Dirk Praetorius,
Marcus Page

Sommersemester 2012
18.03.2012

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

In dieser Übungsserie wiederholen Sie die Konzepte von Schleifen und Arrays. Achten Sie darauf, Ihren Code gut zu kommentieren. Das sollten Sie fortan immer tun.

Aufgabe 3.1*. (Schleifen) Die Quotientenfolge $(a_{n+1}/a_n)_{n\in\mathbb{N}}$ zur Fibonacci-Folge $(a_n)_{n\in\mathbb{N}}$,

$$a_0 := 1, \quad a_1 := 1, \quad a_n := a_{n-1} + a_{n-2} \quad \text{für } n \geq 2,$$

konvergiert gegen den goldenen Schnitt $(1+\sqrt{5})/2$. Insbesondere konvergiert die Differenz

$$b_n := \frac{a_{n+1}}{a_n} - \frac{a_n}{a_{n-1}}$$

gegen Null. Schreiben Sie eine Funktion `cauchy`, die zu gegebenem $k \in \mathbb{N}$ die kleinste Zahl $n \in \mathbb{N}$ mit $|b_n| \leq 1/k$ zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahl $k \in \mathbb{N}$ einliest und den zugehörigen Index $n \in \mathbb{N}$ ausgibt. Speichern Sie den Source-Code unter `fibonacci.c` in das Verzeichnis `serie03`.

Aufgabe 3.2*. (Schleifen, Arrays) In vielen mathematischen Bibliotheken werden Matrizen $A \in \mathbb{R}^{m\times n}$ spaltenweise gespeichert, d.h. in Form eines Vektors $a \in \mathbb{R}^{mn}$, wobei $a_{j+km} = A_{jk}$ gilt, wenn die Indizierung (wie in C üblich) bei 0 beginnt. Schreiben Sie eine `void`-Funktion `frobenius`, die die Frobeniusnorm einer spaltenweise gespeicherten Matrix $A \in \mathbb{R}^{m\times n}$ zurückgibt. Die Frobeniusnorm von $A = \{a_{ij}\}$ ist hierbei durch

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$

definiert. Die Dimensionen $m, n \in \mathbb{N}$ können Konstanten im Hauptprogramm sein, müssen aber als Parameter an die Funktion übergeben werden. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem A eingelesen, `frobenius` aufgerufen und der Wert der Frobeniusnorm ausgegeben wird. Speichern Sie den Source-Code unter `frobenius.c` in das Verzeichnis `serie03`.

Aufgabe 3.3*. (a) Was sind Verzweigungen, was sind Schleifen? Geben Sie ein Beispiel.
(b) Was ist eine rekursive Funktion? Geben Sie ein Beispiel, bei dem Rekursion durch induktives Vorgehen ersetzt werden kann.
(c) Wieso sollte Code immer gut dokumentiert werden?

Aufgabe 3.4*. (Schleifen, Arrays) 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 zurückgibt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Vektor $x \in \mathbb{N}^n$ einliest und an `minmaxmean` übergibt. Das Rückgabe der Funktion `minmaxmean` ist dann auszugeben. Die Länge $n \in \mathbb{N}$ des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `minmaxmean` ist für beliebige Länge n programmieren. Speichern Sie den Source-Code unter `minmaxmean.c` in das Verzeichnis `serie03`.

Aufgabe 3.5. (Schleifen, Arrays) Schreiben Sie eine Funktion `skalarprodukt`, die zu gegebenen Vektoren $x, y \in \mathbb{R}^n$ das Skalarprodukt $x \cdot y := \sum_{j=1}^{n} x_j y_j$ berechnet und zurückgibt. Ferner schreibe man ein aufrufendes Hauptprogramm, das die Vektoren x und y einliest und $x \cdot y$ ausgibt. Die Länge n der Vektoren soll eine Konstante im Hauptprogramm sein, die Funktion `skalarprodukt` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `skalarprodukt.c` in das Verzeichnis `serie03`.

Aufgabe 3.6. (Schleifen, Arrays) *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 ein gegebenes Array $x \in \mathbb{R}^n$ 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. Die Länge des Vektors soll eine Konstante im Hauptprogramm sein, die Funktion `bubblesort` ist für beliebige Länge n zu programmieren. Speichern Sie den Source-Code unter `bubblesort.c` in das Verzeichnis `serie03`.

Aufgabe 3.7. (Schleifen) Schreiben Sie einen naiven Primzahlentest in einer Funktion `isNotPrime`. Dieser soll (relativ ineffizient) wie folgt arbeiten: Sie lesen eine Zahl $p \in \mathbb{N}$ von der Tastatur ein. Ihr Programm soll nun für alle natürlichen Zahlen $a \leq p$ prüfen, ob a Teiler von p ist, d.h. $p \mod a = 0$. Gilt $p \mod a \neq 0$ für alle $a \leq p$, so ist p prim und es werde 0 zurückgegeben. Ansonsten soll der kleinste Teiler a zurückgegeben werden. Schreiben Sie ein aufrufendes Hauptprogramm, in dem p eingelesen und das Ergebnis von `isNotPrime` entsprechend ausgegeben wird. Speichern Sie den Source-Code unter `isNotPrime.c` in das Verzeichnis `serie03`. Wieso ist dieser Test nicht effizient? Was ließe sich verbessern?

Aufgabe 3.8. (Schleifen, Arrays) Schreiben Sie eine Funktion `eratosthenes`, die das Sieb des Eratosthenes realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl $n \in \mathbb{N}$ errechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Dabei soll für im Hauptprogramm konstantes $n \in \mathbb{N}$ ein Vektor $(1, \ldots, n)$ an die Funktion `eratosthenes` übergeben werden. In diesem Vektor sollen alle Nicht-Primzahlen durch 0 ersetzt werden („gestrichen“ = keine Primzahl). Der Algorithmus von Eratosthenes sieht folgendermaßen aus:

- Man streicht zunächst die 1.
- Man streicht aus der Liste alle Vielfachen der ersten nicht-gestrichenen Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfachen.

Die Funktion `eratosthenes` soll für beliebiges n programmiert werden. Das aufrufende Hauptprogramm soll schließlich alle Primzahlen $\leq n$ ausgeben. Speichern Sie den Source-Code unter `eratosthenes.c` in das Verzeichnis `serie03`.

Aufgabe 3.9. (Schleifen) Schreiben Sie eine Funktion `exponential`, die den Funktionswert $\exp(x)$ approximativ berechnet: Dazu berechnen Sie die Partialsumme

$$S_N(x) := \sum_{j=0}^{N} \frac{x^j}{j!},$$

wobei die Summationsgrenze $N \in \mathbb{N}$ durch das Kriterium

$$\left|\frac{x^{N+1}}{(N+1)!}\right| \leq \left|\frac{x^N}{N!}\right| \leq \varepsilon$$

für eine gegebene Toleranz $\varepsilon > 0$ bestimmt werde. Intern realisiere man die Berechnung der Summationsglieder $x^j/j!$ möglichst rechenökonomisch. Vergleichen Sie den Fehler $|S_N(x) - \exp(x)|$ für verschiedene Wahlen von $\varepsilon > 0$ und Auswertungspunkten $x \in \mathbb{R}$. Speichern Sie den Source-Code unter `exponential.c` in das Verzeichnis `serie03`.

Aufgabe 3.10. (Schleifen) Die Cosinus-Funktion hat die Darstellung $\cos(x) = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$. Wir betrachten die Partialsummen

$$C_n(x) = \sum_{k=0}^{n}(-1)^k \frac{x^{2k}}{(2k)!}.$$

Schreiben Sie eine Funktion `cos_`, die für gegebene $x \in \mathbb{R}$ und $\tau > 0$ den Wert $C_n(x)$ zurückliefert, sobald

$$|C_n(x) - C_{n-1}(x)|/|C_n(x)| \leq \tau \quad \text{oder} \quad |C_n(x)| \leq \tau$$

gilt. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem $x \in \mathbb{R}$ und $\tau > 0$ eingelesen werden. Neben dem berechneten Wert $C_n(x)$ sollen auch der korrekte Wert $\cos(x)$ und der absolute Fehler $|C_n(x) - \cos(x)|$ ausgegeben werden sowie der relative Fehler $|C_n(x) - \cos(x)|/|\cos(x)|$ im Fall $\cos(x) \neq 0$. Schreiben Sie die Funktion möglichst so, dass diese mit einer Schleife auskommt und dass x^{2k} und $(2k)!$ möglichst kostensparend realisiert werden. Man vermeide also insbesondere (vor- oder selbst implementierte) Funktionen zur Berechnung der Potenz oder der Faktoriellen. Speichern Sie den Source-Code unter `cos.` in das Verzeichnis `serie03`.