Dirk Praetorius
Michele Ruggeri
Bernhard Stiftner

Sommersemester 2016
26.04.2016

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 7

Aufgabe 7.1. Schreiben Sie eine Funktion `checkoccurrence`, die einen String s und einen Buchstaben b übernimmt und zurückgibt wie oft b in s vorkommt. Dabei zähle man sowohl das Vorkommen als Großbuchstabe als auch das Vorkommen als Kleinbuchstabe. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem s und b eingelesen und checkoccurrence aufgerufen werden. Testen Sie ihr Programm entsprechend! Speichern Sie den Source-Code unter `checkoccurrence.c` in das Verzeichnis `serie07`.

Aufgabe 7.2. Schreiben Sie eine Funktion `palindrom`, welches überprüft ob ein Wort (String) ein Palindrom ist. Ein Palindrom ist ein Wort, welches von vorne und hinten gelesen gleich lautet, z.B.: Anna, Otto, Reliefpfeiler. Schreiben Sie ferner ein aufrufendes Hauptprogramm, welches das Wort einliest und `palindrom` aufruft. Testen Sie ihr Programm entsprechend! Speichern Sie den Source-Code unter `palindrom.c` in das Verzeichnis `serie07`.

Aufgabe 7.3. Jede Integerzahl $x \in \mathbb{N}_0$ lässt sich in der Form $x = \sum_{k=0}^{N-1} a_k 2^k$ darstellen mit Bits $a_k \in \{0, 1\}$, wobei $N \in \mathbb{N}$ die kleinste Zahl ist mit $x < 2^N$. Schreiben Sie eine Funktion `integer2bin`, die eine Integer Zahl x als Input nimmt und als Output einen (dynamischen) String der Bit-Kodierung $a_0 a_1 a_2 a_3 \dots$ zurückgibt. Beispielsweise gilt

```
0 für 0
1 für 1
01 für 2
11 für 3
001 für 4
101 für 5
011 für 6
111 für 7
etc.
```

Schreiben Sie ferner eine Funktion `bin2integer`, die die Bit-Kodierung einer Integer Zahl $x \in \mathbb{N}_0$ als (dynamischen) String übernimmt und den entsprechenden Integer-Wert $x \in \mathbb{N}_0$ zurückgibt. Testen Sie Ihren Code entsprechend! Speichern Sie den Source-Code unter `integerVSbin.c` in das Verzeichnis `serie07`.

Aufgabe 7.4. Schreiben Sie eine Bibliothek zur Verwaltung von *spaltenweise* gespeicherten $m \times n$-Matrizen. Implementieren Sie die folgenden Funktionen

- `double* mallocmatrix(int m, int n)`
  Allokieren von Speicher für eine spaltenweise gespeicherte $m \times n$- Matrix.
- `double* freematrix(double* matrix)`
  Freigeben des allokierten Speichers einer Matrix.
- `double* reallocmatrix(double* matrix, int m, int n, int mNew, int nNew)`
  Reallokieren und initialisieren von neuen Einträgen.

Speichern Sie die Funktionssignaturen in das Header-File `dynamicmatrix.h`. Schreiben Sie auch entsprechende Kommentare zu den Funktionen in das Header-File. In die Datei `dynamicmatrix.c` kommt dann die Implementierung der Funktionen. Verwenden Sie dynamische Arrays.

Aufgabe 7.5. Erweitern Sie die Bibliothek aus Aufgabe 7.4 um folgende Funktionalitäten

- `void printmatrix(double* matrix, int m, int n)`
  Gibt eine spaltenweise gespeicherte $m \times n$-Matrix als Matrix am Bildschirm aus. Die 2×3-Matrix `double matrix[6]={1,2,3,4,5,6}` soll wie folgt ausgegeben werden:

```
1 3 5
2 4 6
```

- `double* scanmatrix(int m, int n)`
  Allokiert Speicher für eine Matrix und liest die Koeffizienten der Matrix von der Tastatur ein.
- `double* cutOffRowJ(double* matrix, int m, int n, int j)`
  Schneidet die j-te Zeile aus einer $m \times n$-Matrix heraus.
- `double* cutOffColK(double* matrix, int m, int n, int k)`
  Schneidet die k-te Spalte aus einer $m \times n$-Matrix heraus.

Verwenden Sie dynamische Arrays. Schreiben Sie ein main-Programm um die Funktionen aus dieser Aufgabe und aus Aufgabe 7.4 zu testen.

Aufgabe 7.6. Für eine Matrix $A \in \mathbb{R}^{m\times n}$ ist die Zeilensummennorm durch

$$\|A\| = \max_{j=1,\dots,m} \sum_{k=1}^{n} |A_{jk}|$$

gegeben. Schreiben Sie eine Funktion `zeilensummennorm`, die die Zeilensummennorm einer Matrix A berechnet, die spaltenweise gespeichert ist. Schreiben Sie ein aufrufendes Hauptprogramm, in dem A eingelesen und $\|A\|$ ausgegeben wird. Verwenden Sie die Funktionen aus der Bibliothek aus Aufgabe 7.4 und Aufgabe 7.5. Speichern Sie den Source-Code unter `zeilensummennorm.c` in das Verzeichnis `serie07`.

Aufgabe 7.7. 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

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1,\dots,n, \quad k = i,\dots,n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{für } i = 1,\dots,n, \quad k = i+1,\dots,n, \\ \ell_{ii} &= 1 \quad \text{für } i = 1,\dots,n, \end{aligned}$$

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. Schreiben Sie 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). Schreiben Sie ein main-Programm, in dem Sie die Funktion `computeLU` an einen geeigneten Beispiel testen. Speichern Sie den Source-Code unter `computeLU.c` in das Verzeichnis `serie07`.

Aufgabe 7.8. Die Matrix $A \in \mathbb{R}^{n \times n}$ sei eine Tridiagonalmatrix, d.h.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & \\ & a_{3,2} & a_{3,3} & \ddots & \\ & & \ddots & \ddots & a_{n-1,n} \\ & & & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

wobei alle anderen Einträge von A Null sind, und besitze eine LU-Zerlegung. Mit Hilfe der Formeln in Aufgabe 7.7 überlege man sich Formeln für die Einträge von L und U in diesem speziellen Fall. Dann schreibe man eine Funktion `computeLU3` bei der keine unnötigen Rechenoperationen (d.h. Additionen/Multiplikationen von Null) durchgeführt werden und nur Einträge von L und U berechnet werden, die nicht Null sind. Testen Sie Ihren Code an einem geeigneten Beispiel.
Speichern Sie den Source-Code unter `computeLU3.c` in das Verzeichnis `serie07`.