Dirk Praetorius
Marcus Page

Sommersemester 2011
4. April 2011

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 5

Aufgabe 5.1*. Schreiben Sie eine Funktion `minmaxmean`, die von einem gegebenem Vektor $x \in \mathbb{N}^n$ das Minimum, das Maximum und den Mittelwert $\frac{1}{n}\sum_{j=1}^{n} x_j$ berechnet und geeignet zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor $x \in \mathbb{N}^n$ einliest und Minimum, Maximum und Mittelwert ausgibt. Die Länge $n \in \mathbb{N}$ des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `minmaxmean` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `minmaxmean.c` in das Verzeichnis `serie05`.

Aufgabe 5.2*. Schreiben Sie eine Funktion `dec2bin`, die zu einer natürlichen Zahl $0 \leq n < 256$ die Binärdarstellung berechnet und zurückgibt (als dynamischer Vektor). Es sollen die Koeffizienten $a_i \in \{0, 1\}$ für $i = 0, \ldots, 7$ ermittelt werden, sodass $n = \sum_{i=0}^{7} a_i 2^i$ gilt. Beispielsweise gebe die Funktion für $z = 77$ die Zeichenfolge `0 1 0 0 1 1 0 1` aus. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem n eingelesen und `dec2bin` aufgerufen werden. Die Binärdarstellung soll schliesslich ausgegeben werden. Speichern Sie den Source-Code unter `dec2bin.c` in das Verzeichnis `serie05`.

Aufgabe 5.3*. Eine Matrix $L \in \mathbb{R}^{n \times n}$, deren Einträge oberhalb der Hauptdiagonale gleich 0 sind, bezeichnet man als untere Dreiecksmatrix,

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

bzw. mathematisch formuliert: $\ell_{jk} = 0$ für $j < k$. Dieses Format soll nun sinnvoll, d.h. ohne die überflüssigen Nullen gespeichert werden. Schreiben Sie hierfür Funktionen `mallocLmatrix` und `freeLmatrix`, welche den nötigen Speicher allokieren (und initialisieren), bzw. freigeben. Schreiben Sie zusätzlich eine Funktion `readLmatrix`, die eine untere Dreiecksmatrix von der Tastatur einliest und entsprechend zurückgibt. Ausserdem ist eine Funktion `mvm` zu schreiben, die eine Matrix-Vektor-Multiplikation einer Matrix im L-Matrix-Format mit einem beliebigen Vektor v realisiert. Testen Sie ihr Programm anhand einiger Beispiele. Speichern Sie den Source-Code unter `matrixfun.c` in das Verzeichnis `serie05`.

Aufgabe 5.4*. Schreiben Sie eine Funktion `solveLmatrix`, die ein Gleichungssytem der Form $Lx = b$, wobei L eine untere Dreiecksmatrix ist, löst und den Ergebnisvektor x zurückgibt. Ein aufrufendes Hauptprogramm soll L und b von der Tastatur einlesen und die Lösung x ausgeben. Speichern Sie den Source-Code unter `solveLmatrix.c` in das Verzeichnis `serie05`.

Aufgabe 5.5. *Bubble-Sort* ist ein ineffizienter, aber kurzer Sortier-Algorithmus: Man vergleicht aufsteigend jedes Element eines Arrays x_j mit seinem Nachfolger x_{j+1} und —falls notwendig— vertauscht

die beiden. Nach dem ersten Durchlauf muss zumindest das letzte Element bereits am richtigen Platz sein. Der nächste Durchlauf muss also nur noch bis zur vorletzten Stelle gehen, usw. Wie viele geschachtelte Schleifen braucht dieses Vorgehen? Schreiben Sie eine Funktion `bubblesort`, die einen gegebenen Vektor $x \in \mathbb{R}^n$ mittels Bubble-Sort aufsteigend sortiert, d.h. $x_1 \leq x_2 \leq \cdots \leq x_n$, und zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor x einliest und in sortierter Reihenfolge ausgibt. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie05`.

Aufgabe 5.6. Schreiben Sie eine verbesserte Variante von Bubble-Sort, bei der man sich die Stelle des letzten Tausches merkt. Ab dieser Stelle müssen die Daten ja bereits sortiert sein. Der nächste Durchlauf braucht also nur noch bis zu dieser Array-Position zu gehen. Speichern Sie den Source-Code unter `bubblersortNew.c` in das Verzeichnis `serie05`.

Aufgabe 5.7. Schreiben Sie eine Funktion `unique`, die einen gegebenen Vektor $v \in \mathbb{N}^m$ sortiert und alle mehrfach auftretenden Einträge entfernt, d.h. zum Beispiel `unique(3 3 1 7 9 7 2) = (1 2 3 7 9)`. Achten Sie darauf, den Speicher entsprechend neu zu allokieren. Schreiben Sie ferner ein aufrufendes Hauptprogramm, welches v von der Tastatur einliesst und `unique(v)` ausgibt. Speichern Sie den Source-Code unter `unique.c` in das Verzeichnis `serie05`.

Aufgabe 5.8. Die Funktion `unique` aus Aufgabe 5.7 ist von der Form `pointer = unique(pointer)`. Ändern Sie die Implementierung auf die Form `void unique(&pointer)` ab. Erklären Sie den Unterschied. Speichern Sie den Source-Code unter `unique2.c` in das Verzeichnis `serie05`.

Aufgabe 5.9. Schreiben Sie eine Funktion `invertString`, die eine gegebene Zeichenkette erhält und in invertierter Reihenfolge zurückgibt. Orientieren Sie sich hierbei an der Funktion `stringcopy` aus der Vorlesung. Speichern Sie den Source-Code unter `invertString.c` in das Verzeichnis `serie05`.

Aufgabe 5.10. Schreiben Sie eine Funktion `cutoff`, die aus einem gegebenen Vektor $v \in \mathbb{R}^n$ alle Einträge entfernt, die grösser als eine gewisse Schranke c sind. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem v und c von der Tastatur eingelesen werden, und der Ergebnisvektor ausgegeben wird. Speichern Sie den Source-Code unter `cutoff.c` in das Verzeichnis `serie05`.