Dirk Praetorius
Christoph Überhuber

Sommersemester 07
18. April 2007

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

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 `serie05` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit dem* `gcc` *compiliert werden können. In den folgenden Aufgaben sollen im wesentlichen* ***dynamische Arrays*** *sowie* ***Zählschleifen*** *geübt werden. Außer Verzweigungen und elementarer Arithmetik sind keine weiteren Elemente von C nötig.*

Aufgabe 41*. Man schreibe eine Funktion `maxcount`, die von einem Vektor $x \in \mathbb{R}^n$ das Maximum zurückliefert und die Anzahl, wie oft dieses im Vektor vorkommt. Ferner schreibe man ein aufrufendes Hauptprogramm, das die Länge $n \in \mathbb{N}$ und den Vektor $x \in \mathbb{R}^n$ einliest und das Ergebnis der Funktion ausgibt. Den Source-Code speichere man unter `maxcount.c` ins Verzeichnis `serie05`.

Aufgabe 42*. Schreiben Sie eine Bibliothek, die für dynamische Vektoren $x \in \mathbb{R}^n$ und Matrizen $A \in \mathbb{R}^{m \times n}$ die folgenden Funktionen beinhaltet: `mallocvector`, `freevector`, `reallocvector`, `mallocmatrix`, `freematrix` und `reallocmatrix`. Ferner sollen Dateifunktion `loadvector` und `loadmatrix` enthalten sein, die dynamische Vektoren und Matrizen aus einer Datei einlesen. Matrizen sollen hierbei dynamisch über `double**` realisiert werden wie in der Vorlesung. An arithmetischen Funktionen soll die Bibliothek Funktionen `matrixmatrix` und `matrixvector` zur Matrix-Matrix- und zur Matrix-Vektor-Multiplikation enthalten. Speichern Sie den Source-Code `bib.c` und das Header-File `bib.h` ins Verzeichnis `serie05`.

Aufgabe 43*. Eine Matrix $U \in \mathbb{R}^{n \times n}$, deren Einträge unterhalb der Hauptdiagonale gleich 0 sind, bezeichnet man als obere Dreiecksmatrix,

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix}$$

bzw. mathematisch formuliert: $u_{jk} = 0$ für $j > k$. Schreiben Sie eine Funktion `loadmatrixU`, die eine dynamische Matrix aus einer Datei einliest, überprüft, ob diese Matrix eine obere Dreiecksmatrix ist, und in diesem Fall U zurückgibt. Anderenfalls werde `NULL` zurückgegeben. Binden Sie bei der Implementierung die Funktionen aus Aufgabe 42 mittels `#include "bib.c"` ein. Speichern Sie den Source-Code unter `loadu.c` ins Verzeichnis `serie05`. Testen Sie Ihren Code, indem Sie ein zusätzliches Hauptprogramm schreiben, unter `testloadu.c` ins Verzeichnis `serie05` speichern und Beispieldateien anlegen.

Aufgabe 44*. Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ mit $u_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebenem $y \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Ux = y$. Man schreibe eine Funktion `solveU`, die x berechnet. Dabei soll auf die offensichtlichen Nulleinträge von U nicht zugegriffen werden, um den Rechenaufwand gering zu halten. Speichern Sie den Source-Code unter `solveu.c` ins Verzeichnis `serie05`. — Um den Algorithmus herzuleiten, schreibe man das Matrix-Vektor-Produkt $y = Ux$ komponentenweise für y_j mit $j = 1, \dots, n$ als Summe hin. Man überlege, wie die spezielle Gestalt von U die

Laufindizes der Summe vereinfacht und löse diese Gleichung nach x_j auf. Testen Sie Ihren Code, indem Sie ein zusätzliches Hauptprogramm schreiben, das die Matrix U und den Vektor y aus Dateien einliest und mit dem berechneten Ergebnis x das Matrix-Vektor-Produkt $\widetilde{y} = Ux$ berechnet. Bei korrekter Implementierung sollte $y \approx \widetilde{y}$ gelten.

Aufgabe 45. Man schreibe eine Funktion `matrixvectorU`, die das Matrix-Vektor-Produkt $y = Ux$ mit einer oberen Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ berechnet. Dabei soll auf die offensichtlichen Nulleinträge von U nicht zugegriffen werden, um den Rechenaufwand gering zu halten.

Aufgabe 46. Das Produkt $U = AB$ zweier oberer Dreiecksmatrizen $A, B \in \mathbb{R}^{n\times n}$ ist wieder eine obere Dreiecksmatrix. Man beweise diese Aussage zunächst mathematisch, indem man sich die Formel für das Matrix-Matrix-Produkt hinschreibe und mittels der Voraussetzung an A und B die Indizes vereinfache. Danach schreibe man eine Funktion `matrixmatrixU`, die die Produktmatrix berechnet und zurückgibt. Dabei sollen natürlich nur die nicht-trivialen Einträge von U, d.h. U_{jk} für $j \leq k$, berechnet werden. Ferner soll auf die trivialen Einträge von A und B nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel.

Aufgabe 47. Eine Matrix $L \in \mathbb{R}^{n\times n}$, deren Einträge oberhalb der Hauptdiagonale gleich 0 sind, bezeichnet man als untere Dreiecksmatrix,

$$L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \dots & \dots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

bzw. mathematisch formuliert: $\ell_{jk} = 0$ für $j < k$. Schreiben Sie eine Funktion `loadmatrixL`, die eine dynamische Matrix aus einer Datei einliest, überprüft, ob diese Matrix eine untere Dreiecksmatrix ist, und in diesem Fall L zurückgibt. Anderenfalls werde `NULL` zurückgegeben. Binden Sie bei der Implementierung die Funktionen aus Aufgabe 42 mittels `#include "bib.c"` ein.

Aufgabe 48. Man schreibe eine Funktion `matrixvectorL`, die das Matrix-Vektor-Produkt $y = Lx$ mit einer unteren Dreiecksmatrix L berechnet. Dabei soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten.

Aufgabe 49. Gegeben sei eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit $\ell_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebenem $y \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Lx = y$. Man schreibe eine Funktion `solveL`, die x berechnet. Dabei soll auf die offensichtlichen Nulleinträge von L nicht zugegriffen werden, um den Rechenaufwand gering zu halten.

Aufgabe 50. Man implementiere eine Funktion `eratosthenes`, die das *Sieb des Eratosthenes* realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl `nmax` errechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Der Algorithmus sieht folgendermaßen aus:

- Man legt eine Liste (Vektor) $\texttt{prim} = (2, \dots, \texttt{nmax}) \in \mathbb{N}^{\texttt{nmax}-1}$ an.
- Man streicht aus der Liste alle Vielfachen der ersten Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Der Rückgabevektor `prim` soll am Ende (gekürzt auf minimale Länge) alle Primzahlen $\leq$ `nmax` enthalten (Sie müssen zusätzlich die Länge des Rückgabevektors zurückgeben!). Realisieren Sie das Streichen geeignet, z.B. indem Sie die entsprechenden Einträge auf 0 setzen.