Dirk Praetorius, Samuel Ferraz-Leite
Christoph Überhuber

Sommersemester 2010
20. April 2010

# Übungen zur Vorlesung
# Einführung in das Programmieren für TM

## Serie 5

*Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche vorzubereiten. Kopieren Sie die Source-Codes vor der Übung auf Ihren Account auf der* `lva.student.tuwien.ac.at` *und überprüfen Sie, ob diese mit dem* `gcc` *compiliert werden können. In den folgenden Übungsaufgaben sollen* ***dynamische Arrays*** *und* ***Strukturen*** *geübt werden.*

**Aufgabe 5.1*.** Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$, d.h. $U \in \mathbb{R}^{n\times n}$ mit $U_{jk} = 0$ für $j > k$. Sei $u_{jj} \neq 0$ für alle $j = 1, \dots, n$, dann existiert zu gegebenem $y \in \mathbb{R}^n$ ein eindeutiges $x \in \mathbb{R}^n$ mit $Ux = y$. Schreiben Sie eine Funktion `solveU`, die $x$ berechnet. Dabei soll auf die offensichtlichen Nulleinträge von $U$ nicht zugegriffen werden, um den Rechenaufwand gering zu halten. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die Dimension $n \in \mathbb{N}$ sowie die Einträge der Matrix $U$ und der rechten Seite $y$ eingelesen und die Lösung $x$ ausgegeben werden. Speichern Sie den Source-Code im Verzeichnis `serie05` unter dem Namen `solveU.c`.
— Um den Algorithmus herzuleiten, schreibe man das Matrix-Vektor-Produkt $y = Ux$ komponentenweise für $y_j$ mit $j = 1, \dots, n$ als Summe. Überlegen Sie, wie die spezielle Gestalt von $U$ die Laufindizes der Summe vereinfacht und lösen Sie diese Gleichung nach $x_j$ auf.

**Aufgabe 5.2*.** Schreiben Sie 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. Speichern Sie den Source-Code im Verzeichnis `serie05` unter dem Namen `eratosthenes.c`.

**Aufgabe 5.3*.** 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 im Verzeichnis `serie05` unter `polynomial.c`.

**Aufgabe 5.4*.** Das Produkt $r = pq$ zweier Polynome $p(x) = \sum_{j=0}^{m} a_j x^j$ und $q(x) = \sum_{j=0}^{n} b_j x^j$ ist wieder in Polynom. Schreiben Sie eine Funktion `prodPoly`, die das Produktpolynom $r$ berechnet und in der Struktur aus Aufgabe 5.3 speichert. Überlegen Sie sich zunächst, welchen Grad das Polynom $r$ hat und wie sich die Koeffizienten berechnen lassen. Schreiben Sie ein aufrufendes Hauptprogramm, in dem $p$ und $q$ eingelesen und $r = pq$ ausgegeben wird. Speichern Sie den Source-Code im Verzeichnis `serie05` unter `prodPoly.c`.

**Aufgabe 5.5.** Eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ (vgl. Aufgabe 5.01) hat potentiell $\frac{n(n+1)}{2} = \sum_{j=1}^{n} j$ nicht-triviale Einträge. Schreiben Sie eine Struktur `matrixU`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten $U_{ij}$ in einem dynamischen Vektor der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Schreiben Sie die entsprechenden Zugriffsfunktionen (`newMatrixU`, `delMatrixU`, `getMatrixUN`, `getMatrixUij`, `setMatrixUij`), und überlegen Sie sich zuvor, an welcher Stelle $u_\ell$ im dynamischen Vektor ein Eintrag $U_{ij}$ gespeichert werden soll.

**Aufgabe 5.6.** Schreiben Sie eine Funktion `mvmU`, die die Matrix-Vektor-Multiplikation mit einer oberen Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$ realisiert. Die Matrix $U$ sei dabei in der Struktur aus Aufgabe 5.5 gespeichert, der Vektor in der Struktur aus der Vorlesung. Überflüssige Multiplikationen mit Nulleinträgen der Matrix $U$ sollen aus Effizienzgründen vermieden werden.

**Aufgabe 5.7.** Schreiben Sie eine Funktion `primfaktoren`, die für eine natürliche Zahl $n \in \mathbb{N}$ deren Primfaktoren bestimmt und als Vektor $p \in \mathbb{N}^k$ zurückgibt. Die Koeffizienten $p_j$ des Vektors $p \in \mathbb{N}^k$ sind also Primzahlen, und es gilt $n = \prod_{j=1}^{k} p_k$. Um eine Liste aller möglichen Primfaktoren zu erhalten, verwende man das Sieb des Eratosthenes aus Aufgabe 5.2.

**Aufgabe 5.8.** Schreiben Sie eine Funktion `evalDiffPoly`, die für ein gegebenes Polynom $p$, eine Ableitungsordnung $k \in \mathbb{N}_0$ und einen Punkt $x \in \mathbb{R}$ den Funktionswert $p^{(k)}(x)$ zurückgibt. Dabei soll das Polynom $p^{(k)}$ nicht explizit gebildet und gespeichert werden. Verwenden Sie für das Polynom $p$ die Struktur aus Aufgabe 5.3. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem der Grad $n$, das Polynom $p$, die Ableitungsordnung $k$ und der Punkt $x$ eingelesen werden und $p^{(k)}(x)$ ausgegeben wird.

**Aufgabe 5.9.** Das Produkt $C = AB$ zweier Matrizen $A \in \mathbb{R}^{\ell\times m}$ und $B \in \mathbb{R}^{m\times n}$ ist eine Matrix $C \in \mathbb{R}^{\ell\times n}$, deren Einträge durch $C_{ik} = \sum_{j=1}^{m} A_{ij}B_{jk}$ gegeben ist. Das Produkt $U = AB$ zweier oberer Dreiecksmatrizen $A, B \in \mathbb{R}^{n\times n}$ ist wieder eine obere Dreiecksmatrix $U \in \mathbb{R}^{n\times n}$. 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 nichttrivialen Einträge von $U$, d.h. $U_{jk}$ für $j \le k$, berechnet werden. Ferner soll auf die trivialen Einträge von $A$ und $B$ nicht zugegriffen werden, d.h. man verwende die anfangs hergeleitete Formel. Alle Matrizen sollen in der Struktur aus Aufgabe 5.5 gespeichert werden.

**Aufgabe 5.10.** Gegeben sei ein Vektor $x \in \mathbb{R}^n$ und eine Schranke $k \in \mathbb{N}$. Man schreibe eine Funktion `cut`, die aus $x$ alle Glieder $x_j$ mit $|x_j| > k$ streicht und den gekürzten Vektor zurückgibt. Dies kann auf verschiedene Arten erfolgen:

- ohne Zusatzspeicher: Man geht den Vektor $x$ von vorne durch, streicht die entsprechenden Glieder und kopiert die nachfolgenden Glieder um eine Stelle nach vorne. Am Ende muss $x$ geeignet gekürzt werden.
- ohne Zusatzspeicher: Man geht den Vektor $x$ von hinten durch, streicht die entsprechenden Glieder und kopiert die nachfolgenden Glieder um eine Stelle nach vorne. Am Ende muss $x$ geeignet gekürzt werden.
- mit Zusatzspeicher: Man legt Zusatzspeicher für $y$ in der Länge von $x$ an und kopiert $x(j)$ nach $y$, falls $|x(j)| \le k$ gilt. Am Ende muss $y$ geeignet gekürzt und $x$ gelöscht werden.
- mit Zusatzspeicher: Man geht zunächst $x$ durch und zählt die Anzahl der Glieder $x(j)$ mit $|x(j)| \le k$. Man legt dann $y$ mit geeigneter Länge an und kopiert $x(j)$ nach $y$, falls $|x(j)| \le k$ gilt. Am Ende muss $x$ gelöscht werden.

Wie wird sich die Laufzeit dieser Varianten verhalten? Welche ist wohl die schnellste, welche die langsamste (zumindest wenn $x$ vergleichsweise lang ist)?