Dirk Praetorius,
Marcus Page,
Thomas Führer

Sommersemester 2013
19.03.2013

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 3

Aufgabe 3.1. Eine (möglicherweise nicht die beste) Art die Zahl π anzunähern liefert die als **Leibniz-Reihe** bekannte Formel:

$$\pi = \sum_{k=0}^{\infty} \frac{4(-1)^k}{2k+1}$$

Die n-te Partialsumme

$$P(n) = \frac{4(-1)^n}{2n+1} + P(n-1)$$

können wir also als rekursive Funktion auffassen, für die $\lim_{n\to\infty} P(n) = \pi$ gilt. Implementieren Sie eine Funktion `double P(int n);` die obige Funktionalität realisiert. Schreiben Sie auch ein Hauptprogramm, das obige Partialsumme für verschiedene Werte berechnet.
Hinweis: Für die Potenzen $(-1)^n$ benötigen sie **keine** Potenzfunktion. Überlegen Sie sich hierzu wie $(-1)^n$ von n abhängt. Speichern Sie den Source-Code unter `piRekursiv.c` in das Verzeichnis `serie03`.

Aufgabe 3.2. Die Fibonacci-Folge ist definiert durch $x_0 := 0$, $x_1 := 1$ und $x_{n+1} = x_n + x_{n-1}$. Schreiben Sie eine rekursive Funktion `fibonacciRek`, die zu gegebenem Index n das Folgenglied x_n zurückgibt. Schreiben Sie weiters eine nicht rekursive Funktion `fibonacci`, die dasselbe leistet, wobei die Berechnung aber über geeignete Schleifen realisiert wird. Welche der beiden Funktionen ist effizienter und warum? Speichern Sie den Source-Code unter `fibonacci.c` in das Verzeichnis `serie03`.

Aufgabe 3.3. Schreiben Sie eine `void`-Funktion `teiler`, die für eine gegebene Zahl $x \in \mathbb{N} := \{1, 2, 3, \dots\}$ ausgibt, ob diese durch 2, durch 3 oder durch 6 teilbar ist. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das den Integer x einliest und `teiler` aufruft. Speichern Sie den Source-Code unter `teiler.c` in das Verzeichnis `serie03`.

Aufgabe 3.4. Gegeben seien drei Punkte (x, y), (u, v) und (a, b) in $\mathbb{R}^2$. Schreiben Sie eine Funktion `punkte`, die überprüft, ob die 3 Punkte auf einer Geraden liegen. Schreiben Sie ferner ein aufrufendes Hauptprogramm, in dem die 6 Parameter eingelesen werden und das Resultat ausgegeben wird. Speichern Sie den Source-Code unter `punkte.c` in das Verzeichnis `serie03`.

Aufgabe 3.5. 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. 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. Welchen Aufwand hat ihr Algorithmus und warum? Speichern Sie den Source-Code unter `skalarprodukt.c` in das Verzeichnis `serie03`.

Aufgabe 3.6. Schreiben Sie eine Funktion `maxcompare`, die für zwei gegebene Vektoren $a, b \in \mathbb{R}^n$ zählt wie oft das Maximum $M = \max\{a_i, b_i \,|\, i = 1, \dots, n\}$ im Vektor a und b an der gleichen Stelle vorkommt. Zum Beispiel soll die Funktion für die Vektoren $a = (1.1, 4, 2e-4, 4, 4, 3, 4, -1.5)$ und $b = (2.2, 4, 4, 2e-5, 4, -1, 2.7, 4)$ den Wert 2 zurückgeben, da das Maximum $M = 4$ in beiden Vektoren an zwei Stellen nämlich $a_2 = b_2 = a_5 = b_5 = M$, gleichermaßen vorkommt. Welchen Aufwand hat ihr Algorithmus und warum? Speichern Sie den Source-Code unter `maxcompare.m` in das Verzeichnis `serie03`.

Aufgabe 3.7. *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 3.8. Schreiben Sie eine Funktion `rundung`, die für eine gegeben Zahl $x \in \mathbb{R}$ die Zahl $n \in \mathbb{N}$ zurückliefert, die x am nächsten liegt. Falls x genau in der Mitte zwischen zwei ganzen Zahlen liegt, werde die größere zurückgeliefert. Schreiben Sie ferner ein aufrufendes Hauptprogramm, das die Zahl x einliest und gerundet ausgibt. Speichern Sie den Source-Code unter `rundung.c` in das Verzeichnis `serie03`.