Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Wintersemester 2008/09
19. November 2008

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 6

Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung und uns als zusätzliche Grundlage für den Prüfungsstoff in den schriftlichen Tests. Kopieren Sie bitte den Source-Code auf Ihren Account auf der `lva.student.tuwien.ac.at` *in ein Unterverzeichnis* `serie06`*. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit dem* `gcc` *compiliert werden können. In den folgenden Übungsaufgaben sollen* ***Strukturen*** *geübt werden.*

Aufgabe 51*. Man schreibe eine Struktur `vector` zur Speicherung von `double`-Vektoren der Länge n. Die Struktur enthalte neben der Dimension n den dynamischen Datenvektor. Ferner schreibe man die zugehörigen Funktionen `newVector`, `delVector`, `getVectorLength`, `getVectorEntry`, `setVectorEntry`. Speichern Sie den Source-Code unter `vektor.c` in das Verzeichnis `serie06`.

Aufgabe 52*. Man schreibe eine Struktur `matrix` zur Speicherung von quadratischen $n \times n$ `double` Matrizen, in der neben vollbesetzten Matrizen (Typ `0`) auch untere (Typ `'L'`) und obere (Typ `'U'`) Dreiecksmatrizen gespeichert werden können. Dabei bezeichnet man Matrizen

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

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 im Fortran-Format spaltenweise als dynamischer Vektor der Länge $n \cdot n$ gespeichert. Dreiecksmatrizen sollen in einem Vektor der Länge $\sum_{j=1}^{n} j = n \cdot (n+1)/2$ gespeichert werden. Man schreibe 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`.

Im Folgenden sollen zur Speicherung die Strukturen aus den Aufgaben 51-52 verwendet werden.

Aufgabe 53*. Man schreibe eine Funktion `matrixvektor` zur Berechnung des Matrix-Vektor-Produkts $Ax \in \mathbb{R}^n$. Schreiben Sie die Funktion möglichst effizient, d.h. eventuelle Struktur (Dreiecksmatrix!) von A soll ausgenutzt werden. Speichern Sie den Source-Code unter `matrixvektor.c` in das Verzeichnis `serie06`.

Aufgabe 54*. Schreiben Sie eine Funktion `matrixmatrix`, die für gegebene Matrizen A und B das Produkt $C = AB$ berechnet. Sind A und B Dreiecksmatrizen vom selben Typ, so soll bei der Berechnung auf die Struktur Rücksicht genommen werden. — Das Produkt zweier oberer bzw. zweier unterer Dreiecksmatrizen ist wieder eine obere bzw. eine untere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an die Matrizen die Indizes vereinfache. Mithilfe der hergeleiteten Formel soll die Funktion so programmiert werden, dass nur die nichttrivialen Einträge von C berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden. Speichern Sie den Source-Code unter `matrixmatrix` in das Verzeichnis `serie06`.

Aufgabe 55. Man schreibe eine Struktur `Cvector` zur Speicherung von Vektoren $x \in \mathbb{C}^n$. Die Struktur enthalte neben der Dimension n den dynamischen Datenvektor. Benutzen Sie zur Darstellung komplexer Zahlen die Struktur aus Aufgabe 43. Ferner schreibe man die zugehörigen Funktionen `newCVector`, `delCVector`, `getCVectorLength`, `getCVectorEntry`, `setCVectorEntry`.

Aufgabe 56. Man schreibe eine Funktion `Cvektorvektor`, die das Skalarprodukt zweier komplexwertiger Vektoren berechnet. Verwenden Sie zur Speicherung Ihrer Vektoren die Struktur aus Aufgabe 56.

Aufgabe 57. Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

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

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

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

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

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

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. Man schreibe eine Funktion `computeLU`, 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).

Aufgabe 58. Man schreibe eine Funktion `solve`, die die Lösung x des linearen Gleichungssystem $Ax = b$ berechnet. Dabei soll die Struktur des Gleichungssystems (Dreiecksmatrix!) ausgenutzt werden. Ist A keine Dreiecksmatrix, berechne man die LU-Zerlegung und löse die Faktorisierung $LUx = b$ in zwei Schritten: (1) $Ly = b$, (2) $Ux = y$.

Aufgabe 59. Einem C-Programm können bei Aufruf aus der Unix-Shell heraus Parameter übergeben werden. Die Signatur der Funktion `main` lautet dazu

```
main(int nargin, char** varargin)
```

Dabei ist `nargin` die Anzahl der Parameter und `varargin` ist ein Vektor von Strings der Parameter. Man beachte, dass der Name des Executable als erster Eingabeparameter verstanden wird, d.h. Aufruf eines Programms `beispiel` in der Form

- `beispiel 1 2 3 4 5`

aus der Shell heraus führt auf `nargin = 6`, `varargin[0] =` ″`beispiel`″, `varargin[1] =` ″`1`″ etc. Insbesondere gilt also stets `nargin` ≥ 1. Man schreibe ein C-Hauptprogramm, das bei Aufruf seinen Namen, die Anzahl aller echten Parameter sowie die Parameterliste ausgibt.

Aufgabe 60. Man schreibe ein C-Hauptprogramm, dem aus der Shell heraus ein Vektor $x \in \mathbb{R}^n$ in folgender Notation übergeben werden kann:

- `readvektor [ 1 2 3 4 ]`

Dabei kennzeichnen die eckigen Klammern den Beginn und das Ende eines Vektors. Der Einfachheit halber dürfen Sie annehmen, das vor und nach den eckigen Klammern ein Blank (Leerzeichen) steht. Eingabefehler sollen entsprechend ausgegeben werden, z.B. im Fall

- `readvektor [ 1 2 3 4`

bei dem die schließende Klammer fehlt. Der Vektor werde nach fehlerfreiem Einlesen am Bildschirm ausgegeben.

In der nächsten Woche wird am 26.11.2008 keine neue Übungsserie ausgegeben (Serie 7 folgt am 03.12.2008). In der Übung am 03/04.12.2008 wird stattdessen der C-Test besprochen!