein aufrufendes Hauptprogramm, in dem die Dimension $n \in \mathbb{N}$ sowie die Matrix A und der Vektor b eingelesen und die Lösung x ausgegeben werden.

Aufgabe 57. Die Summe $r = p+q$ zweier Polynome ist wieder ein Polynom. Man schreibe eine Funktion `addPolynomials`, die die Summe r berechnet. Das Ergebnispolynom r soll minimalen Grad $n \in \mathbb{N}$ haben, d.h. es gilt $a_n \neq 0$ für den höchsten Koeffizienten von r. Zur Speicherung verwende man die Struktur aus Aufgabe 53. Schreibe Sie ein aufrufendes Hauptprogramm, das zwei Polynome einliest und deren Summe ausgibt.

Aufgabe 58. Das Produkt $r = pq$ zweier Polynome p, q ist wieder ein Polynom. Man programmiere eine Funktion `multiplyPolynomials`, die das Produkt r berechnet. Das Ergebnispolynom r soll minimalen Grad $n \in \mathbb{N}$ haben, d.h. es gilt $a_n \neq 0$ für den höchsten Koeffizienten von r. Zur Speicherung verwende man die Struktur aus Aufgabe 53. Man schreibe ein aufrufendes Hauptprogramm, das zwei Polynome einliest und deren Produkt ausgibt.

Aufgabe 59. 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 in der Datei `cdouble.c` ins Verzeichnis `serie06`.

Aufgabe 60. 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 59, 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 die Code aus Aufgabe 59 mittels `#include ''cdouble.c''` ein.